SMART SECURITY CAMERA USING MACHINE LEARNING

FOO JIA LIM

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electronic and Communication Engineering

Lee Kong Chian Faculty of Engineering and Science
Universiti Tunku Abdul Rahman

January 2019

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : _____

Name : Foo Jia Lim

ID No. : 1503274

Date : _____

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"SMART SECURITY CAMERA USING MACHINE LEARNING"** was prepared by **FOO JIA LIM** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electronic and Communication Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : _____

Supervisor : Dr. Tham Mau Luen

Date : _____

Signature : _____

Co-Supervisor : _____

Date : _____

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Tham Mau Luen for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement.

ABSTRACT

With the emergence of convolutional neural networks (CNN), the application of object classification and detection using deep learning is getting more and more common. However, the real-time performance of CNN on embedded system is poor, it need a few seconds to run an inference on embedded devices due to resource limitations. In this project, real-time object classification and detection on embedded system are realised with the use of Intel Movidius Neural Compute Stick (NCS). Combining the concept of Internet of Things (IoT), a cloud-based security system is built. This system records and uploads the video clips only when human is detected, and it subsequently notifies clients for video retrieval. For the deep learning algorithms, single shot detector (SSD) algorithm with a MobileNet architecture which is pre-trained with caffe framework is adopted. This project also aims to study the end-to-end performance of the system by evaluating detection accuracy, image preprocessing and inference time, video upload time and video qualities. Video upload time and push notification time are crucial when it comes to the performance of a real-time system.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
FB	Facebook
FN	False Negatives
FP	False Positives
fps	Frames per second
GPU	Graphic Processor Unit
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
IoT	Internet of Things
IP	Internet Protocol
ML	Machine Learning
mAP	Mean Average Precision
PSNR	Peak Signal to Noise Ratio
SSIM	Structural Similarity Index
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RPN	Region Proposal Network
SSD	Single Shot Detector
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
VQMT	Video Quality Measurement Tool
WHO	World Health Organization
YOLO	You Only Look Once

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In this section, background, applications, and some relevant terms of Artificial Intelligence (AI) are briefly discussed.

AI can be described as the ability of thinking and learning of machines. The word ‘artificial intelligence’ was first introduced by John McCarthy in 1955. John McCarthy, the father of AI, had defined seven development aspects of artificial intelligence in Dartmouth Conference 1956 - the first artificial intelligence conference. The seven aspects mentioned are simulation of complex activities of human brain on computer; instructions for a computer to interpret general language; arrangements of artificial neurons to form concept; ways to identify and diagnose problem complexity; ability of promoting personal development; stochasticity and inventiveness. (J. McCarthy, 1956).

It can be seen that a number of AI applications are emerging in today’s society. Several examples and applications of AI are introduced below.

Starting from 1st of February 2019, Sunway College Malaysia adopted facial recognition technology in their campus library. Faces of library users are scanned and verified whether he/she is authorized user at the entrance and exit of library. Comparing to commonly used methods like ID checking and barcode scanning, the fast and accurate facial recognition saves up time and helps to prevent identity fraud. (Sunway Campus Library Adopts Facial Recognition Technology, 2018).

Figure 1.1 A Sunway College student using facial recognition system (Sunway Campus Library Adopts Facial Recognition Technology, 2018).

Proactive detection, an AI tool launched by Facebook (FB) starting from September 2018. The tool is used at detecting people who might be at risk of committing suicide without human's report. Depending on severity, FB may pop messages or warnings to reach out to the person's friends and helpline contacts. When forthcoming of self-harm is detected, FB may contact local authorities. (Card, 2018). According to World Health Organization (WHO), death caused by suicide happens every 40 seconds. (Suicide data, 2019). From 1960s, the number of suicide cases of Malaysia has raised by 60 %, from information provided by Malaysian Psychiatric Association. (Dudley, 2018). With this tool, the lives can be saved as quickly as possible.

Figure 1.2 Prompts on suicidal post (Card, 2018)

How is AI developed? The answer is Machine Learning (ML). ML is the art of making computers to learn by itself, without being explicitly taught or instructed. Two definitions of ML are introduced. Arthur Samuel described ML as: "the field of study that gives computers the ability to learn without being explicitly programmed." (Samuel, 1959). After several decades, Tom Mitchell gave a more particular statement: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell, 1997). For example in predicting the probability of raining in Kuala Lumpur,

E = the past records of rain in Kuala Lumpur

T = the temperature of Kuala Lumpur

P = the probability that Kuala Lumpur will rain

In general, machine learning algorithm can be assigned to one of two broad classifications: supervised learning or unsupervised learning.

For supervised learning, the data set is well-labelled. The computer learns from the data set and make prediction on new examples, these processes are referred as training and inferring respectively. Supervised learning is further divided into “regression” and “classification”. Regression gives predictions of continuous value while classification gives predictions of discrete value. An example of regression is age prediction because age is continuous value instead of just ‘yes’ or ‘no’ answer. For classification, the example would be gender classification, the output of gender classification is discrete with just two values, either male or female.

Unsupervised learning is used to find implicit relations from an unlabelled data set. The structure of data set would be derived by clustering. For example, given a data set of humans with different races, they would be automatically sorted by skin colours, facial features, clothing and so on. Unsupervised data can be used on the field where the effect of its variables is not well-understand yet, for example, biological data analysis. The structures of brain tumours and neural networks were

able to be detected using unsupervised learning. (Chien-Chang Chen, Hung-Hui Juan, Meng-Yuan Tsai & Henry Horng-Shing Lu, 2018).

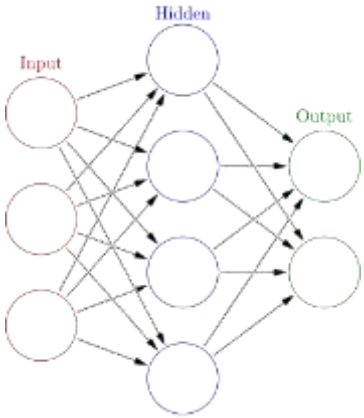

Figure 1.3 Structure of ANN

ML can be applied through different types of model, for examples, artificial neural network (ANN), Bayesian networks, support vector machines (SVM), and etc. ANN is the most popular machine learning model nowadays. ANN has the similar structure with biological neural network that constitutes animal brain. ANN is formed by interconnected artificial neurons. The connections between neurons act like biological synapses, to transmit signal from one neuron to another. Neurons could be activated by input signal. The behaviour of neurons is controlled by each activation function. Some common activation functions are Rectified Linear Unit (ReLU), step function, sigmoid function, tanh function and linear function.

ANN has many variants and convolutional neural network (CNN) is one of it. CNN is generally used for image classification.

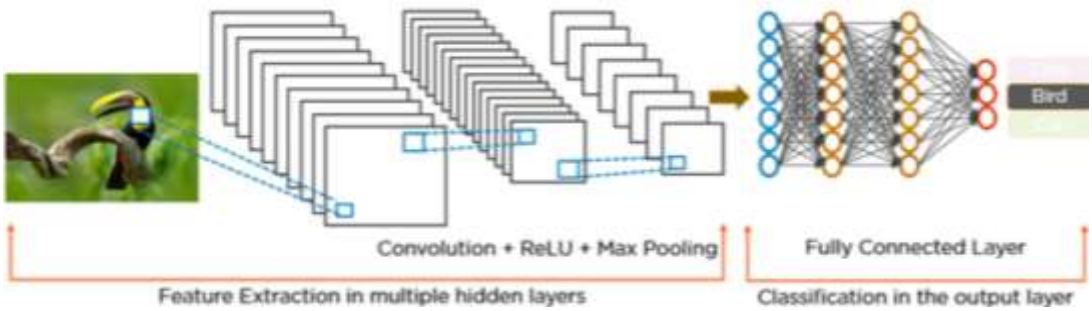

Figure 1.4 Structure of CNN (joshinishant2305, 2018).

CNN consists of four types of layer: firstly, convolution layer, secondly, ReLU layer following by pooling layer and fully connected layer. Convolutional layers are core layers that handle most computational tasks. It reads image spatially into width, height and depth. Width and height refer to the dimension of image while depth refer to the colour channels of the image. The depth has a size of 3 to cover all three colour channels which are red, green and blue. ReLU layers are actually layers of activation function, in these layers, the negative values of the images are suppressed and left only positive values. Pooling layers are used progressively reduce the size of samples to prevent overfitting. Overfitting refers to the amount of data is too much for a model could handle. Fully connected layers are layers where every neurons are interconnected. The function of fully connected layers is to generated desired amount of outputs. For CNN, input layer is often followed by convolutional layers lining with ReLU layers and pooling layers. Fully-connected layers are lastly added.

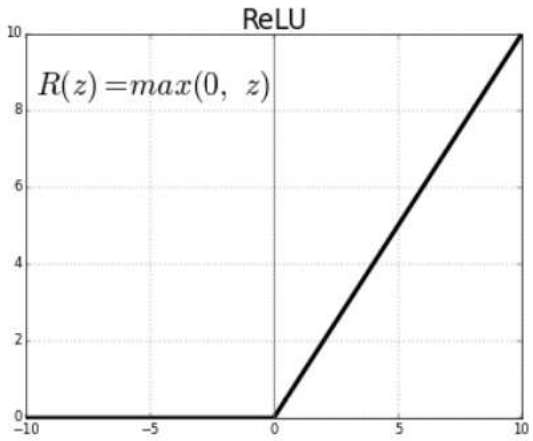

Figure 1.5 ReLU Function

1.2 Importance of the Study

The limited performance of embedded system on deploying machine learning applications has been overcome by using NCS. A real-time cloud-based security system is built on embedded platform with machine learning algorithm, and its end-to-end performance is analysed. This system detects human using machine learning algorithm and records when human is detected. As an implementation of IoT, this system alerts clients through Internet Protocol (IP) messaging apps and uploads

video clips to cloud for further retrieval. Existing literature generally analyse the performance of machine learning system regarding to detection accuracy and inference time, not the overall process including push notifications. In real-time applications, time has to be critically controlled in order to guarantee outcome within short time. In this project, performance of the proposed system is completely studied on speed, accuracy, real-time stability and video quality.

1.3 Problem Statement

Current IP camera is likely to raise false alarm in the event of brightness change, curtain flapping, pet running and etc. of curtain/kids and require huge storage for footage as the camera is recording at all the time. This project aims to reduce false alarm and shorten the length of video using machine learning algorithm. IP camera is best to be small in size, providing satisfactory performance and affordable in prices. CNN is usually utilized on home computer (PC) or portable computer (laptop) as it requires lots of computational power. Overcoming hardware constrains, a CNN application on a single-board computer which is small in size and low-power is required to build.

1.4 Aims and Objectives

- To create a smart security system using machine learning at least hardware and affordable price
- To integrate suitable machine learning model with camera and single-board computer
- To study end-to-end performance of the system

1.5 Scope and Limitation of the Study

The time performance – frame per second (fps) of the system is successfully improved to 3.5 which is 70 times of its original performance. Due to hardware limitations, the system is unable to achieve the same fps as what a laptop does.

1.6 Contribution of the Study

This project makes the following contributions:

- A cloud-based real-time smart security system was built at least hardware and affordable price
- End-to-end performance of smart security camera system is evaluated in terms of time required, accuracy, real-time stability and video quality.
- Use readily available machine learning model, less training overhead

1.7 Outline of the Report

This report covers a total 5 chapters of the following:

1. Chapter 1: Introduction
2. Chapter 2: Literature Review
3. Chapter 3: Methodology and Work Plan
4. Chapter 4: Results and Discussion
5. Chapter 5: Conclusion and Recommendations

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, famous examples of CNN architecture are introduced. In addition, the applications in relevant fields such like surveillance camera, object detection, machine learning and etc., are discussed.

2.2 Famous CNN architectures

In this section, the neural network architectures and their remarkable achievements on computer vision in recent years are analysed.

The paper titled “ImageNet Classification with Deep Convolutional Networks” published at Neural Information Processing Systems Conference 2012 cited for 26900 times, is considered as one of the must-read papers for beginners to deep learning. It can be said that the trend of deep learning had been brought up by this paper during year 2012. The CNN named AlexNet published in this paper had won 2012 ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) with its 14.5 % top-5 error rate. Top-5 error rate refers to the rate that first five predictions of model are all wrong. AlexNet is a CNN trained on 15 million labelled images of over 22000 categories. In this paper, architecture of AlexNet is discussed. (Alex Krizhevsky, 2012).

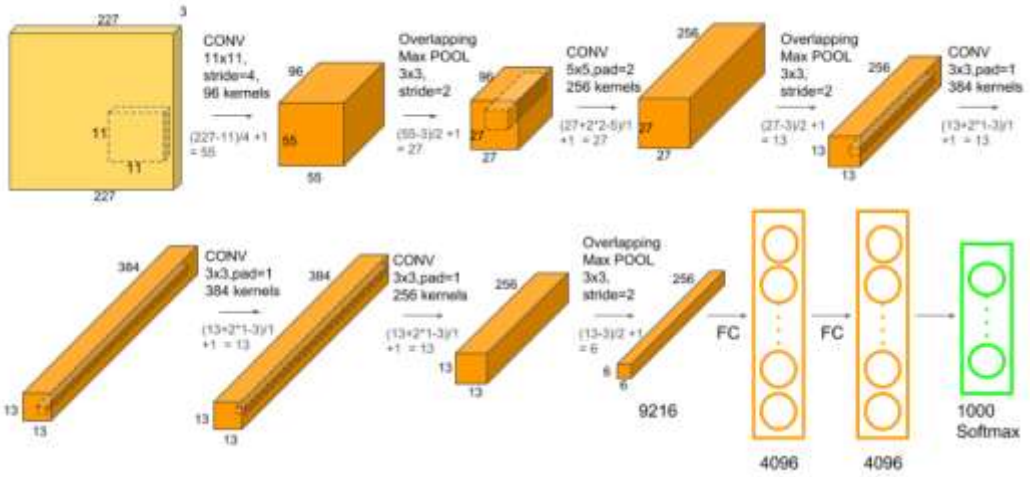

Figure 2.1 Architecture of AlexNet (Alex Krizhevsky, 2012).

There are five convolutional layers and three fully-connected layers in Alexnet. The second fully-connected layer which is also the output layer is connected to a 1000-way softmax regression because the network is purposed to classify 1000 objects. Softmax regression as known as multinomial logistic regression is a classification method that handle outputs of multiple classes. The training of network was done on two GTX 580 Graphic Processor Units (GPU) using five to six days. The GPUs do not communicate at all layers but only at certain layers for example in third convolutional layer and fully-connected layers. This is the reason why the network architecture is split into two pipelines. This helps to achieve good performance as communication overhead is kept low. The training time is successfully decreased with the use of non-saturating ReLU activation function which is several times faster than tanh function and sigmoid function. The data augmentation and dropout technique are introduced in this paper to reduce overfitting. Data augmentation is an image processing technique including image translations, horizontal reflections and patch extraction. Dropout is a technique that sets output of random neurons to zero such that complex co-adaptions of neurons are reduced. Dropout was applied at the first two fully-connected layer. Without dropout, substantial overfitting would have occurred in the network. However, drop out doubles the number of iterations required to converge. In this paper, it is shown that CNN is capable of achieving high accuracy and short training time in computer vision. (Alex Krizhevsky, 2012).

CNN had successfully grabbed everyone's attention since its success in 2012. In 2013, ILSVRC is won by convolutional neural network again. With top-5 error rate of 11.2%, ZFNet had defeated other opponents. ZFNet is a convolutional neural network built by Matthew Zeiler and Rob Fergus from New York University. The network is published through a paper titled "Visualizing and Understanding Convolutional Neural Networks" by them in 2013. The architecture of ZFNet is similar to AlexNet. It is a fine-tuned version of AlexNet. (Fergus, 2014).

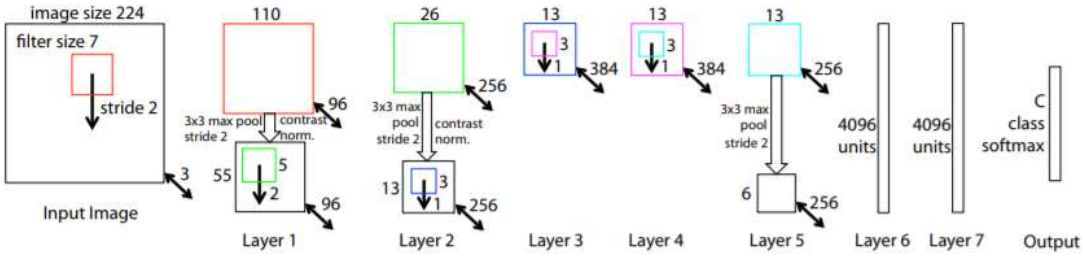

Figure 2.2 Architecture of ZFNet (Fergus, 2014).

Similar to AlexNet, it consists of five convolutional layers and three fully-connected layers. Minor modifications are made on first layer, 11x11 sized filters are used in Alexnet while 7x7 sized filters are used in ZFNet. The reason to decrease size of filter is to capture more original pixel information of input image. Deconvolutional network is proposed in this paper. Deconvolutional network does the opposite of convolutional layer; it maps features to pixel. By using this technique, the problems and inner activities such as feature evolution during training and feature invariance are illustrated and thus the network become easier for debugging and improvement. In this paper, the Alexnet is remodelled to have higher performance. Detail explanations on how a CNN works, which was a limited knowledge previously, are provided. The useful technique to visualise inner activities of CNN is revealed. This has helped in future improvements of CNN; CNNs are able to be adjusted based on inner activities to obtain better performance. In addition, ablation study to investigate performance contribution of different model layers is carried out. It is also revealed that the minimum depth of the network is vital to the performance of model. (Fergus, 2014).

There is another remarkable paper titled “Very Deep Convolutional Networks for Large-scale Image Recognition”. This paper is said remarkable because it proposed a concept that CNN have to be deep and simple. In this paper, the performance of CNN is manipulated by the depth of network while other parameters remained constant. 3x3 sized filters are used in convolutional layer while 2x2 sized filters are used in pooling layer. In such condition, two convolutional layers can produce effective receptive field equally to single 5x5 filter. An effective receptive field of a 7x7 filter can be made when connecting three convolutional layers. The benefit of using multiple layers of 3x3 filter instead of one 5x5 or 7x7 filter is the output of model is more discriminative. The size of filters could be remained with the increase of depth of network by adding more convolutional layers. (Karen Simonyan, 2015).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.3 Architecture of VGG Net (Karen Simonyan, 2015).

The proposed CNN in this paper is VGG Net. Total six configurations of VGG Net are compared. Configuration D of total 16 layers has acquired the best

performance with top-5 error rate of 6.8 % among six configurations. In this paper, the importance of depth of CNN in computer vision is presented. A simple yet effective CNN architecture is introduced; the size of filters is kept constant and the only parameter to control is number of convolutional layers. However, the drawback is longer training time required, VGGNet consists of 138 million parameters, and it took 2 to 3 weeks to train on 4 Nvidia Titan Black GPUs. (Karen Simonyan, 2015).

When the simplicity of CNN architecture is the main stream, the winner of ILSVRC 2014, GoogLeNet from Google held the different point of view. GoogLeNet impressively won the competition at the top-5 error rate of 6.67% which is very close to or even exceeds human level. To measure the performance of GoogLeNet, organisers of ILSVRC invited an artificial intelligence expert, Andrej Karpathy, to challenge the same image classification task given to GoogLeNet. Everyone might think that human accuracy would be 100 %, however, it is not due to limitations of memory, knowledge and cognitive ability. After few days of training, Andrej Karpathy achieved a top-5 error rate of 5.1 %. This had proven that the performance of GoogLeNet is close to human in object classification. GoogLeNet was published in the paper titled “Going Deeper with Convolutions”. The architecture of GoogLeNet is complex and deep. Inception module was first introduced in the paper. (Christian Szegedy, 2015).

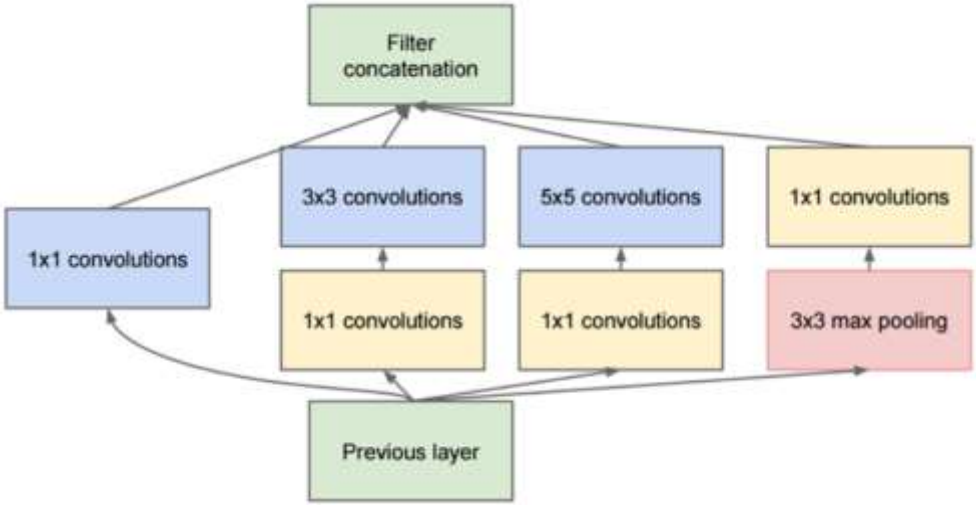

Figure 2.4 Architecture of Inception Module (Christian Szegedy, 2015).

The architecture of inception module is shown in Fig. 2.4. Inception module consists of six convolution layers of different sizes and one max-pooling layer. The reason of using different sizes of convolution layers is to extract pixel information in a more completed attitude. The max-pooling layer helps in lowering the size of samples and thus solves overfitting. Inception module is able to perform tasks of multiple layers in parallel therefore the computational time can remain efficient. GoogLeNet consists of 9 inception modules and the depth of whole network is about 100 layers. To improve computational efficiency, fully-connected layer is rarely used in GoogLeNet as fully-connected layer consumes huge number of parameters. Fully-connected layers are removed from the main trunk of network and could only be found in side branches. This paper has proposed novel architecture idea that connects the essential layers in the form of module. (Christian Szegedy, 2015).

CNN is good for image and video processing. When it comes to text and speech processing, recurrent neural network (RNN) is an ideal choice. Combining both CNN and RNN, a model that able to generate multiple natural language descriptions and object classifications is proposed. (Andrej Karpathy, 2017).

Figure 2.5 Concept Figure of Proposed Model (Andrej Karpathy, 2017).

Fig. 2.5 shows concept figure of the model, each object in the figure is classified and labelled with natural language descriptions. The natural language description is realised by setting short sentences as weak labels corresponding to unknown segments of the image. The function of CNN is to infer the alignment between sentences and the region of object. While multimodal RNN is used to produce object descriptions in text. In training phase, a set of images labelled with short descriptions is used as input. Multimodal embedding in deep learning refers to the mixture of different types of media. The results generated by the model are used for training data for the second model which learns to generate object descriptions. This paper is impressive as the application is a combine of computer vision and natural language processing. (Andrej Karpathy, 2017).

2.3 Relevant Applications

In this section, the applications related to object classification and detection are discussed. In the paper titled “Automatically Identifying, Counting, and Describing Wild Animals in Camera-Trap Images with Deep Learning”, a smart camera with animal identification, counting and behaviour describing functions is proposed. Motion-sensored cameras are widely used in wildlife conservation. Motion-sensored cameras are usually installed in the habitats of wildlife to observe their living habits and perform population count. Information in the captured images is required to be manually extracted by human which is very time-consuming. Implementing deep neural network, valuable information can be automatically extracted. In this paper, machine learning models which recognise 48 wildlife species are trained with different types of CNN. Figure 2.6 shows brief introduction of CNN used in this paper. (Mohammad Sadegh Norouzzadeh, 2018).

Architecture	No. of layers	Short description
AlexNet	8	A landmark architecture for deep learning winning ILSVRC 2012 challenge (31).
NiN	16	Network in Network (NiN) is one of the first architectures harnessing innovative 1 × 1 convolutions (49) to provide more combinational power to the features of a convolutional layers (49).
VGG	22	An architecture that is deeper (i.e., has more layers of neurons) and obtains better performance than AlexNet by using effective 3 × 3 convolutional filters (26).
GoogLeNet	32	This architecture is designed to be computationally efficient (using 12 times fewer parameters than AlexNet) while offering high accuracy (50).
ResNet	18, 34, 50, 101, 152	The winning architecture of the 2016 ImageNet competition (25). The number of layers for the ResNet architecture can be different. In this work, we try 18, 34, 50, 101, and 152 layers.

Figure 2.6 CNN used in producing species-recognising machine learning model (Mohammad Sadegh Norouzzadeh, 2018).

There are total 9 models trained. Four of the models are trained on AlexNet, NiN Net, GoogLeNet while the rest of five were trained on ResNet with different numbers of layer. The 9 trained models are gathered to form an ensemble model, by doing so, the accuracy of prediction is improved. Benchmarking on the expert-labelled test images set, the ensemble model has shown a 99.1 % of top-5 accuracy. For single model, ResNet-152 trained model has achieved the best performance by having top-5 accuracy of 98.8 %. Fig. 2.7 shows the predictions of the model. The left figure was predicted correctly while the right figure was given a wrong prediction. The animals of right figure were too far from camera and made classification difficult. (Mohammad Sadegh Norouzzadeh, 2018).

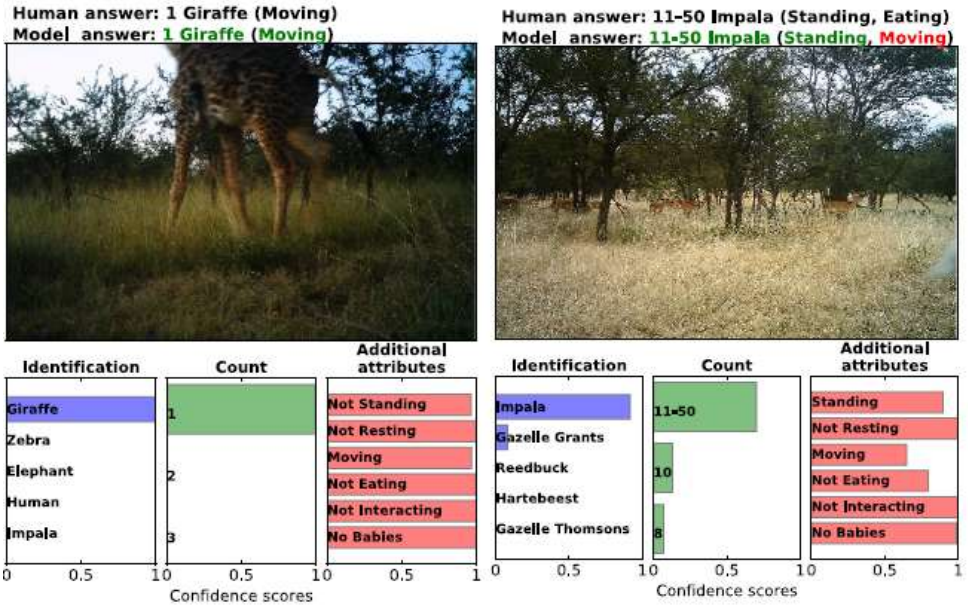

Figure 2.7 Prediction Results (Mohammad Sadegh Norouzzadeh, 2018).

The drawback of this model is that multi-species detection is not available. When there is more than one species on screen, only one species can be recognised. This paper has demonstrated how deep learning helps in wildlife conservation. Cameras with animal classification and behaviour description function has eased the process of analysing photos; the labour cost and time cost are greatly reduced. (Mohammad Sadegh Norouzzadeh, 2018).

Another application of smart camera is car park occupancy detection system. The paper title is "Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning". The car parking occupancy detection system is designed to run real-time occupancy detection using CNN and Raspberry Pi Camera Module on Raspberry Pi. Ground sensors are widely used in parking occupancy detection system, however, it requires lots of human effort to install and maintain quality of every sensor. This parking occupancy detection system has addressed the problem, it can simultaneously monitor 50 parking lots using one camera. The datasets use in model training and testing are created by themselves. The datasets contain images of car park taken from different points of view under various light conditions. The datasets are made public such that science community working on car parking occupancy detection system could use. Two CNNs are used in training the model; one is mLeNet while another is mAlexNet. The 'm' stands for mini. Fig. 2.8 shows the architecture of both CNN. (Giuseppe Amato, 2016).

net	*conv1*	*conv2*	*conv3*	*fc4*	*fc5*
mLeNet	30x11x11+4 pool 5x5+5 -	20x5x5+1 pool 2x2+2 -	-	100 ReLU	2 soft-max
mAlexNet	16x11x11+4 pool 3x3+2 LRN, ReLU	20x5x5+1 pool 3x3+2 LRN, ReLU	30x3x3+1 pool 3x3+2 ReLU	48 ReLU	2 soft-max

Figure 2.8 Architecture of mLeNet and mAlexNet (Giuseppe Amato, 2016).

SINGLE CAMERA EXPERIMENTS

train	*test*	*net*	*base lr*	*accuracy*
A (even)	A (odd)	mLeNet	0.001	0.993
		mAlexNet	0.01	0.996
A (odd)	A (even)	mLeNet	0.001	0.982
		mAlexNet	0.005	0.993
B (even)	B (odd)	mLeNet	0.001	0.861
		mAlexNet	0.01	0.911
B (odd)	B (even)	mLeNet	0.001	0.893
		mAlexNet	0.005	0.898

MULTI CAMERA EXPERIMENTS

train	*test*	*net*	*base lr*	*accuracy*
A	B	mLeNet	0.0001	0.843
		mAlexNet	0.001	0.863
B	A	mLeNet	0.001	0.842
		mAlexNet	0.0005	0.907

Figure 2.9 Performance of mLeNet and mAlexNet under different scenarios (Giuseppe Amato, 2016).

Figure 2.9 shows the performances of different CNNs under single camera and multiple cameras while A and B represent different data sets.

The performance of mAlexNet model is better than mLeNet model. Accuracy of mAlexNet model has achieved a maximum of 99.6 % in single camera scenario and 90.7 % in multi-camera scenario. (Giuseppe Amato, 2016).

	INTRA-DATASET			**INTER-DATASET**	
train	PKLot-2D	CNRP.-Odd	CNRP.-Even	PKLot-2D	CNRP.
test	PKLot-N2D	CNRP.-Even	CNRP.-Odd	CNRP.	PKLot
model					
mAlex	0.981	0.901	0.907	0.829	0.904
LPQu	0.966	0.869	0.815	0.646	0.398
LPQgd	0.970	0.877	0.813	0.617	0.410
LPQg	0.957	0.869	0.816	0.638	0.438
LBPuri	0.874	0.850	0.763	0.653	0.497
LBPu	0.951	0.868	0.800	0.643	0.467
LBPri	0.879	0.865	0.820	0.642	0.487
LBP	0.945	0.874	0.872	0.631	0.529

Figure 2.10 Performance of mAlexNet model and Radian Basis Function Kernel models (Giuseppe Amato, 2016).

The performance of mAlexNet model is compared to other type of model instead of CNN model. The compared model is trained on radian basis function kernel which is a commonly used function in support vector machine classification. From the table, it can be seen that CNN has significantly defeated radian basis function kernel in capability of detecting occupancy. This paper highlights how capable is CNN in real life application. Contrasting with other method like using ground sensors or other mathematic algorithm, CNN still performs the best. (Giuseppe Amato, 2016). Unfortunately, the statistic of time of the car park detection system is not shown in this paper.

2.4 Summary

The literature of CNN applications are usually focusing on comparing accuracy of several CNN model while timing performance is barely mentioned.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

In this chapter, the components and equipment used in this project are introduced. The methods are introduced stage by stage.

3.2 Equipment required

This is a brief introduction on equipment used in this project, both hardware and software. The hardware required are Intel Movidius Neural Compute Stick (NCS), Raspberry Pi Zero W, laptop and Raspberry Pi Camera Module. The software required are Ubuntu 16.04 Operating System, Raspbian Operating System, Software Development Kit for NCS (NCSDK), OpenCV, Caffe, Telegram and Google Drive.

Intel Movidius Neural Computer Stick is a deep learning hardware development platform consisting vision processing unit that allows the process of prototyping, tuning and deploying Convolutional neural network for real-time inferencing with low power consumption. With Intel Movidius Neural Computer Stick, the real-time inference time can be greatly scaled down to a few milliseconds. Loaded with example apps such as live-time image classifier, gender-age classifier and etc., it is suitable for beginner of deep learning application development. User can leverage the existing example applications and tune it to meet own requirements. Intel Movidius Neural Computer Stick is chosen for this project due to its low power consumption, convenient size, low price, and powerful performance on real-time inferencing and user-friendly.

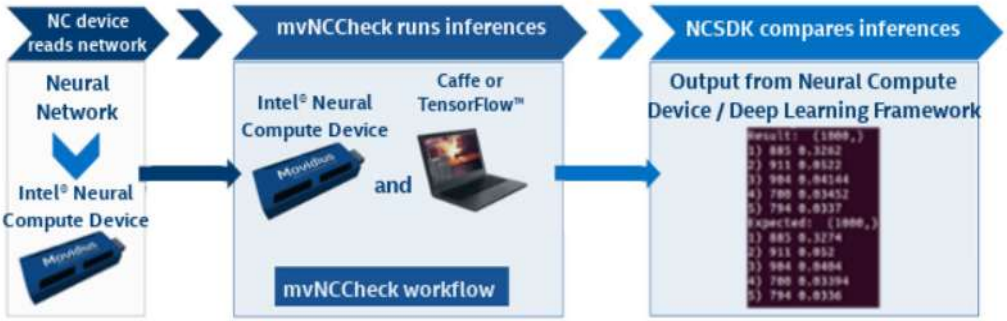

Figure 3.1 Intel Movidius Neural Compute Stick Working Concept

Raspberry Pi Zero W is a computer packed into a single board, it is very small in size compared to normal computer. Raspberry Pi Zero W supports Wi-Fi, Bluetooth and USB on-to-go. It can also work in companion with camera modules and micro SD card. It is ideal for this project due to its small size, low power consumption, low cost and complete function.

Raspberry Pi Camera Module is designed for Raspberry Pi. There are different version of Raspberry Pi Camera Module with different resolutions. The resolution of camera used in this project is 5 Megapixel (MP). It has still picture resolution of 2592 x 1944 and supports video recording at 1080p at 30fps.

Raspbian is the optimized operating system of Raspberry Pi. It is Debian-based as known as Unix-like operation system. Raspbian without graphical interface uses a command line interface named Unix-shell. Bash language is the command language used in Unix-shell. Ubuntu is a widely used free open source operating system. It is chosen because it is same Debian-based like Raspbian. High similarity of operation systems allows the project to go more conveniently.

NCSDK is a software environment for NCS to be deployed. Currently, it only supports two operation system - Ubuntu 16.04 and Raspbian Stretch. It consists of some basic machine learning applications which allows user to further deploy.

OpenCV is a machine learning library consisting functions for real-time computer vision such as frame capturing, image processing and etc.

The deep learning framework used in this project is Caffe. Caffe is originally developed by Berkeley Vision and Learning Center (BVLC) and dedicated contributors. It is written in C++ and it supports Python. Caffe is used in this project because Caffe is developed for image classification purpose; it is widely used in computer vision. In addition, Caffe consists of many pre-trained models with many types of neural networks allowing user to deploy models without writing any script. Users can just edit the existing configuration files. This feature is useful for fine-tuning existing network.

Telegram is an open source messaging and voice over IP apps which can be configured in Unix-based operation system.

Google Drive is an online data storing service for user to save files on their servers. One of the advantage of Google Drive is that users are allowed to access their files across many types of device. Customized solutions are designed to make Google Drive compatible with most of the OS and devices.

3.3 Installation of Operation System (OS), Software and Dependencies

The version of Raspbian OS installed is 2018 Raspbian Stretch Lite. This lite version does not have graphical interface, it is much smaller than normal version. The size of lite version image is 1.8 GB while the size of normal version image is 4.8 GB. After OS installation, Wi-Fi and Secure Shell Protocol (SSH) connection have to be set up. With SSH, the Raspberry Pi Zero W can be accessed by any devices wirelessly connected to the same local network. All of the following tasks are done through SSH. Next, Software Development Kit for NCS (NCSDK) is installed. NCSDK is a software environment for NCS to be deployed. The installation of NCS includes the installation of machine learning framework and library such as Caffe and OpenCV. Then, Telegram and Google Drive are installed and set up in Raspberry Pi Zero W.

3.4 Machine Learning Model Selection

The machine learning model selected for this project is MobileNet-SSD caffemodel trained by a Github user named chuanqi305. (chuanqi305, 2017) The MobileNet-SSD model is trained on MS-COCO datasets and then fine-tuned on VOC0712. It has a mean average precision (mAP) of 0.727. Caffe is a framework for convolutional neural network training. (Center, 2015) Caffemodel is machine learning model that trained with Caffe framework. MobileNet-SSD is formed by a base network - MobileNets and an object detection framework – SSD (Single Shot Multibox Detector). Base networks are usually trained for classification on huge image dataset. ILSVRC is an annual competition for base networks. The examples of base networks include AlexNet, GoogLeNet, VGGNet and etc. Object detection frameworks are algorithms to find the bounding boxes of objects in images. Object detection frameworks cannot work on their own, they must be combined with base networks. RCNN, SSD, YOLO are famous object detection frameworks. MobileNets is a class of computer vision models which are released by Google at 2017, they are designed to allow the use of convolutional neural network on embedded devices or mobile phones. Object recognition is highly dependent on computational power of the platform, platform with weak computational power would cause the object recognition to be slow or not accurate. The publishing of MobileNets have addressed the problem. MobileNets is a group of low-power, small-size, and low-latency models parameterized to suit a variety of use cases on limited resources. Like other large scale models (e.g. Inception), MobileNets can be developed for classification, detection, embeddings and segmentation. (Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, 2017). SSD is an object detection network proposed by Google at 2016. It detects object at an easier, faster and more accurate way. SSD takes one single shot to detect multiple objects within the image while another popular object detection framework, Faster-RCNN, requires two shots. Faster-RCNN is RPN (Region Proposal Network)-based network, the algorithm of RPN in object detection is to generate region proposal at first shot then detect the object in each proposal at the second shot. SSD eliminates the step of generating region proposal and subsequent step of resampling and encapsulation of pixel. SSD makes prediction at

one shot. SSD produces a collection of fixed-size bounding boxes and object prediction scores for each boxes following by non-maxima suppression to remove duplicated predictions describing the same object. Not only RPN-based network, SSD also outperforms the previous most recent single shot detection network, YOLO. The following table shows a comparison on performances of SSD, Faster R-CNN and YOLO. The mean average precision is measured on VOC2007 data set.

	SSD	Faster R-CNN	YOLO
fps	59	7	45
Mean average precision (mAP)	74.3 %	73.2 %	63.4 %

Table 3.1 Comparison on SDD, Faster-RCNN and YOLO on fps and mAP (Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, 2016)

Combining MobileNets which is suitable for embedded devices and SSD which currently is the best object detection framework, MobileNet-SSD has become a fast, powerful deep learning-based method for real-time object detection.

3.5 Integration of Camera, Machine Learning Model and NCS

The script that used to integrate camera, machine learning model and NCS is written in Python. Flowchart of the python script is shown in Fig. 3.2. It can be seen as two parts. One is main loop part, another is video recording part. The main loop part consists of two functions –image processing and inference. The camera is capturing frame continuously. The captured frame is resized and sent for mean subtraction and scaling. The new size of frame depends on the type of network. For the case of MobileNet-SSD, the required size of input image is 300 x 300. Mean subtraction and scaling is used to centre the image. After image processing, the frame is ready to be sent for inference. For the case of using NCS, the inference is done by using graph file. Graph file is a file that converted from the caffemodel and it is compatible with NCSDK. Instead of using recording function of OpenCV, the video is saved frame by frame. This is because this function is blocking function, once using it, it is unable to proceed to next inference. If person is detected, the frame is saved to buffer. If the

person left, the consecutive frames is still being saved for a period of time. When the length of consecutive frames has reached expectation, those frames are saved as video file and uploaded to Google Drive. ‘q’ is the quit button which can be pressed to stop the system.

The system can run without NCS, a little modifications on python scripts are needed. When NCS is not used, the inference is done by directly using the caffemodel and prototxt. As mentioned above, caffemodel is the machine learning model trained using Caffe framework. While prototxt is the human-readable file describing network architecture of the machine learning model. The flowchart is presented in Fig. 3.3.

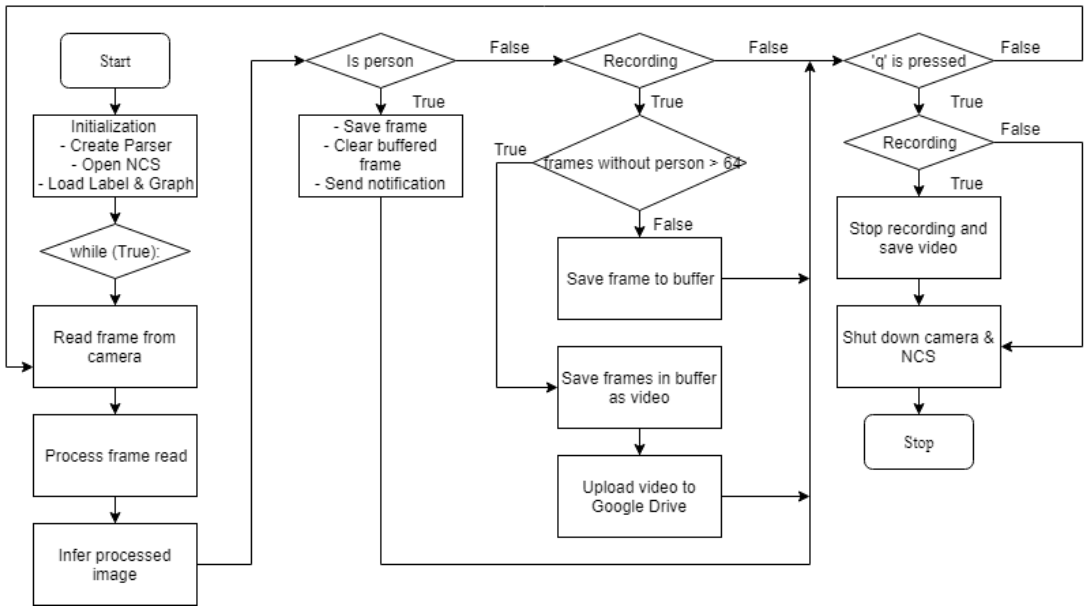

Figure 3.2 Flowchart of the system (with NCS)

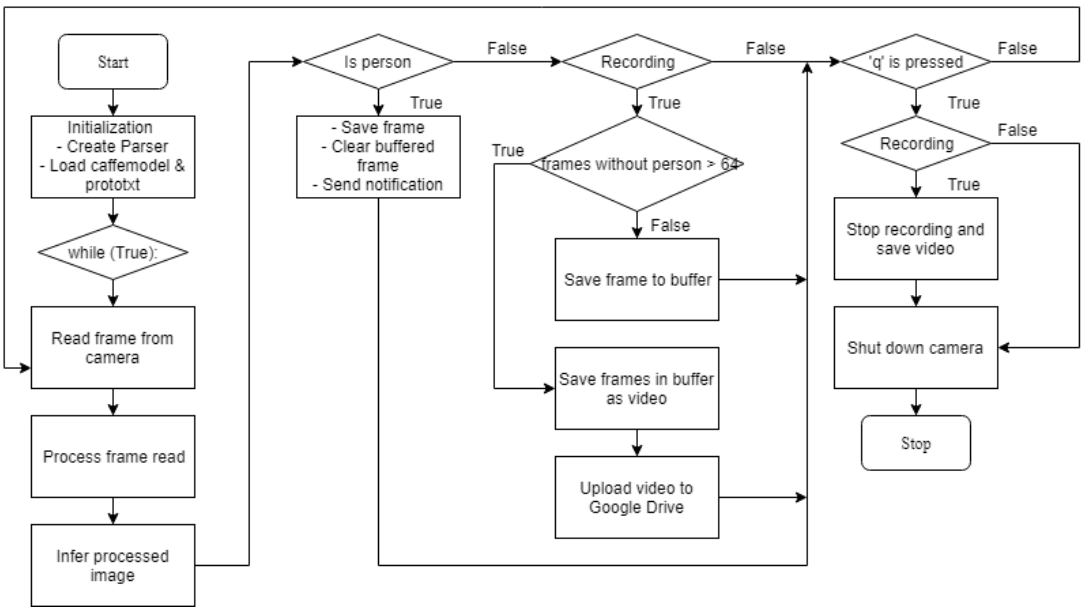

Figure 3.3 Flowchart of the system (without NCS)

3.6 Measurement of different stages of time

The average runtimes of each of the following processes are taken:

- Camera Read – to capture image
- Image Process – to resize and centre the image
- Loop – to finish processing and infer one frame
- Inference – to perform object detection

The readings are averaged from running 5000 loops at each of the following platforms:

- laptop
- laptop + NCS
- Raspberry Pi Zero W
- Raspberry Pi Zero W + NCS

3.7 Measurement of precision and recall

Precision and recall are a measure of correctness of prediction for data set with imbalanced classes. Precision refers to the correct prediction within returned results.

Recall refers to the ability of model to return results. Precision and recall are represented as:

Equation 3.1 Precision

$$Precision = \frac{TP}{TP + FP}$$

Equation 3.2 Recall

$$Recall = \frac{TP}{TP + FN}$$

where

TP = true positives,

FP = false positives,

TN = true negatives,

FN = false negatives.

To obtain precision and recall, a data set consisting at least 1000 samples of interested class and non-interested class is needed. (Md Modasshir, Alberto Quattrini Li, and Ioannis Rekleitis, 2018). Threshold value of the machine learning model is modified and thus changing the values of precision and recall. For the application with human-triggered recording, human is treated as interested class. 1000 samples consisting human and non-human are prepared. If a human was truly identified as human, it would be a true positive case. If a human was identified as non-human, it would be a false negative case. True negative case refers to case when a non-human was identified as non-human, while false positive case refers to a non-human was identified as human. By manipulating the threshold value, the values of precision and recall are obtained and plotted into a graph.

3.8 Analysis of Video Quality

Video quality is an important assessment for security camera because whether the details of detected person can be shown clearly in the footage or not is depending on it. Video quality can vary with compression method, type of camera, human and environmental factor causing camera shake and etc. Video quality is usually

evaluated by human perception, whether the video suits the human's preferences in clearness, smoothness, size and etc. In a more scientific way, video quality can be evaluated through video quality analysis tool. The video quality analysis tool utilised in this section is MSU Video Quality Measurement Tool (VQMT) free version. VQMT free version provides several metrics measurements for low resolution video. Generally, metric measurements can be divided into two types – with reference and without reference. Metric measurements with reference requires at least two inputs to proceed, they are usually purposed to compare video quality before and after transmission. For examples, Peak signal to noise ratio (PSNR) and Structural similarity index (SSIM). Metric measurements without reference provide quality assessment for single video. The metric measurements without reference include blocking, blurring, brightness flicking, frame drop and etc. 5 HEVC-compressed video clips and 5 MPEG-4 compressed video clips each containing 100 frames are evaluated under both metric measurements without reference including blocking, blurring, brightness flicking and frame drop and metric measurements with reference including PSNR and SSIM. MPEG-4 and HEVC are known as H264 and H265 respectively.

HEVC is the successor of MPEG-4. In comparison of data compression, HEVC has 25% to 50% higher data compression rate at the same level of video quality than MPEG-4. (H265, 2019). Currently, MPEG-4 is the most widely adopted video compression standard in digital video compression. Its applications include HDTV broadcasting, normal internet streaming, video compression in Blu-Ray disc and etc. HEVC is not popular yet although it had been released at 2013. (Philippo, 2018). The reason of this is that, HEVC has a few patent pools with different pricing and terms & conditions while MPEG-4 has only one patent pool. This has leaded to unclarity and confusion to consumers and resulted in no support or partly support by major browsers. (Lloyd, 2018). The emergence of new codec, AVI, which is created by Alliance for Open Media has become a strong competitor of HEVC. Alliance for Open Media is a non-profit organization co-founded by Google, Amazon, Netflix, Microsoft, Intel, Cisco and Mozilla. AVI is royalty free and supported by major browsers like Chrome, Firefox, Edge and Safari.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

In this chapter, the results are presented and discussions are made based on the results.

4.2 System Overview

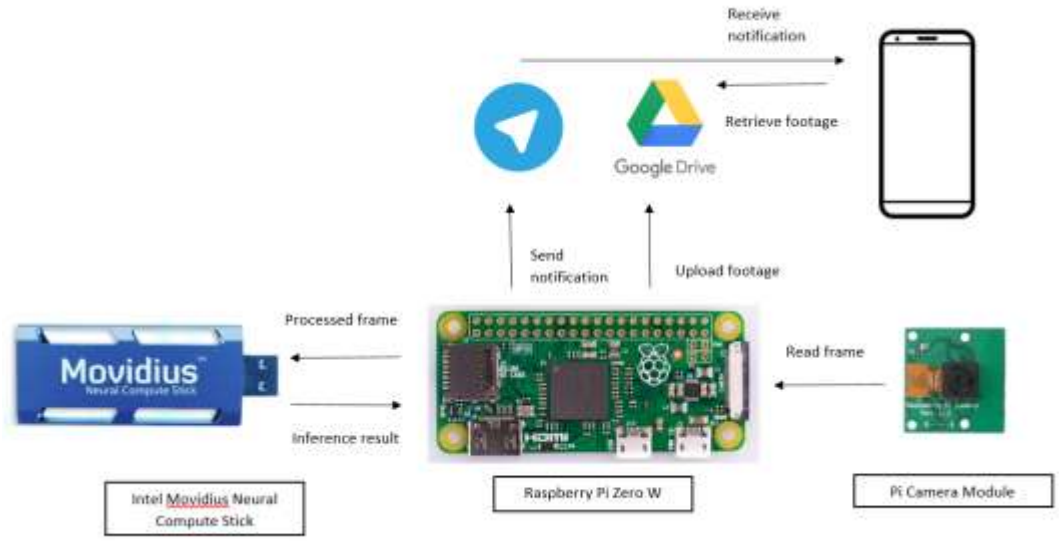

Figure 4.1 Block Diagram

Fig. 4.1 shows the block diagram of a cloud-based security system which detects human and records video only if human is detected. At the moment a person is detected, notification is sent to client through Telegram. The system keeps recording until the person left. Footage is uploaded immediately after the recording ends. The client is able to retrieve the footage at Google Drive. The equipment used and their roles and relationships are shown. The system are formed by Raspberry Pi Zero W, NCS and Raspberry Pi Camera Module. Raspberry Pi Zero W acts like the “brain” of the system which controls and processes all signal to or from its peripheral devices.

NCS consists of vision processing unit which helps in accelerating the inference of neural networks. Pi Camera Module is an input device which captures image.

Work flow of the system can be explained starting from a frame captured by the camera module. Once the frame is captured, Raspberry Pi Zero W reads the frame from the camera module. The frame is then be processed by Raspberry Pi Zero W using OpenCV. The processes include resize, mean subtraction and scaling which are purposed to shape the frame into suitable size and center the data. The processed frame is loaded to NCS for inference. The inference is done in NCS and the results are returned to Raspberry Pi Zero W. Raspberry Pi Zero W analyzes the returned results, and decide whether to take any further actions like saving frames into video, sending notifications and uploading video clips.

Figure 4.2 Prototype

The prototype is presented in Fig. 4.2. Dimension of the prototype is less than 10cm x 5cm x 5cm. This system is small in size which allows it to be conveniently placed at any flat surfaces or mounted on wall.

4.3 Timing Analysis

	Camera Read (s)	Image Process (s)	Inference Time (s)	Loop Time (s)
Raspberry Pi	1.162×10^{-4}	1.121×10^{-1}	1.732×10	1.743×10
Raspberry Pi + NCS	5.966×10^{-4}	2.450×10^{-1}	7.996×10^{-2}	2.867×10^{-1}
Laptop	5.155×10^{-6}	4.509×10^{-3}	1.197×10^{-1}	2.207×10^{-1}
Laptop + NCS	1.652×10^{-5}	2.690×10^{-2}	8.045×10^{-2}	2.022×10^{-1}

Table 4.1 Different stages of time on different platforms

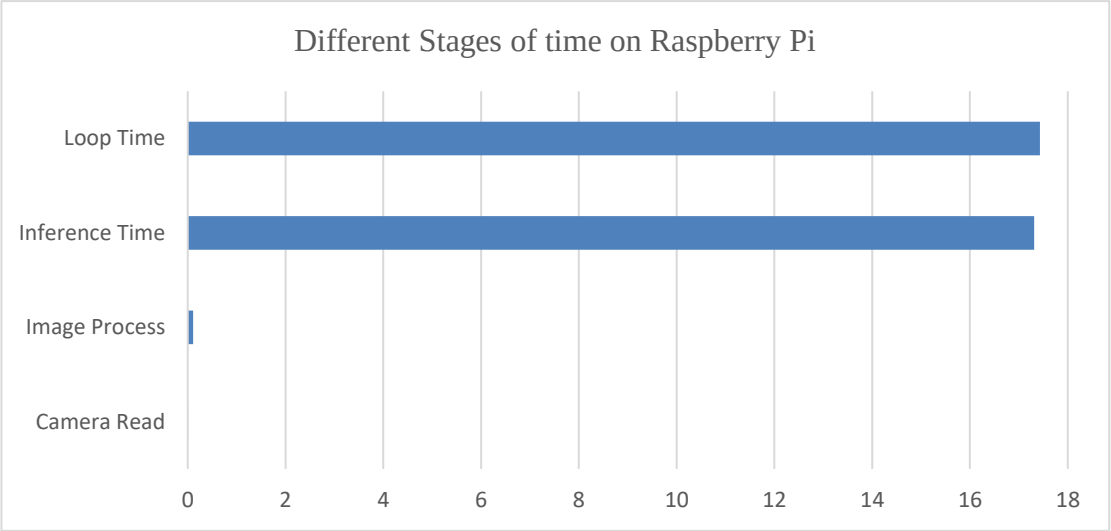

Figure 4.3 Different stages of time on Raspberry Pi

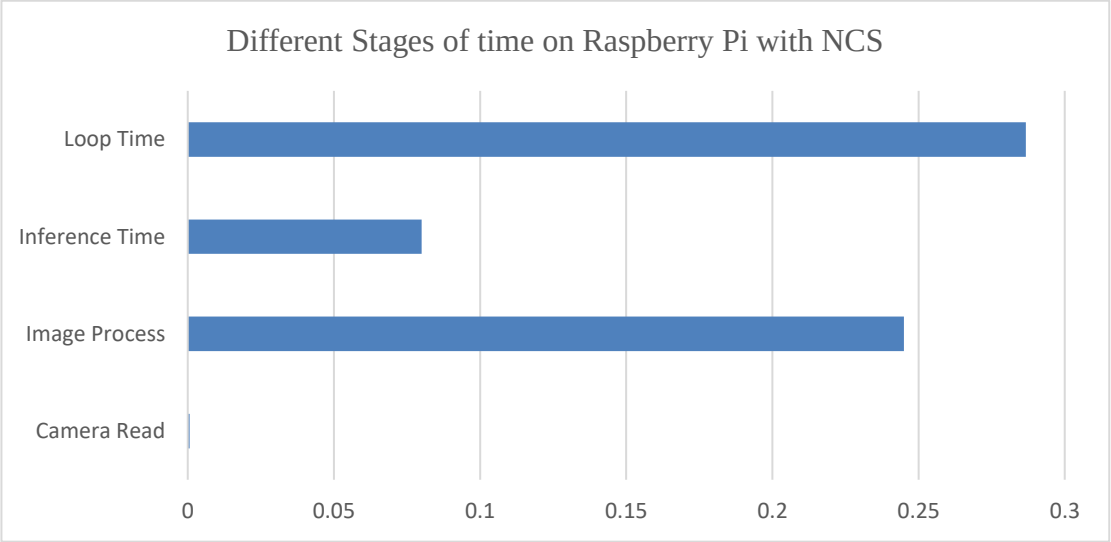

Figure 4.4 Different stages of time on Raspberry Pi with NCS

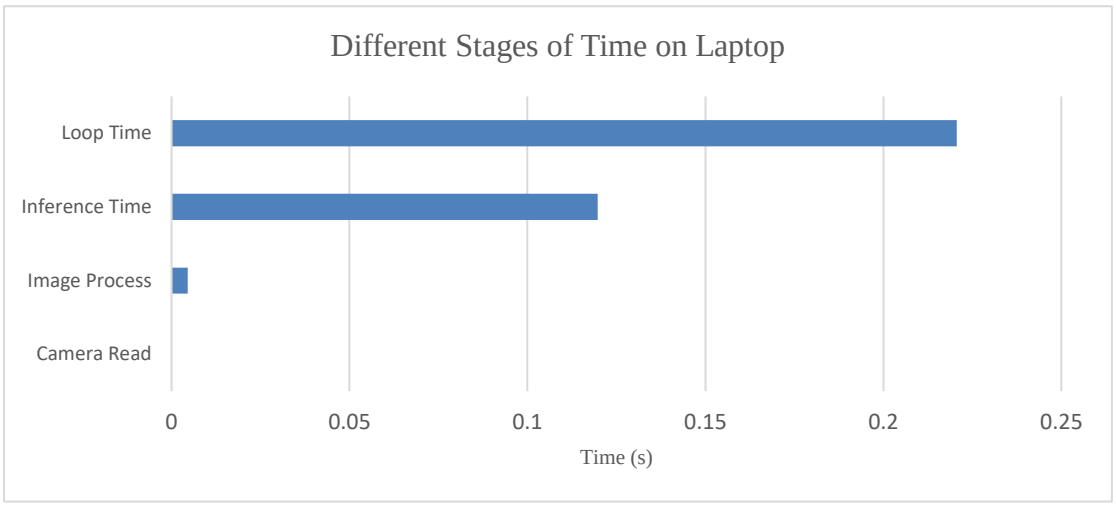

Figure 4.5 Different stages of time on laptop

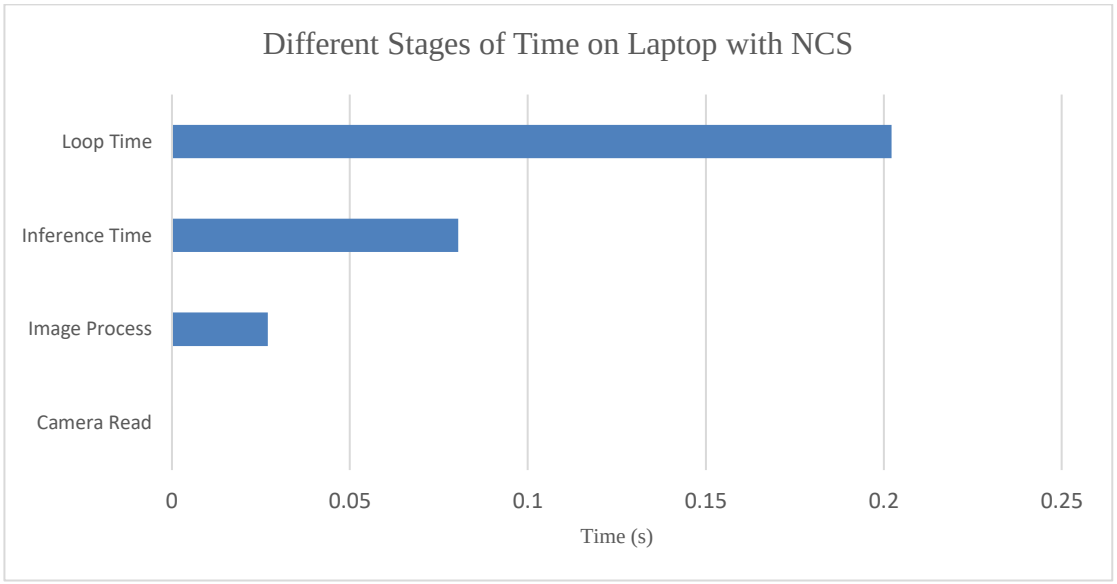

Figure 4.6 Different stages of time on laptop with NCS

In the cases without NCS, time spent on inferring a frame occupied the most of loop time. On Raspberry Pi, the inference time occupies almost 99.33 % of loop time. On Laptop, the inference time occupies almost 54.26 % of loop time. It is concluded that inference time affects loop time significantly. As the video is saved frame by frame in every loop, the loop time also indicating the frame could be saved in a period of time. Hence, inference time is the decisive factor on quality of video, the more the frames could be saved, the better the quality of video. Comparing the overall readings for cases of Raspberry Pi and Raspberry Pi with NCS, it can be seen that there is a great difference in inference time. The inference time after using NCS has

shorten 17.24 s compared to case of just using Raspberry Pi. The inference time is also found shorten 0.039 s after using NCS on laptop. Comparing the performance of Raspberry Pi and Laptop, Raspberry Pi has longer camera read time, image process time and loop time than laptop. This is because the processor of Raspberry Pi is less powerful than laptop's. With the use of NCS, Raspberry Pi can achieve almost same loop time with laptop – 0.29 s on Raspberry Pi + NCS vs 0.22 s on laptop vs 0.20 s on laptop + NCS. In short, NCS can greatly improve the performance of embedded system with weak computational power.

4.4 Precision and Recall

Threshold	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Precision	0.936345	0.936475	0.941053	0.946309	0.949339	0.955711	0.958128	0.971279	0.976744
Recall	0.912	0.908549	0.894	0.862	0.846	0.82	0.778	0.744	0.672

Table 4.2 Value of precision and recall at different threshold levels

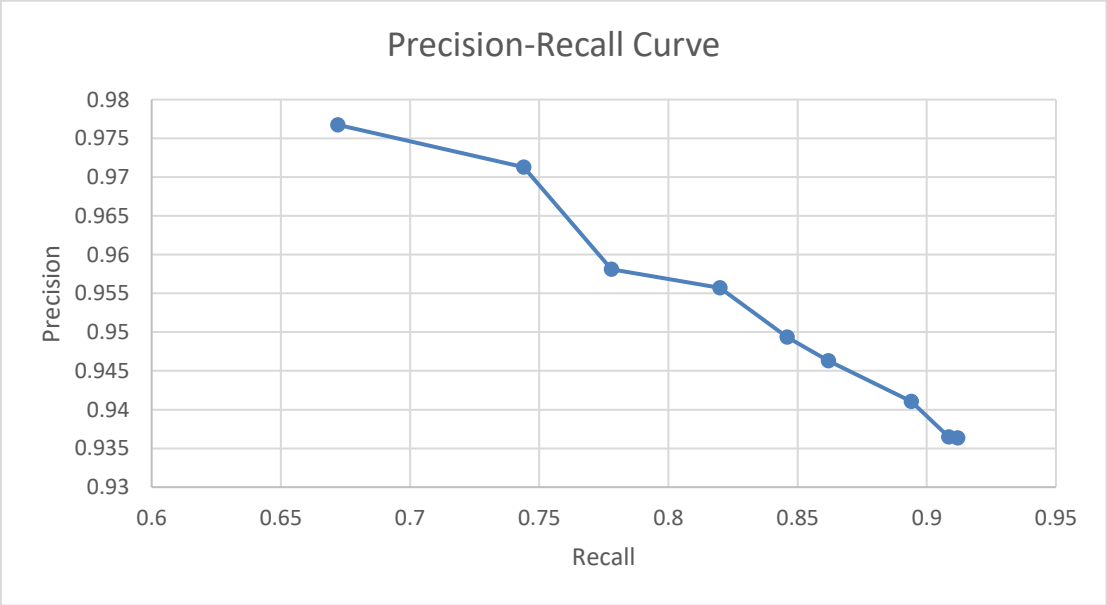

Figure 4.7 Precision-recall curve

Using 1000 samples of human and non-human, 500 for each category. The number of TP, FP, TN and FN at threshold levels of 0.1 to 0.9 are recorded. Precision and recall are calculated using Eqn. 3.1 and Eqn. 3.2. It is found that with the increment of threshold level, the precision increases while the recall decreases. A system with

higher recall and lower precision returns greater number of results with lower credibility. Oppositely, a system with higher precision and lower recall returns very few number of results with high credibility. An ideal system should have balanced value of precision and recall such that it returns optimum amount of results with high correctness. Threshold level shall be adjusted to achieve this. From Table 2, it can be concluded that 0.6 is the optimum threshold level because from 0.5 to 0.6, there is a large increment in precision (0.006) with a little sacrifice of recall (0.026).

4.5 Real-time performance

4.5.1 Stability of the system

Figure 4.8 Person detected at low light condition

Figure 4.9 Person undetected but caught by camera at low light condition

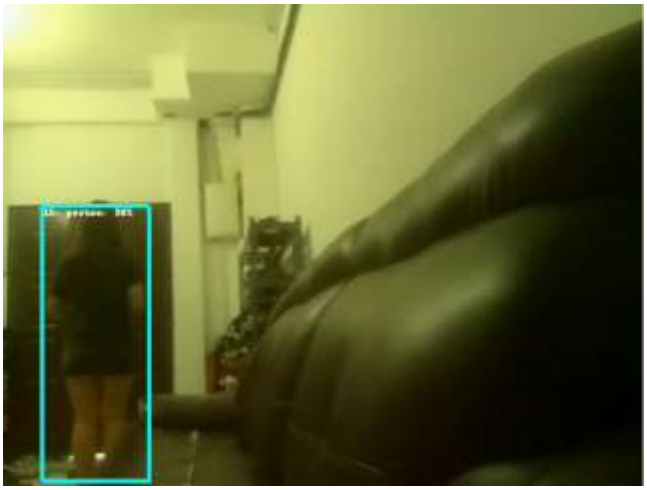

Figure 4.11 Person detected at medium bright condition

Figure 4.10 Person detected at extreme bright condition

Figure 4.12 Notifications received by owner

If a system does not have consistent performance, the system is useless. To evaluate the stability of our system, the system was placed in a room under different light conditions. The system was able to run smoothly most of the time. Humans were able to be detected and notifications were sent instantly. The system was found possible to crash when there was no enough resource on Raspberry Pi Zero W. The feature of extended recording was able to record the full scene even the person was blur and undetected during bad lightning.

4.5.2 Efficiency of the system

Measured under Internet speed of 180 Mbps, the average video upload speed is 155.55 kb/s. The average time spent to send a notification is 0.533 s. Speed of uploading video and sending notification is crucial when evaluating the performance of real-time system. In this application, the sooner the client receives notification and retrieves video, the faster the client can take action, the higher the chance to stop the unpleasant event from happening.

4.6 Video Quality Analysis

4.6.1 Video Quality Metrics without reference

In this section, video quality metrics without reference of MPEG-4-compressed video and HEVC-compressed video are compared.

	Blocking Metric	Blurring Metric	Brightness Flicking Metric	Drop Frame Metric
Video 1	18.4	0.0139	2.74	0
Video 2	18.3	0.0141	6.50	0
Video 3	17.2	0.0050	5.84	0
Video 4	17.5	0.0179	0.40	0
Video 5	16.4	0.0210	4.20	0
Average	17.6	0.0143	3.936	0

Table 4.3 Video quality metrics without reference (MPEG-4 compression)

	Blocking Metric	Blurring Metric	Brightness Flicking Metric	Drop Frame Metric
Video 6	16.4	0.0123	1.89	0
Video 7	15.2	0.0259	1.25	0
Video 8	16.6	0.0188	1.43	0
Video 9	14.0	0.0279	0.35	0
Video 10	13.8	0.0078	0.40	0
Average	15.2	0.0185	1.064	0

Table 4.4 Video quality metrics without reference (HEVC compression)

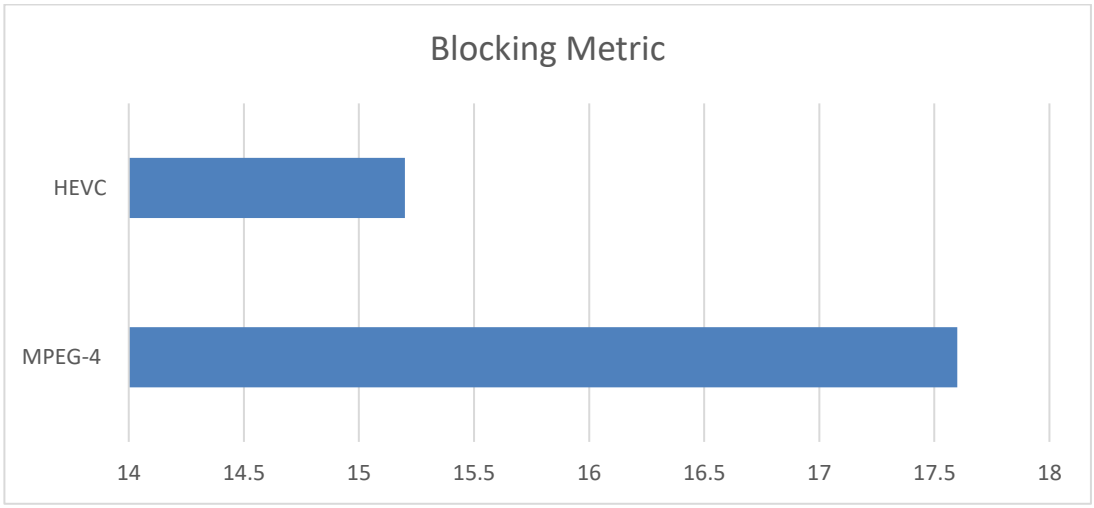

Figure 4.13 Blocking metric of MPEG-4 and HEVC compressions

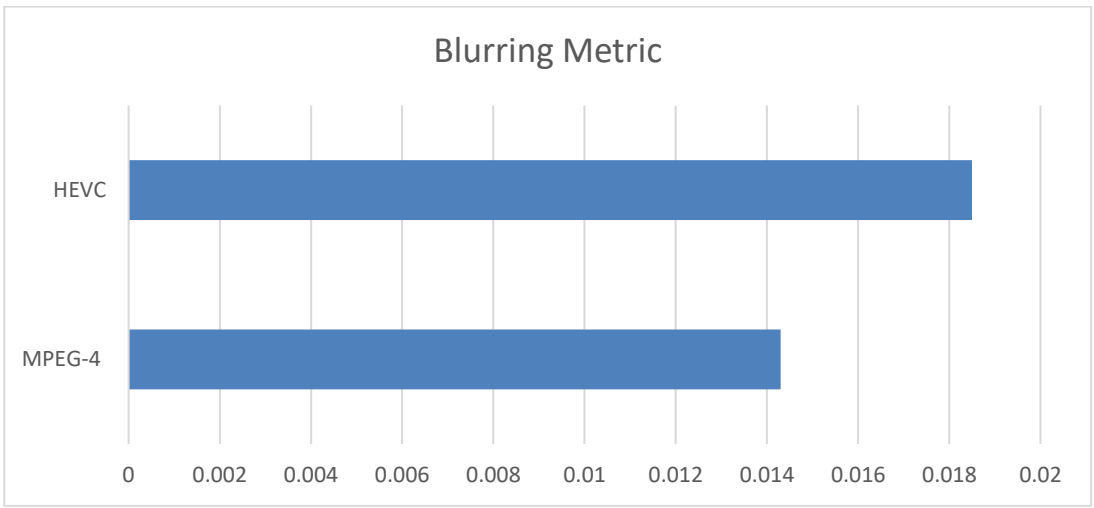

Figure 4.14 Blurring metric of MPEG-4 and HEVC compressions

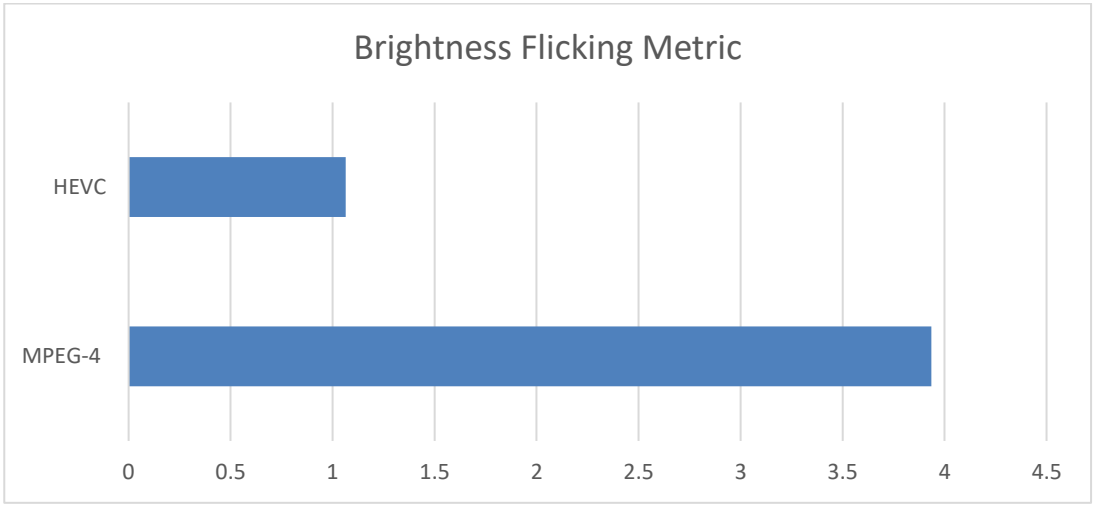

Figure 4.15 Brightness flicking metric of MPEG-4 and HEVC compressions

Figure 4.16 Source

Figure 4.17 Blocking metric visualisation

Blocking metric is a measure of blocking effect in a video. The blocking is more noticeable in smooth areas of the video comparing to contrast areas. Information in previous frame is required to achieve better accuracy in detecting blocking. Lower value of blocking metric indicates better quality of video. As shown in Fig. 4.13, HEVC-compressed video has lower blocking metric than MPEG-4-compressed video. Hence, HEVC has better video quality in term of blocking.

Figure 4.18 Blurring metric visualisation

Blurring metric is measured by comparing two consequent frames. The more blurred frame gets lower score than the less blurred frame. Therefore, it is preferred to have higher value of blurring metric. In Fig. 4.14, it is shown that HEVC-compressed video has higher value of blurring metric compared to MPEG-4-compressed video. It is found that HEVC compression provides slightly clearer video than MPEG-4 compression.

Figure 4.19 Brightness flicking metric visualisation

Brightness flicking metric is purposed to measure flicking rate of two consequent frames. Modulus of difference of average brightness of the frames is calculated. The larger the modulus the higher the brightness flicking between two frames. Brightness flicking can be seen as update of brightness. From Fig. 4.15, it can be concluded that the update of brightness in HEVC-compressed video is less than MPEG-4-compressed video.

Figure 4.20 Drop frame metric visualisation

Drop Frame Metric measures drop-frame in a video. There are two possible output of drop frame metric – zero or one. Zero indicates existing frame while 1 indicates drop-frame. Both compression methods show a perfect result with zero frame-drop.

4.6.2 Video Quality Metrics with reference

Reference 1	Reference 2	PSNR	SSIM
Video 1	Video 6	48.8	0.992
Video 2	Video 7	50.9	0.998
Video 3	Video 8	48.8	0.998
Video 4	Video 9	49.0	0.996
Video 5	Video 10	46.7	0.991
Average		48.84	0.995

Table 4.5 Video quality metrics with reference (MPEG-4 vs HEVC)

In this section, HEVC-compressed video is used as first source while MPEG-4-compressed video is used as second source for video quality metrics with reference. PSNR, SSIM and VQM of these two types of video will be measured.

PSNR stands for “Peak Signal to Noise Ratio”, which is a ratio of the maximum signal over the mean-squared error between two references where these two references are usually formed by one original sample and one distorted or reconstructed sample. This ratio is usually used as the quality assessment between two images or videos. PSNR can be mathematically expressed as Eqn. 4.1. It can be seen that as the value of mean-squared error approaches zero, the value of PSNR approaches infinity. The larger the value of PSNR, the lower the mean-squared error, the higher the similarity between original and reconstructed sample indicating the better visual quality of distorted or reconstructed sample.

Equation 4.1 Mean-squared error (MSE)

$$MSE = \frac{\sum_{m=1}^{M}\sum_{n=1}^{N}[\, I_1(m,n) - I_2(m,n)]^2}{M \times N}$$

Equation 4.2 PSNR

$$PSNR = 10\log_{10}\frac{R^2}{MSE}$$

where

I_1 = first sample,

I_2 = second sample,

M = number of rows of input images

N = number of columns of input images

R = maximum fluctuation in input image data type

From Table. 4.5, the average PSNR of MPEG-4-compressed video and HEVC-compressed video is found to be 48.84. PSNR value does not have absolute meaning. It is used for comparison purpose.

Structural Similarity Index as known as SSIM, is another standard in measuring image or video quality. Both PSNR and SSIM compare the similarity between two samples, however, they have different sensitivity varying with the format of samples. (Alain Horé, Djemel Ziou, 2010) Unlike PSNR which emphasizes on error of samples, SSIM tends to evaluate the quality of samples like human visual system (HVS) where distortion of luminance, distortion of contrast and loss of correlation will be considered. (Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, Eero P. Simoncelli, 2004) SSIM is illustrated through Eqn. 4.3.

Equation 4.3 SSIM

$$SSIM(x, y) = l(x, y)c(x, y)s(x, y)$$

Equation 4.4 Luminance comparison function

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{{\mu_x}^2 + {\mu_x}^2 + C_1}$$

Equation 4.5 Contrast comparison function

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{{\sigma_x}^2 + {\sigma_x}^2 + C_2}$$

Equation 4.6 Structure comparison function

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}$$

where

μ_x = mean intensity of sample ‘x’

μ_y = mean intensity of sample ‘y’

σ_x = standard deviation of intensity of sample ‘x’

σ_y = standard deviation of intensity of sample ‘y’

C_1, C_2, C_3 = constant to avoid instability when denominator is close to zero

The maximum value of SSIM is 1, and the minimum value of SSIM is -1, representing best quality and poorest quality respectively. As in Table. 4.5, the SSIM is 0.995, since video with MPEG compression is used as the second source, it can be explained that the quality of video with MPEG compression did not significantly drop in comparison with HVEC-compressed video.

4.6.3 Feasibility in terms of size and compression time

Comparing both standards, it is found that MPEG-4 is more feasible. The main reason that HEVC is less feasible is long encoding time. OpenCV does not directly support HEVC codec. (Getting Started with Videos, 2017) Therefore, it requires extra lines of code to convert the compression of video from MPEG-4 to HEVC format. Time taken for the conversion is extremely long because the conversion speed is low. It is found that the average conversion speed is 0.25 fps. This problem does not exist when using MPEG-4, because it is supported by OpenCV, the encoding is done along with OpenCV function - VideoWriter() without extra time.

The advantage of HEVC over MPEG-4 is higher compression efficiency. The size of HEVC-compressed video is much less than MPEG-4-compressed video. For video length of 30 secs, the average size of HEVC-compressed video is 67 kb while for MPEG-4-compressed video is 211 kb. 68 % of video size is reduced when HEVC compression standard is used.

4.7 Summary

In this chapter, the performance of the system is discussed and analysed in aspects of time, accuracy, real-time performance and video quality. Inference time is proven to be greatly reduced with the use of NCS. The optimum threshold level is 0.6 where the values of precision and recall at that level are 0.9557 and 0.82 respectively. The system is found to be stable enough for real-time application. When it comes to video quality, the video quality of MPEG-4-compressed video is slightly lower than HEVC-compressed video. However, MPEG-4 compression is still preferred because it is much more efficient in time.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusions

In conclusion, a smart security camera with human-triggered recording was built. The use of NCS and MobileNet-SSD machine learning model had made object detection on low-powered embedded platform possible. The time required for the system running on Raspberry Pi Zero W with NCS to capture, process and infer a frame was 0.29s. In other words, the timing performance of the system is 3.5 frames per second. The highest achieved precision is 0.9767 at threshold level of 0.9 while for recall is 0.912 at 0.1 threshold level. The final threshold level is set to be 0.6 to ensure high amount of results and high accuracy, balanced values of precision and recall are required. At threshold level of 0.6, the precision and recall are 0.9557 and 0.82 respectively. For video compression method, it is found that MPEG-4 compression is more suitable for this application as it is more efficient in time without significantly reducing video quality. Finally, this system is put to the practical test and found stable at most of the time.

5.2 Recommendations for future work

System fps is still very low compared to normal video recording. Future work shall be working on improvement of fps which is related to timing parameters studied in this paper i.e. camera read time, image process time, inference time and etc. In addition, the problem of long time spent on compressing video with HVEC codec shall be overcome such that compression codec of newer generation can be used without increasing burden on the system. This system can be further modified to be used for different purpose for example ATM monitoring, wild-life counting, traffic light signal control and etc.

REFERENCES

Alain Horé, Djemel Ziou. (2010). Image Quality Metrics: PSNR vs. SSIM. *2010 International Conference on Pattern Recognition*, 2366-2369.

Alex Krizhevsky, I. S. (2012). ImageNet Classification with Deep Convolutional Networks. Lake Tahoe, Nevada, USA: Neural Information Processing Systems Conference .

Andrej Karpathy, L. F.-F. (2017). Deep Visual-Semantic Alignments for Generating Image Descriptions. *IEEE Transactions on Pattern Analysis and Machine Intelligence , 39*(4), 664-676.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Card, C. (2018). *Facebook Newsroom*. Retrieved February 25, 2019, from https://newsroom.fb.com/news/2018/09/inside-feed-suicide-prevention-and-ai/

Center, B. V. (2015, May 27). *Caffe: a fast open framework for deep learning*. Retrieved from Github: https://github.com/BVLC/caffe

Chien-Chang Chen, Hung-Hui Juan, Meng-Yuan Tsai & Henry Horng-Shing Lu. (2018). Unsupervised Learning and Pattern Recognition of Biological Data Structure with Density Functional Theory and Machine Learning. *Nature*.

Christian Szegedy, W. L. (2015). Going Deeper with Convolutions. Boston, Massachusetts: IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

chuanqi305. (2017, June 8). *MobileNet-SSD*. Retrieved from Github: https://github.com/chuanqi305/MobileNet-SSD

Dhanashree Vijayrao Madhekar, Prof. Mrinal Rahul Bachute. (2017). Real Time Object Detection and Tracking using Raspberry Pi. *International Journal of Engineering Science and Computing*, 13233-13236.

Dudley, R. (2018). *The Star Online*. Retrieved February 25, 2019, from https://www.thestar.com.my/opinion/letters/2018/06/15/suicide-epidemic-is-a-concern/

Fergus, M. Z. (2014). Visualizing and Understanding Convolutional Neural Networks. Zurich, Switzerland: European Conference on Computer Vision.

Gary J. Sullivan, Jens-Rainer Ohm, Woo-Jin Han, Thomas Wiegand. (2012). Overview of the High Efficiency Video Coding. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 1649-1667.

Getting Started with Videos. (2017, December 7). Retrieved from OpenCV: https://docs.opencv.org/3.4.0/dd/d43/tutorial_py_video_display.html

Giuseppe Amato, F. C. (2016). Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning. *IEEE Symposium on Computers and Communication (ISCC), 10.1109/ISCC.2016.7543901* .

Google. (2018). *Google Trends*. Retrieved June 24, 2018, from https://trends.google.com/trends/explore?date=all&geo=US&q=machine%20learning

H265. (2019, January 7). Retrieved from FFMPEG: https://trac.ffmpeg.org/wiki/Encode/H.265

joshinishant2305. (2018, December 20). *Transfer Learning in Object Detection and API Used in Object Detection*. Retrieved from Medium: https://medium.com/@joshinishant2305/transfer-learning-in-object-detection-and-api-used-in-object-detection-4673244cf

Kalva, H. (2006). The H.264 Video Coding Standard. *IEEE Multimedia*, 86-90.

Karen Simonyan, A. Z. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. International Conference on Learning Representations.

Lloyd, R. (2018, April 9). *HEVC Patent Pool* . Retrieved from IAM Media: https://www.iam-media.com/patent-pools/despite-patent-pools-comeback-video-compression-market-shows-licensees-still-have

M, D. (2018, November 19). *Data Science Using Unsupervised Learning & Visualization of Astronomy Data*. Retrieved from Towards Data Science: https://towardsdatascience.com/data-science-using-unsupervised-learning-visualization-of-astronomy-data-b6b1c61f6922

Mitchell, T. M. (1997). *Machine Learning.* New York: McGraw-Hill Science/Engineering/Math.

Mohammad Sadegh Norouzzadeh, A. N. (2018). Automatically Identifying, Counting, and Describing Wild Animals in Camera-Trap Images with Deep

Learning. *Proceedings of the National Academy of Sciences of the United States of America, 115*(25), 5716-5725.

Philippo, E. J. (2018, January 24). *5 reasons why h.265 is not the future of video compression technology*. Retrieved from Eagle Eye Networks: https://www.een.com/h-265/

Rodigues, A. (2016, June 9). *A technical comparison of H264 vs H265* . Retrieved from Medium: https://medium.com/advanced-computer-vision/h-264-vs-h-265-a-technical-comparison-when-will-h-265-dominate-the-market-26659303171a

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development* , 210-229.

Suicide data. (2019, March 20). Retrieved from World Health Organization: https://www.who.int/mental_health/prevention/suicide/suicideprevent/en/

Sunway Campus Library Adopts Facial Recognition Technology. (2018, February 1). Retrieved from Sunway College : https://college.sunway.edu.my/news/2019/sunway-campus-library-adopts-facial-recognition-technology

Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, Ajay Luthra. (2003). Overview of the H.264/AVC Video Coding Standard. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 560-576.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. (2016). SSD: Single Shot MultiBox Detector. *arXiv:1512.02325*.

Xiao, F. (2000). Dct-based Video Quality Evaluation. *Final Project for EE392J*, 769.

Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh, Eero P. Simoncelli. (2004). Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 600-612.

APPENDICES

APPENDIX A: Scripts

Security Camera with NCS (Laptop and Raspberry Pi share the same script)

```
#!/usr/bin/python3

# ****************************************************************************
# Copyright(c) 2017 Intel Corporation.
# License: MIT See LICENSE file in root directory.
# ****************************************************************************

# Detect objects on a LIVE camera feed using
# Intel® Movidius™ Neural Compute Stick (NCS)

import argparse
import cv2
import numpy
import ntpath
import os
import sys
import select
import time
import telepot
import upload
import mvnc.mvncapi as mvnc
from time import localtime, strftime
from utils import visualize_output
from utils import deserialize_output
from imutils.video import VideoStream
from kcw import KeyClipWriter

# ---- Pre-start settings ------------------------------------------------

# "Class of interest" - Display detections only if they match this class ID
CLASS_PERSON        = 15

# Detection threshold: Minimum confidance to tag as valid detection
CONFIDANCE_THRESHOLD = 0.60 # 60% confidant

# Variable to store commandline arguments
ARGS                = None

# OpenCV object for video capture
camera              = None

#Initialize global variable
consecFrames=0

#Initialize telegram pass token
bot = telepot.Bot('625446283:AAGva0AVLPMYjCbOHbVlrImmVyjEdLihceQ')


# ---- Step 1: Open the enumerated device and get a handle to it -------------

def open_ncs_device():

    # Look for enumerated NCS device(s); quit program if none found.
    devices = mvnc.EnumerateDevices()
    if len( devices ) == 0:
        print( "No devices found" )
        quit()

    # Get a handle to the first enumerated device and open it
    device = mvnc.Device( devices[0] )
    device.OpenDevice()
```

```
    return device

# ---- Step 2: Load a graph file onto the NCS device -------------------------

def load_graph( device ):

    # Read the graph file into a buffer
    with open( ARGS.graph, mode='rb' ) as f:
        blob = f.read()

    # Load the graph buffer into the NCS
    graph = device.AllocateGraph( blob )

    return graph

# ---- Step 3: Pre-process the images ----------------------------------------

def pre_process_image( frame ):

    # Resize image [Image size is defined by choosen network, during training]
    img = cv2.resize( frame, tuple( ARGS.dim ) )

    # Convert RGB to BGR [OpenCV reads image in BGR, some networks may need RGB]
    if( ARGS.colormode == "rgb" ):
        img = img[:, :, ::-1]

    # Mean subtraction & scaling [A common technique used to center the data]
    img = img.astype( numpy.float16 )
    img = ( img - numpy.float16( ARGS.mean ) ) * ARGS.scale

    return img

# ---- Step 4: Read & print inference results from the NCS -------------------

def infer_image( graph, img, frame ):

    # Use global variable in local
    global consecFrames
    global updateConsecFrames
    global cur_time

    # Load the image as a half-precision floating point array
    graph.LoadTensor( img, 'user object' )

    # Get the results from NCS
    output, userobj = graph.GetResult()

    # Get execution time
    inference_time = graph.GetGraphOption( mvnc.GraphOption.TIME_TAKEN )

    #print(str(numpy.sum(inference_time)))

    # Deserialize the output into a python dictionary
    output_dict = deserialize_output.ssd(
                output,
                CONFIDANCE_THRESHOLD,
                frame.shape )
    # When no object is detected
    if(output_dict['num_detections'] == 0):
        updateConsecFrames=True
        if kcw.recording:
            kcw.update(frame)
        if kcw.recording and consecFrames == ARGS.buffer_size:
            kcw.finish()
            upload.upload(cur_time)
            os.remove(cur_time+'.avi')

    # Print the results (each image/frame may have multiple objects)
    for i in range( 0, output_dict['num_detections'] ):

        # Filter a specific class/category
        # When object detected is person
        if( output_dict.get( 'detection_classes_' + str(i) ) == CLASS_PERSON ):
```

```
            # Extract top-left & bottom-right coordinates of detected objects
            (y1, x1) = output_dict.get('detection_boxes_' + str(i))[0]
            (y2, x2) = output_dict.get('detection_boxes_' + str(i))[1]

            # Prep string to overlay on the image
            display_str = (
              labels[output_dict.get('detection_classes_' + str(i))]
              + ": "
              + str( output_dict.get('detection_scores_' + str(i) ) )
              + "%" )

            # Overlay bounding boxes, detection class and scores
            frame = visualize_output.draw_bounding_box(
                    y1, x1, y2, x2,
                    frame,
                    thickness=4,
                    color=(255, 255, 0),
                    display_str=display_str )

            consecFrames = 0
            updateConsecFrames = False

            if not kcw.recording:
              cur_time = strftime( "%Y_%m_%d_%H_%M_%S", localtime() )
              print("person detected at " + str(cur_time))
              bot.sendMessage(749556817,"Person Detected at " + str(cur_time))
              cur_time = strftime( "%Y_%m_%d_%H_%M_%S", localtime() )
              kcw.start( cur_time + '.avi',fourcc,ARGS.fps)

          # When object other than person is detected
          else:

            updateConsecFrames = True

            if kcw.recording and consecFrames == ARGS.buffer_size:
              kcw.finish()
              print("video saved")
              upload.upload(cur_time)
              os.remove(cur_time+'.avi')

          if kcw.recording:
            kcw.update(frame)

        if updateConsecFrames:
          consecFrames = consecFrames + 1

        # If a display is available, show the image on which inference was performed
        if 'DISPLAY' in os.environ:
          cv2.imshow( 'NCS live inference', frame )


    # ---- Step 5: Unload the graph and close the device -------------------------

    def close_ncs_device( device, graph ):
        graph.DeallocateGraph()
        device.CloseDevice()
        camera.stop()
        if kcw.recording:
          kcw.finish()
          upload.upload(cur_time)
          os.remove(cur_time+'.avi')
        cv2.destroyAllWindows()

    # ---- Main function (entry point for this script ) --------------------------

    def main():

        device = open_ncs_device()
        graph = load_graph( device )

        # Main loop: Capture live stream & send frames to NCS
        while( True ):
          frame = camera.read()
```

```
        img = pre_process_image( frame )
        infer_image( graph, img, frame )
        # Display the frame for 5ms, and close the window so that the next
        # frame can be displayed. Close the window if 'Enter' is pressed.
        i,o,e = select.select([sys.stdin],[],[],0.1)
        if( i ):
          break

    close_ncs_device( device, graph )

# ---- Define 'main' function as the entry point for this script ----------

if __name__ == '__main__':

    parser = argparse.ArgumentParser(
                    description="DIY smart security camera PoC using \
                    Intel® Movidius™ Neural Compute Stick." )

    parser.add_argument( '-g', '--graph', type=str,
                    default='../../caffe/SSD_MobileNet/graph',
                    help="Absolute path to the neural network graph file." )

    parser.add_argument( '-l', '--labels', type=str,
                    default='../../caffe/SSD_MobileNet/labels.txt',
                    help="Absolute path to labels file." )

    parser.add_argument( '-M', '--mean', type=float,
                    nargs='+',
                    default=[127.5, 127.5, 127.5],
                    help="',' delimited floating point values for image mean." )

    parser.add_argument( '-S', '--scale', type=float,
                    default=0.00789,
                    help="Absolute path to labels file." )

    parser.add_argument( '-D', '--dim', type=int,
                    nargs='+',
                    default=[300, 300],
                    help="Image dimensions. ex. -D 224 224" )

    parser.add_argument( '-C', '--colormode', type=str,
                    default="bgr",
                    help="RGB vs BGR color sequence. This is network dependent." )

    parser.add_argument("-f", "--fps", type=int, default=3,
                            help="FPS of output video")

    parser.add_argument("-b", "--buffer-size", type=int, default=64,
                            help="buffer size of video clip writer")

    ARGS = parser.parse_args()

    #Load the labels file
    labels =[ line.rstrip('\n') for line in
            open( ARGS.labels ) if line != 'classes\n']

    #Initialize Video Writer
    kcw=KeyClipWriter(bufSize=ARGS.buffer_size)
    fourcc = cv2.VideoWriter_fourcc(*'XVID')

# ---- Camera Initialization ----------------------------------------------------

    print("WARMING UP CAMERA")
    camera = VideoStream(src=0).start()
    time.sleep(2.0)
# ---- Start --------------------------------------------------

    main()

# ==== End of file ===========================================================
```

Security Camera without NCS (Laptop and Raspberry Pi share the same script)

```
#!/usr/bin/python3
# Detect objects on a LIVE camera feed
import argparse
import imutils
import cv2
import csv
import os
import select
import sys
import time
import numpy as np
from kcw import KeyClipWriter
from imutils.video import VideoStream
from time import localtime, strftime


def pre_process_image( frame ):

   # Mean subtraction and scaling

   blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
                                0.007843, (300, 300), 127.5)
   return( blob )

def infer_image(frame,blob):

   global consecFrames
   global updateConsecFrames
   global cur_time
   (h, w) = frame.shape[:2]
   # Set the new input value for the network
   net.setInput(blob)
   # send the input value for inference
   out = net.forward()
    # loop over the detections
      # When no object is detected
   if(out.shape[2] == 0):
      updateConsecFrames=True
      if kcw.recording:
         kcw.update(frame)
      if kcw.recording and consecFrames == ARGS.buffer_size:
         kcw.finish()

   for i in np.arange(0, out.shape[2]):

      idx = int(out[0, 0, i, 1])
      if (idx == 15):

      # extract the confidence (i.e., probability) associated with
      # the prediction

         confidence = out[0, 0, i, 2]

      # filter out weak detections by ensuring the `confidence` is
      # greater than the minimum confidence

         if confidence > ARGS.confidence:

            # extract the index of the class label from the
            # `detections`, then compute the (x, y)-coordinates of
            # the bounding box for the object

            box = out[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")

            # draw the prediction on the frame

            label = "{}: {:.2f}%".format(CLASSES[idx],confidence * 100)
            cv2.rectangle(frame, (startX, startY), (endX, endY),COLORS[idx], 2)
```

```
            y = startY - 15 if startY - 15 > 15 else startY + 15
            cv2.putText(frame, label, (startX, y),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)

            consecFrames = 0
            updateConsecFrames = False
            if not kcw.recording:
               cur_time = strftime( "%Y_%m_%d_%H_%M_%S", localtime() )
               kcw.start(cur_time + '.avi',fourcc,20)
      else:
         updateConsecFrames = True
         if kcw.recording and consecFrames == ARGS.buffer_size:
            kcw.finish()

      if kcw.recording:
         kcw.update(frame)

   if updateConsecFrames:
      consecFrames = consecFrames + 1

   # If a display is available, show the image on which inference was performed
   ##  cv2.imshow( 'Live Inference', frame )
def shut_down ():

   camera.stop()
   if kcw.recording:
      kcw.finish()
   cv2.destroyAllWindows()

def main ():

   while (True):

      frame = camera.read()
      blob = pre_process_image(frame)
                        infer_image(frame,blob)
      i,o,e = select.select([sys.stdin],[],[],0.1)
      if( i ):
         break
      shut_down()

# ---- Define 'main' function as the entry point for this script ----------

if __name__ == '__main__':

   parser = argparse.ArgumentParser()

   parser.add_argument("-p", "--prototxt",
               default='MobileNetSSD_deploy.prototxt',
                     help="path to Caffe 'deploy' prototxt file")

   parser.add_argument("-m", "--model",
               default='MobileNetSSD_deploy.caffemodel',
                     help="/")

   parser.add_argument("-c", "--confidence", type=float, default=0.4,
                     help="minimum probability to filqter weak detections")

   parser.add_argument("-f", "--fps", type=int, default=20,
                        help="FPS of output video")

   parser.add_argument("-b", "--buffer-size", type=int, default=32,
                        help="buffer size of video clip writer")

   ARGS = parser.parse_args()

   CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
               "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
               "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
               "sofa", "train", "tvmonitor"]

   COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
```

```
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
# ---- Camera Initialization -------------------------------------------------

    print("WARMING UP CAMERA")
    camera = VideoStream(src=0).start()
    time.sleep(2.0)
    consecFrames=0
    kcw=KeyClipWriter(bufSize=ARGS.buffer_size)
    net = cv2.dnn.readNetFromCaffe(ARGS.prototxt, ARGS.model)
# ---- Start -----------------------------------------------------------------

    main()

# ==== End of file ============================================================
```