Application of Object Detection for Creating Intelligent Home Surveillance System

Final Year Project

Tarun Bisht, M.Sc. Computer Science

Department of Computer Science, DSB Campus

Kumaun University Nainital

Uttarakhand – 263002

Abstract

Home security is one of the major demand of the modern home, most cheaper home security systems are not intelligent to understand and process visual information in an image, detect different threats and alert if they found one. This project implements a cheaper solution for home security system that uses power of deep learning and object detection inside a raspberry pi that process and understand visual information captured using pi camera and alert user as it detects any threat. This project currently detects four categories (person, monkey, cat and dog) but can be extended to detect more categories. This project can also be extended from home security system to other surveillance task like detecting hunters in forest, detecting poisonous snakes, detecting threats in country border etc. by re-training object detection model with data of desired categories. A more unified and precise surveillance system can be created by using more data and compute.

Keywords: surveillance system, object detection, computer vision application

1. INTRODUCTION

The use of home surveillance system is necessary for modern houses, CCTV cameras are placed on every street, shops and societies but they are not intelligent that they can detect and alert us from threats as they appear, which sometime can save us from big trouble. This project, tries to implement an intelligent security system by re-using hardware of installed surveillance system like IP camera which will save cost or using a raspberry pi and a camera. This system is capable of detecting and alerting threats if it found one, it even record environment video at that instance which can be used for further analysis. In India there are a lots of places where monkey theft occurs a lot they silently enter the house and steal away things, sometimes they even steal or destroy some precious items, not only monkeys are thieves, cats are also expert in this task. This project extends this idea to a person and dog as well. Further, we can also extend it not only for home surveillance but to some specific task like detecting snakes that accidentally enters the house or cattle sheds which can harm the residents, protect wildlife from hunters by alerting the authorities as it detects them, alerting people about the presence of leopard or tiger in the locality there are a lot of cases that happens in our area a lot where people get attacked by them, and much more endless possibilities. To make system understand, detect and act on environment object detection is used which is part of computer vision. It let computers to classify and localize objects in an image. Using it we gave computers the power to locate objects in an environment. Object detection can be done based on classical methods of hand extracted features from images and classify them or using deep learning techniques which automatically learns features from images. These deep learning based methods outperforms classical algorithms by large margin. Currently object detection is mostly done using deep learning techniques and is one of the most active area of research. The working demo of the project detects threats using digital image

provided by a camera and alerts user through the mail with image of detected threat. As object get detected the video recorder gets activated that records short video of that instance for later watch and analysis. The demo can detect four classes monkeys, cats, dogs and person. For optimization, the project uses multiple threads that enable the program to run different threads parallelly and take advantage of full compute resources. Email and video recorder runs on their separate thread so they do not interfere with the main thread which does detections, this results the boost in inference time of the model. The object detection models were implemented using TensorFlow's object detection API which contains lots of pre-trained models. For comparing and selecting a model based on use case, three best models: SSDMobileNetV2, SSDMobileNetV2 FPNLite and EfficientNet D0 were selected based on their inference time. The inference time of these models is important since we want realtime object detection. These models were trained for 10000 steps in data collected from internet and open image dataset (OID) and compared. From comparison of these models we understand the relation between model's inference time and its accuracy are inversely proportional, more the model is accurate more it takes to infer.

In the next section we have reviewed all points which are necessary to understand the base of project from object detection, techniques and metrics to measure performance of models. Section 3 devotes to all methodology and setups along with all comparisons between models. Section 4 analyses these comparisons and deduce result and make an analysis of the further research directions.

2. REVIEW OF LITERATURE

For understanding content of image in security camera we have to use computer vision techniques that let machines to interpret contents of digital images feed through camera. This project uses computer vision techniques to detect threats in images.

2.1. Computer Vision

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos we try to create machines with vision capabilities that can detect, localize, classify object in the environment and can react based on the information they gather through their vision.

Based on the task performed with computer vision, computer vision can be classified into three levels:

Low-Level vision.

Vision tasks that include basic image manipulation came under this category. Example: resizing the image, converting to grayscale, changing exposure, finding edges, finding image gradients etc. These tasks more emphasize on pixel manipulation in images.

Mid-Level Vision.

Vision tasks that include connecting different images that could be used to map the environment came under this category. Example: panorama stitching, Multi-view stereo, Rangefinding, optical flow etc. These tasks emphasize on connecting images to form real-world scenes.

High-Level vision.

Vision task that includes extracting and features from images and understanding semantics. Example: image classification, object detection, semantic segmentation, image to 3D map etc. These tasks emphasize on extracting and understanding the content of images.
This project needed to detect threats like monkeys, person, dogs, snakes etc. from images and alert user, this task came under the category of high-level vision in machine has to understand semantic of image i.e. understand contents of image. Object detection is one of the high level vision technique that is base of this project.

2.2. Object Detection

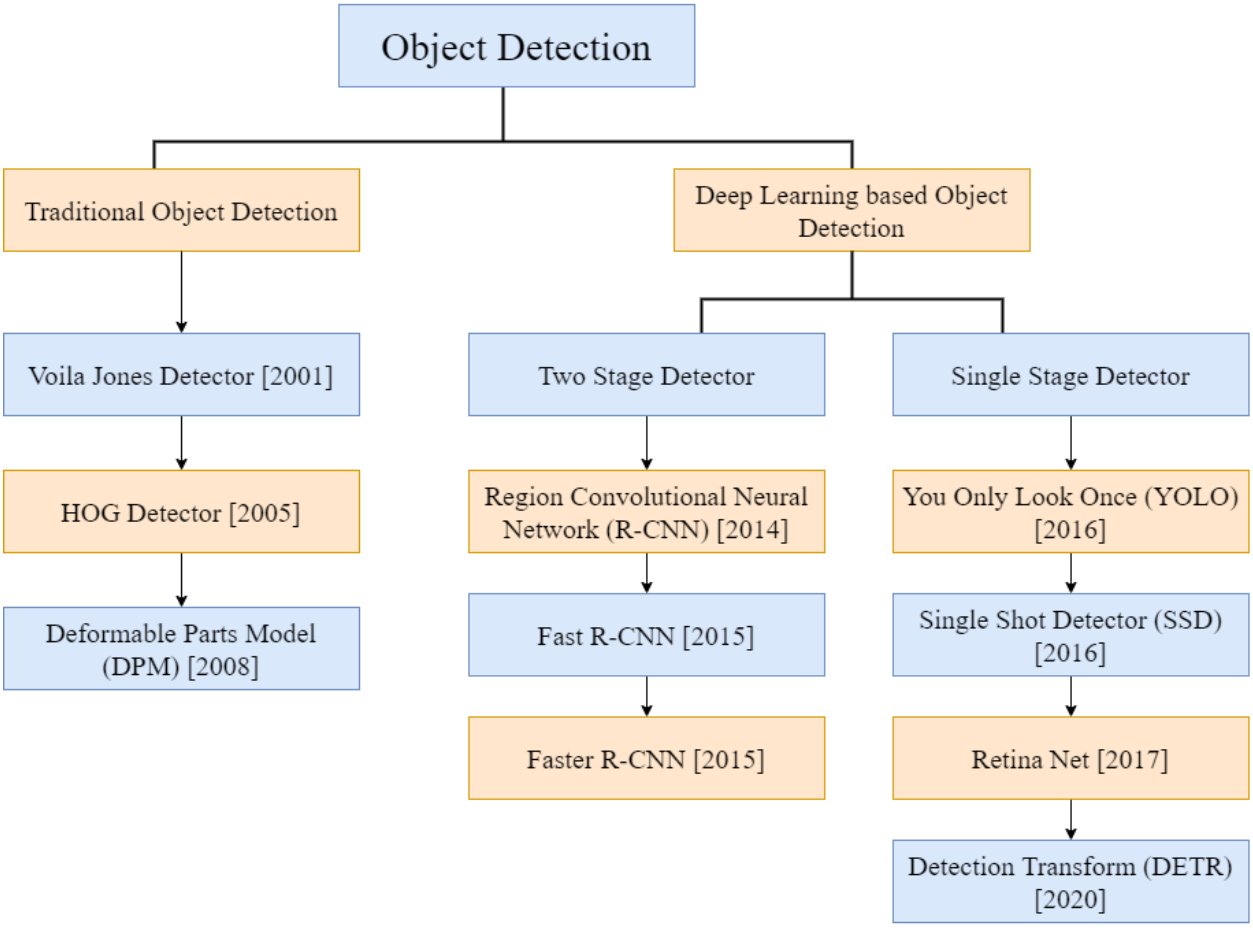
Object Detection is high level computer vision task that deals with detecting instances of visual objects in digital images. It classifies and localize objects presented in images. It helps computer to answer the question of What object is where?
Object detection is basis of many task in computer vision like object tracking, instance segmentation etc. From past two decades the object detection research is flourishing and improvement process still continues. The major impact on researches were seen after re-emergence of deep learning techniques in computer vision from then this field is progressing exponentially.
Object detection techniques are now widely used in many real-world applications like video surveillance, self-driving cars, robot vision, augmented reality etc.

Evolution of Object Detection.

The past two decades were boon for field of object detection thanks to all dedicated computer vision scientist. So without understanding the evolution of object detection it would be

difficult to understand how this field reached the current milestone. Evolution of object detection happens in phases, we can divide these phases as

- Traditional object detection
- Deep Learning based object detection



Traditional Object detection.

Deep learning phase has seen exponential growth in past decade and so object detection too, but traditional methods are one that led us to reach the position in which we are now. Most of the early object detection algorithms were based on handcrafted features due to lack of effective image representation at that time and speed hacks due to lack of computational resources.

Voila Jones Detectors.

In 2001 Paul Viola and Michael Jones achieved real-time detection of human faces in digital images for the first time which ran on a 700MHz Pentium III CPU. They used combination of simple features and boosted classifiers which run on whole image using sliding window approach. For achieving real-time detection speed it uses idea of integral image that saves lots of computation while window slide across image. The features were very simple called Haar wavelets, which were created randomly by taking average of pixels in one box and subtract with average of other box in example faces. Using this a huge set of random features pool (around 180k-dimensions) were created. Then using Adaboost algorithm a small set of features was selected which are helpful for face detection. These feature sets were easy to compute, for more optimization in calculations integral image technique was used. Integral image is computation speed up method which makes computational complexity of each window in voila jones detector independent of its window size.

HOG Detector.

HOG stands for histogram of oriented gradients proposed in 2005 by N. Dalal and B. Triggs. It is considered as improvement of SIFT (Scale invariant feature transform) which is a feature detector and descriptor. The part which is different from SIFT is normalization of gradients using local window using this balances feature invariance. the HOG descriptor also uses overlapping local contrast normalization for improving accuracy. These HOG features then trained using SVM to detect different classes of objects. The primarily motivation of HOG detection is pedestrian detection but can be used to detect different objects. To detect objects of different sizes it rescales input image for multiple times while keeping size of window unchanged.

DPM.

Deformable parts model (DPM) was originally proposed by P. Felzenszwalb in 2008 as an extension of the HOG detector. It combines knowledge of physics and mathematics therefore often called physics based model. Deformable objects are geometric objects whose shape can change overtime. DPM can deal with change in dynamics of objects an example of which is in pedestrian detection a pedestrian can have different poses. Instead of running HOG in whole image, we have HOG features of different parts of body. The result from all individual parts are merged to get output. DPM are more robust to noise than HOG.

Many of new object detection techniques are still deeply influenced by its valuable insights from P. Felzenszwalb, e.g., mixture, hard negative mining, bounding box regression, etc. In 2010 P. Felzenszwalb and R. Girshick were awarded the lifetime achievement by PASCAL VOC.

Deep Learning based object detection.

These handcrafted features models performance become saturated and very little improvements were happening in this field between 2010 – 2012. These little improvements were based on previous models which shows very insignificant progress. In 2012 after rebirth of convolutional network a new blood pumped in the field of object detection, which also led to the birth of modern object detection. Convolutional deep learning based object detection can further be divided into two groups

- Two stage detector
- Single stage detector

Two stage detector detects objects in two phases, in first phase they detect regions where objects are present and in second stage they detect bounding boxes and classes of objects in

image. While in one stage detectors combine both these phase into one, they detect and classify objects in one go.

Two stage detectors.

- *R-CNN*

RCNN uses selective search algorithm for generating region proposals by merging similar pixels into regions. The regions got from this step were warped, resized and pre-processed then passed into a CNN which produces feature vectors these feature vectors are then used for classification and bounding box regression which result bounding boxes of objects and their classes. RCNN yields a significant performance boost on VOC07 dataset, with a large improvement of mean Average Precision (mAP) from 33.7% in DPM-v5 to 58.5%.This algorithm was fast with respect to sliding window approach and then passing each window to CNN but it was still quite slow to be used in realtime object detection.

- *Fast RCNN*

In 2015, R. Girshick proposed Fast RCNN detector which is further improvement of RCNN. Fast RCNN also uses selective search to find region proposals but it redefined architecture of RCNN to make it fast. In RCNN first regions were proposed then these regions were passed to a CNN which lead to multiple pass of images to CNN which is costly operation. Fast RCNN improves this process of multiple passes to CNN which makes it much faster than RCNN. In Fast RCNN whole image is passed through CNN and regions proposals are extracted from image. These proposals then then pooled directly on feature map by using ROI (region of interest) pooling. These pooled vectors are then passed through fully connected layer for classification and bounding box regression. On VOC07 dataset, Fast RCNN increased the mAP from 58.5% in RCNN to 70.0% while having detection speed over 200 times faster than R-CNN.

This algorithm saves multiple passes to CNN but still there is bottleneck of finding region proposals which limit speed of algorithm. This was solved in later version of RCNN called Faster RCNN

- *Faster RCNN*

In 2015 shortly after Fast RCNN, S. Ren et al. proposed Faster RCNN detector. Faster RCNN was the first end-to-end, and the first near-realtime deep learning detector. Faster RCNN unified region proposal and detection network by introducing region proposal networks that enables cost free region proposals. Region proposal use anchor boxes in image and predicts if feature map contains foreground or background. This RPN network is part of training network whereas in previous RCNN networks they use selective search techniques for this task which is not part of network this decrease training and inference time of network. On VOC07 dataset, Faster RCNN increased the mAP from 70.0% in RCNN to 73.2% and COCO mAP@.5 is 42.7%.

One stage detectors.

- *YOLO*

YOLO stands for '*you only look once'* it is the first one stage object detector proposed by R. Joseph et al. in 2015. YOLO is popular object detection network because of its speed. A fast and enhanced version of YOLO can run at 155fps with VOC07 map = 63.4%. These networks are not the accurate one they suffer from drop of the localization accuracy compared with two-stage detectors, especially for some small objects but due to their speed they are used in many real world applications. R. Joseph has made series of improvements in YOLO and released YOLOv2 and YOLOv3. For detecting objects of different sizes YOLOv3 introduced detection in multiple scales of image. Currently YOLO is in active development phase the latest version YOLOv4 and YOLOv5 have been released in 2020 with major improvements on speed and

accuracy. This network divides image into certain regions and predicts bounding boxes and classification score for each region by applying a single convolutional network in image at once. YOLO networks are fully convolutional networks the final predictions of network are also in form of a feature map rather than a dense layer. It's like implementing sliding window using convolutional network.

- *SSD*

SSD was proposed by W. Liu et al. in 2015, it stands for single shot detection and second one stage network using deep learning. SSD used multi-reference and multi-resolution which improve detection accuracy for small objects. SSD have good combination of detection speed and accuracy with VOC07 mAP = 76.8% and COCO mAP@.5=46.5% and can run at 59fps. SSD detects objects of different scales on different layers of network while previous networks only run detection on top layers.

2.3. Object Detection Metrics

These metrics let us to measure performance and effectiveness of different models, and helps to compare them. Previously there was no standardize method to measure any objet detection models, different models use different metrics to measure their performance like miss rate vs false positive rate per window (FPPW) or false positive per image (FPPI) used in Caltech pedestrian detection. In recent years the most frequently used algorithm for object detection is Average Precision (AP) which was introduced in VOC2007. It is defined as average detection precision under different recalls and measured for every specific category. To compare performance over all categories, mean AP (mAP) average of AP for all categories is taken. This mAP score is taken as final measure to compare different object detection networks.

To measure accuracy of object localization the Intersection over Union (IoU) is used. It measures how much predicted bounding box overlaps with ground truth box. It is ratio of area of overlap of predicted box with ground truth box to the total area (union) of predicted box and ground box.

$$IOU = \frac{P \cap G}{P \cup G}$$

$$where\ P = predicted\ bounding\ box$$

$$and\ G = ground\ truth\ bounding\ box$$

Based on some threshold value (generally 0.5) we can categorize box as false positive or true positive.

if IOU < Threshold:
    false+ve
else:
    true+ve

The 0.5-IoU based mAP has then become the widely used metric for object detection problems for years. To calculate mAP we need to calculate two things precision and recall for every class and plot them in a graph with varying thresholds. The area under precision-recall curve gives AP for that class. To find mean average precision (mAP) we take average of AP across all classes.

2.4. Transfer Learning

Transfer learning is technique of training a network by using and fine-tuning a pre-trained model. This method of training requires less data and less computation power since most of the features were previously learned by network. This helps to train network if data and computation are limit with good accuracy. It is called transfer learning since previous knowledge of network about a task was transferred as knowledge for new task. This is same as human learns things, we

use some previous experience and learnings into new tasks and fine-tune them which helps us to learn faster.

In analogy of deep learning model we transfer weights of previous training as starting point for new training and fine-tuning our new model.

2.5. TensorFlow Object Detection API

The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. Google use this codebase to be for their computer vision needs. It let us to create accurate machine learning models capable of localizing and identifying multiple objects in a single image.

This API supports both TensorFlow 1 and TensorFlow 2. Majority of modules are compatible with both versions.

TensorFlow object detection API provide us pre-trained object detection models that can be used to train custom models with transfer learning. It also provides cutting edge new research object detection models.

This project also uses this API because it provides wide ranges of object detection models as per need and pre-trained weights for transfer learning which require less computation time and data for training as compared to training model from scratch.

3. REPORT ON THE PRESENT INVESTIGATION

3.1. Methodology

The project development was divided into phases that help in structuring the project and increases productivity.

Planning and structuring the idea

The first step for any project is crucial, here I planned all strategies and approaches to deal with the idea of the project. Materials related to such were searched and collected from the internet with information regarding best tools, practices, research papers and lectures necessary to learn before creating project. Learning tasks were prepared to efficiently gain the required knowledge for the project.

Acquiring Knowledge

All collected resources were skimmed through in the first step and useful resources are selected from all which can be referred later to acquire knowledge of my problem domain. Knowledge is acquired on computer vision from the classical era to the modern era with more emphasis on object detection. Resources include lecture videos and texts.

Setting up project environment

Concurrently with phase two, the project environment was set up to start the development of the project and implementing the idea. All tools which are needed were setup and configured on the machine. I used my system for prototyping which was ideal for prototyping and testing the idea later I use free cloud services with GPU provided by Google and Kaggle for the high intensive task and at last, the project has to be ported to raspberry pi, all platforms were set up for further development.

Planning development phase

Plan for the development of the project was created in this phase which provides a more efficient workflow and management of the project became easier. For this Google Tasks application was used. The project was divided into stages and the first five stages were the heart of the project which includes detection of threats from images and last five tasks were functionalities required for a security camera.

Project development phase

According to the plan defined in previous phase work on the project was started. The first step was to prototype the first five stages of the plan which include detection of threats. Data was collected for the training of model and a working prototype was created in local machine to train and test model. After a successful attempt, the code was shifted to Google Colab and Kaggle kernels to train.

Data Collection.

Data collection is a crucial part of training a good model. The data which I needed was location annotated images of threats which I want to detect using a camera. For this task images were scraped from the internet. These images then were hand-annotated by using an image labelling software called labelimg.

There were around 1000 images to annotate containing images of threats. Since later more threats categories were inserted it became difficult to hand annotate these images and also we need plenty more images per category to train so for that I used open image dataset which contains hand-annotated images of various categories. To download data of selected classes, a script was used that downloaded around 2000 images per category for training and around 400 images for testing from open images dataset.

Training and testing models.

After data was collected the next step was to load the data in the project and use that loaded data for training and testing the model. The models were trained on free cloud GPU provided by Google and Kaggle in Google Colab and Kaggle Kernels respectively.

Different deep learning models were trained in this phase and their progress was tracked. The models which were trained were selected on the basis their mAP score and time taken to inference an image. Then using transfer learning on those models they were trained. Transfer learning was used because of my limit of data and computation power.

Comparing and selecting model.

Since I will be using a raspberry pi later for production which is low computational device hence the model selection is very crucial for a better experience. Out of all trained models, a less computation model with mAP score greater than 50 has to be selected so to give better accuracy since all models scores above 50 the model with less inference time was selected which can be used with raspberry pi for realtime detection.

Combining detection in live feed.

Scripts for detection of threats on a live video feed from the camera were created. This script creates bounding boxes around threat when detected in feed along with confidence score and class of threat. This script was tested on the local machine with i3 6th gen CPU, the model provides performance around 10 FPS.

After successful implementation of the first stage of the project, security features were implemented the major ones were alerting person about the detection of threat and recording that instance in form of video for later watch.

Implementing Features.

For sending alerts I used emails as it is easy to set up and send notifications and we do not need to pay for any subscription. The program also restricts to send email continuously as it can spam inbox and block account too. In an email, an image of detected threat is also sent so that the user can confirm and check if it's dangerous or not. A user can also block detections emails for some specific classes of threats.

The recording feature starts recording video when any threat appears on video frame to the time it disappears from the frame. It also handles recording when multiple threats appear in the same frame.

Packing and Testing the code.

In this step, all project files are combined into a package with configuration files for script changing parameters easily. Then the whole project again undergoes testing once again which ensures all merged modules are working as expected.

Optimization with threading

The whole project is optimized by using the concept of multithreading. Modules which are independent to each other are divided into different threads so they can run concurrently this take advantage of multiple cores of CPU and performance increases by a large factor. Detection of threats, sending email alerts and recording videos are on separate threads which increases performance.

Porting to Devices

Finally, the whole project is ported on devices (surveillance system, raspberry pi) and inference performance are tested on it. For capturing live feed pi camera was used in raspberry pi

and IPCamera in surveillance system. Some optimizations were again made while reading input from the camera by introducing separate thread for this task. Final testing was again done and check for all modules working well combined with multithreading.

3.2. Experimental Setup

1. Development Tools

Python

Python is an interpreted, high-level and general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python interpreters are available for many operating systems. A global community of programmers develops and maintains CPython, a free and open-source. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and CPython development.

Python is most widely used language when it comes to machine learning due to wide numbers of scientific computation and machine learning frameworks. It is widely used language in machine learning research.

NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. In 2005, Travis Oliphant created NumPy. It is open-source software and has many contributors.

Numpy provide optimized high dimensions matrices and operations it is supported by most of machine learning frameworks.

TensorFlow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production and backbone for lot of machine learning applications

Dataflow programming is a programming paradigm that models a program as a directed graph of the data flowing between operations, thus implementing dataflow principles and architecture.

Differentiable programming is a programming paradigm in which a numeric computer program can be differentiated throughout via automatic differentiation. This allows for gradient based optimization of parameters in the program, often via gradient descent. Differentiable programming has found use in a wide variety of areas, particularly scientific computing and artificial intelligence.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015.

OpenCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a template interface that works seamlessly with STL containers.

2. Machines and Platforms

For this project different sets of machines and platforms are used to prototype, train and deploying. For prototyping and testing my personal laptop was used, since for training a model we need a good GPU without it would be very time consuming. So for training free GPU from Google and Kaggle are very helpful which helps to train model under 4 hours. For deployment raspberry pi was chosen since its size is small and is ideal for creating a security camera. The operating system of these devices vary between Windows and Linux. My personal laptop uses Windows operating system while all other machines run a flavor of Linux.

ACER E-15

This is my personal laptop which runs windows operating system and contains a low spec GPU which is enough for prototyping and training for less iterations also it is always accessible to me.

SPECS:

- RAM: 12GB
- Memory: SATA SSD 256GB
- GPU: NVIDIA GEFORCE 940MX
    i. Transistor Count: 1870 Million
    ii. Memory Type: GDDR5, DDR3
    iii. Memory bus width: 64 Bit
    iv. Max. Amount of Memory: 4GB
    v. CUDA cores: 384

Google Colab

Google Colaboratory, or Colab, allows us to write and execute Python in a web browser, without any extra configurations. This is an interactive notebook based environment which are running on virtual machines provided by Google Research.

Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs and TPUs.

Colab notebooks are stored in Google Drive, or can be loaded from GitHub. Colab is able to provide free resources in part by having dynamic usage limits that sometimes fluctuate, and by not providing guaranteed or unlimited resources. This means that overall usage limits as well as idle timeout periods, maximum VM lifetime, GPU types available, and other factors vary over

time. Colab does not publish these limits, in part because they can (and sometimes do) vary quickly.

SPECS:

- RAM: 12GB
- Memory: 34GB + Google Drive memory
- GPU: NVIDIA TESLA K80
    i. Transistor Count: 7,100 million
    ii. Memory Type: GDDR5
    iii. Memory bus width: 384 Bit
    iv. Max. Amount of Memory: 12GB
    v. CUDA cores: 4992

Kaggle Kernels

Kaggle Kernels are cloud computational environment that can be accessed via web browser that enables reproducible and collaborative analysis without any extra configurations. Kaggle supports Python and R language. Kaggle kernels are also hosted Jupyter notebook service which provide free GPU and TPU for computation.

SPECS:

- RAM: 13GB
- Memory: 34GB + Google Drive memory
- GPU: NVIDIA Tesla P100
    i. Transistor Count: 15,300 million
    ii. Memory Type: HBM2
    iii. Memory bus width: 4096 Bit

iv. Max. Amount of Memory: 16 GB

v. CUDA cores: 3584

The P100 provides 1.6x more GFLOPs and stacks 3x the memory bandwidth of the K80.

Raspberry PI

The Raspberry Pi is a series of small single-board computers comparable to size of a credit card. It contains all the components of a computer and it works as a computer. These were developed in the United Kingdom by the Raspberry Pi Foundation. They are used by hobbyist for creating various projects. As of now Raspberry Pi has model 4 in market with upto 8GB of RAM in its higher configuration.

There are lot more versions of Pi in market, for this project I am using Raspberry Pi 3B with inbuilt wireless LAN and Bluetooth.

SPECS:

- RAM: 1GB
- Memory: size of memory card inserted (32GB class 10)
- CPU: Quad Core 1.2GHz
- Architecture: 64bit
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet

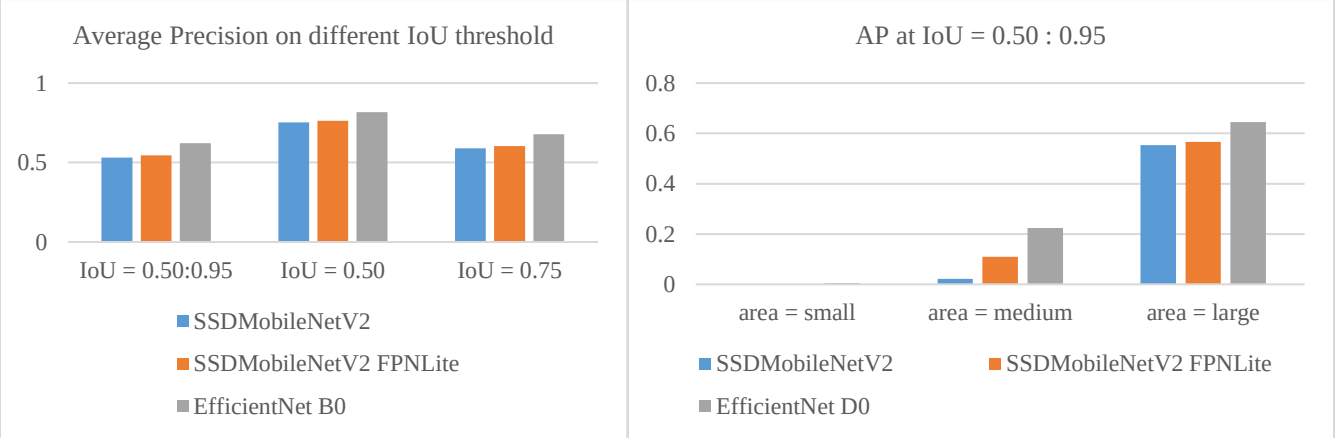
3.3. Comparison of object detection models

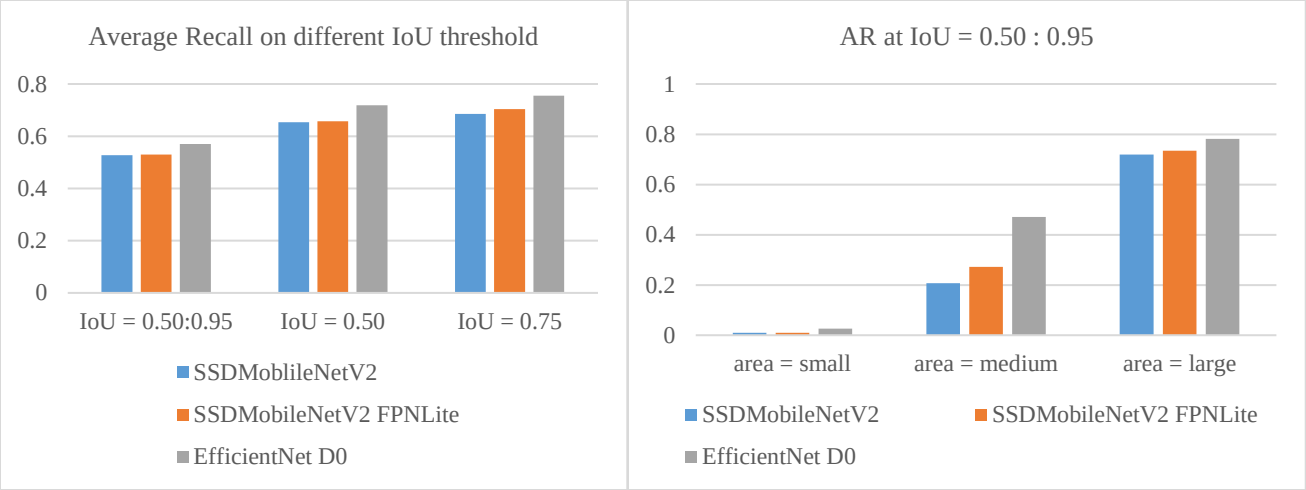
For this project we need to a select a model that has low inference time and greater mAP value for accurate realtime object detection, on this note three models (SSDMobileNetV2, SSDMobileNetV2 FPNLite and EfficientNet D0) were selected as per the information from object detection API, these models have low inference time but in cost of accuracy. These models are used for realtime object detection based on the devices which will be used for inferencing, if device has low compute power we have to use model with low inference time but for more compute powered devices we can deal with more complex model for more accuracy. In this section we will compare these three models based on inference time, mAP score for measuring accuracy and training time. As per the results of comparison we can conclude better about, which model is suitable which device. The below results are based on the training of models for 10000 epochs.

Comparison of object detection metrics

In this part we compare Average Precision (AP), Average Recall (AR) and Mean Average Precision (mAP) for models using thresholds 0.50:0.95, 0.50 and 0.75. Here we also compare these metrics with respect to object size (small, medium and large).

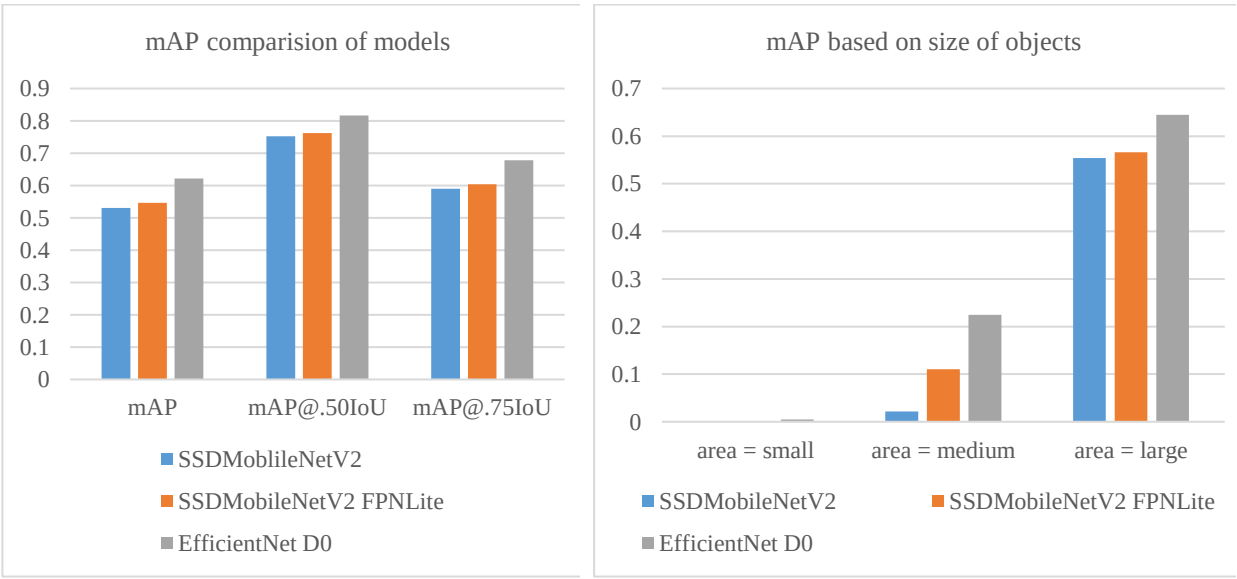
Comparison of Average Precision and Average Recall (AP and AR)





From above graphs we observe that EfficientNet D0 is outperforming SSDMobileNet FPNLite and SSDMobileNetV2 in terms of AP and AR. High precision relates to the low false positive rate and High recall relates to low false negative rate. For small objects all models are struggling hard, EfficientNet has some significant value in average recall. For medium size objects EfficientNet is in lead with large margin followed by SSDFPNLite, margin between SSDMobileNet and SSDFPNLite is low. For larger objects all models are performing well, but EfficientNet is in clear lead while SSDMobileNetV2 and SSDFPNLite are very close.

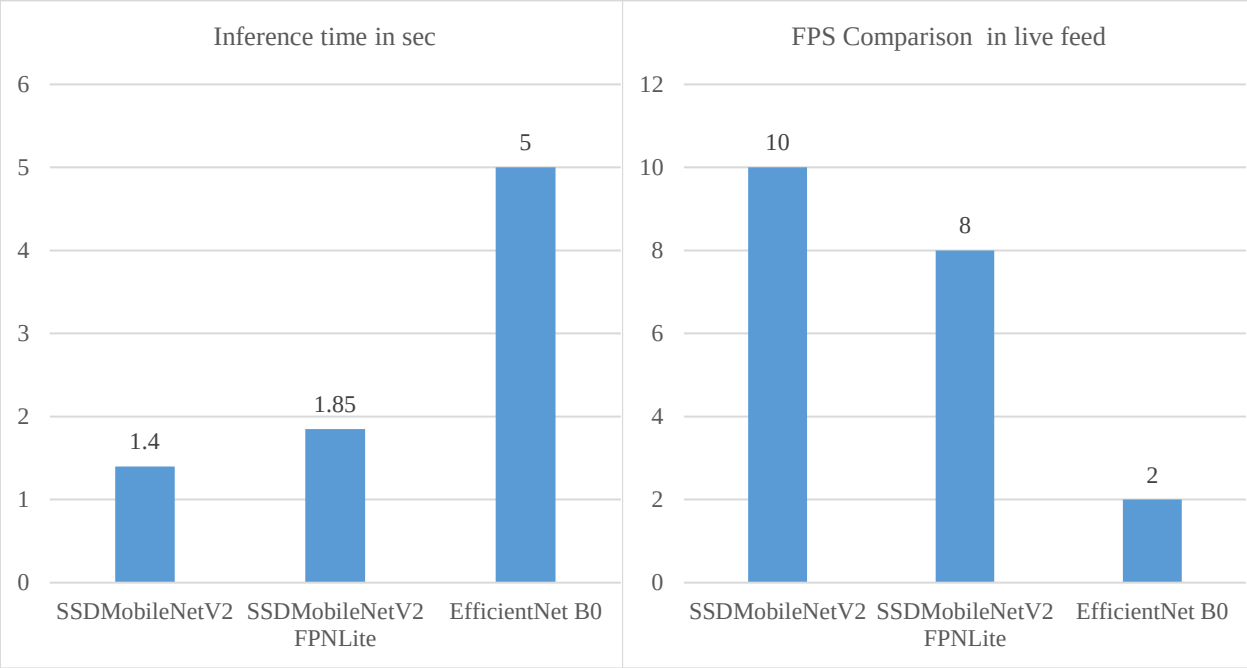
Comparison of mAP metrics.



In terms of mAP scores since EfficientNet is leading in terms of AP and AR therefore it has higher mAP too. Followed by SSDFPNLite then SSDMobileNet both of them are very close in terms mAP score but for medium size objects SSDMobileNet is struggling.

$$EfficientNet\ B0 > SSDMobileNetV2\ FPNLite > SSDMobileNetV2$$

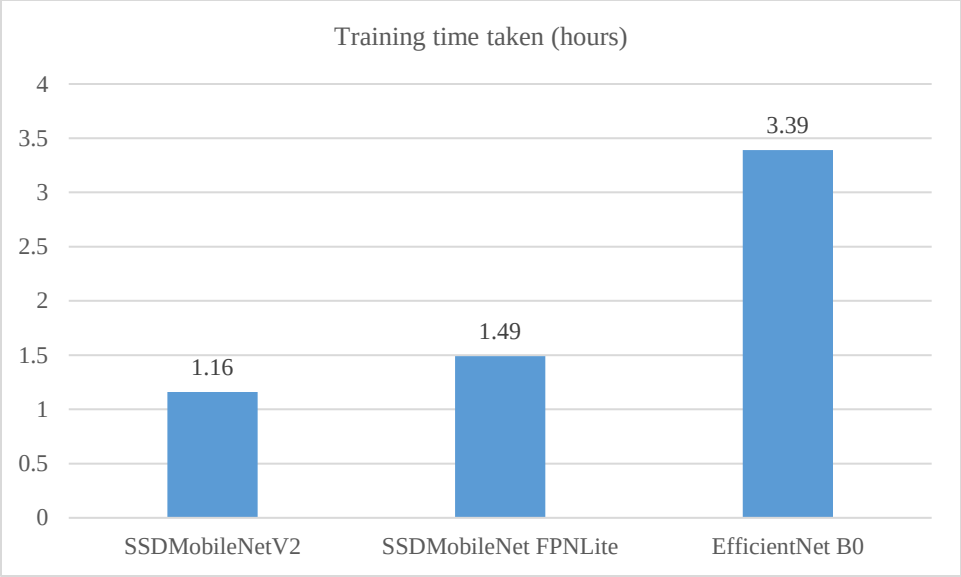
Comparison of Inference time



SSDMobileNet is fastest and can process a single image in 1.4 seconds while EfficientNet takes more than 3 times of that to process an image. So SSDMobileNet is the choice if we need better response time in low computing devices. In an Intel i3 CPU SSDMobileNet on average provides 10 FPS while SSDFPNLite provides 8 FPS and EfficientNet is way behind at 2 FPS.

Comparison in time taken to train models

Here we compared the time required to finish training by a model. We have taken 10000 epochs as standard for comparing training time of a model.

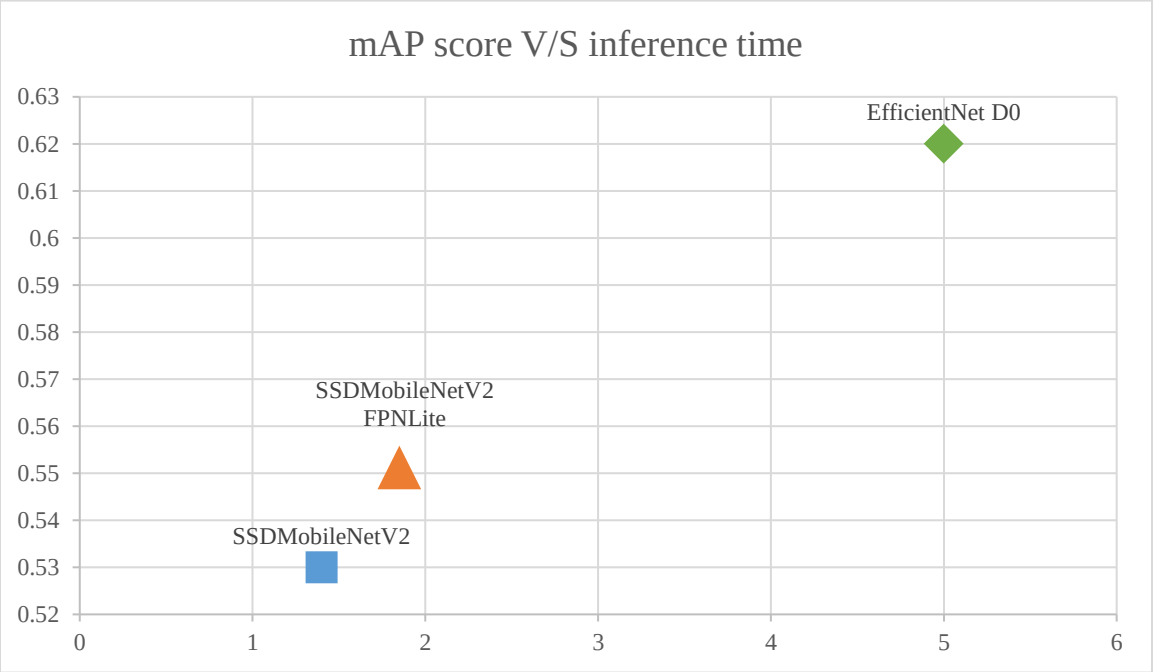


From above graph we can clearly see that SSDMobileNet is fastest to train which take about *1 hour 16 min* to train followed by SSDFPNLite with *1 hours 50 min* and EfficientNet with *3 hours 40 min*. This comparison can be neglected while deriving conclusions, since it does not matter because we do not need to train model every time, we can train once and run infinite. The thing matter is how well model performs in inference device.

4. RESULTS AND DISCUSSIONS

4.1. Results

After all the comparison between models we have to decide which model to be used for final work. From all the comparisons we can conclude that there is no perfect model which we can use, there is tradeoff between inference time and model accuracy. The more accurate model takes more time to infer. The below graph is conclusion of all the comparisons we made.



Since there is tradeoff between accuracy and inference time so our choice of model now depends on inference device we choose. The one property we want most is realtime object detection so in some cases we might have to neglect high accuracy. For low compute power devices like raspberry pi 3 or older we can use SSDMobileNetV2 as it runs faster and gives decent has decent mAP score, it will not be highly accurate but can compete with other in terms of detecting large objects. If we define use case of camera to be static and its angle of coverage (describes the angle range that a camera lens can image) is low, then most of the objects are big enough and can be detected by model easily. If camera has wide line of coverage then

SSDMobileNet can fail or detections might be wrong, to prevent this we can use SSDFPNLite which is much more accurate than SSDMobileNet in terms of detecting medium sized object but it can cause drop in FPS of realtime detection by some amount but it would not be more since from above chart we can infer the difference is very little. If FPS is too low for our use case, then we can use more compute power device like latest raspberry pi model 4. We can also use IP camera for surveillance which will stream feed inside network and can be read using *rtsp* URL and feed to object detection model, this case is ideal since we are re-using hardware which will save cost. The model can be installed in surveillance device which has more compute power and we can run more complex models too. EfficientNet will be useful with very wide coverage range camera since it is good in detecting medium size objects. Very distant objects are hard to pick correctly by these models since they appear small.

4.2. Future Directions

As further research in object detection advances new and better algorithms will be accessible which will improve results of threat detection too. In future TensorFlow lite support for all sorts of models in object detection API will make detections possible in low compute devices. The future research in lightweight object detection will open a doorway for low power devices to become more accurate and fast which concurrently improve the speed and accuracy of detections, this also decreases cost since cheaper low compute device can perform detections using these models. Also the further advancement of research in abstract learning, training a model will be much easier as few training data can train a good object detection model. The project mostly suffers from detection of small objects, as further research advances around this context it opens doors for some potential applications like counting the population of wild animals with remote sensing, detecting anomalies from long distance and surveillance of military

targets etc. Further we can implement image to context which will detects threat and analyses the action performed by it and alerts with the action performed by threat. The further advancement in object detection will lead to betterment of intelligent surveillance systems in terms of performance, speed and accuracy.

5. SUMMARY AND CONCLUSIONS

The aim of this work is to create an intelligent security system using computer vision, this was achieved by using object detection algorithms which detect threats in images taken from camera and alerts user when found one. There are lots of object detection methods out there from traditional object detection using manually extracted features to deep learning based object detection which automatically detects features from images using convolutional neural networks. We use single shot detectors (SSD) which are one stage detector, it enables us to use object detection in real-time while providing reasonable accuracy. Previous two stage detectors are accurate but not ideal for real-time detection and traditional object detection techniques are not accurate enough although they are fast because of optimization techniques applied at that time for old systems, they also could not bear changing lightning conditions. There are lots of version of SSD's but we are interested on those which can provide good FPS in real-time object detection, for this reason three models were chosen based on their inference time record presented in TensorFlow object detection API docs. These models were trained on dataset collected from internet and labeled using LabelImg for 10000 epochs and later tested with evaluation data to find model performance. The performance data of models is then used for comparison of models.

These models were compared on the basis of inference time and accuracy they provide. The main priority feature for comparison was inference time, because we can tackle with some drop in accuracy but it will be hard to run object detection in real-time if inference time was high. From comparisons we find SSDMobileNetV2 is fastest though the problem with this model is it is struggling in terms of detecting smaller and medium sized objects. Although all models we compared struggled to detect small objects but it is struggling with medium sized objects too.

It might not be problem with some security system requirements, but it might be poor while detecting some small threats like snakes. It might also fail to detect if camera has wide line of coverage which zooms out the perspective of environment. In terms of accuracy EfficientNet D0 is leading with large margin but it is resource intensive and takes about 5 secs to infer providing 1-2 fps in real-time detection in an Intel i3 CPU. The third model SSDMobileNet FPNLite is moderate one it provides around 8 FPS which is little less than SSDMobileNetV2 but it provides greater accuracy while detecting medium size objects.

The choice of model to use boils down to the choice of application and choice of device which will be used for inferencing. If we need to detect small objects like snakes or detect distant objects, then we have to choose EfficientNet or SSDFPNLite and device should be powerful enough to maintain around 1 FPS for better results and the choice depends on amount of accuracy we need for the task. In terms of security we can ask how dangerous is threat which we need to detect? For devices like raspberry pi, we are bounded with compute limit so SSDMobileNet is the choice. This model is also great if we want intelligent security setup for cheap.

This work can be later evolved as evolution in object detection continues and new lightweight accurate models come to existence. The detections further can be improved by using more data to train until any abstract deep learning method come to existence. As object detection techniques and research around detection of small objects continues to evolve, then this will positively impact some intense security tasks like detecting military targets.

6. REFERENCES

[1] Z. Zou, Z. Shi, Y. Guo and J. Ye, "Object detection in 20 years: A survey" in arXiv:1905.05055v2, 2019, [online] Available: https://arxiv.org/abs/1905.05055v2.

[2] Joseph Redmon, Ali Farhadi, "The Ancient Secrets of Computer Vision", 2018 [online] Available: https://pjreddie.com/courses/computer-vision

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, "You Only Look Once: Unified, Real-Time Object Detection", 2016 [online] Available: https://pjreddie.com/media/files/papers/yolo.pdf

[4] Rokas Balsys, "Yolo v3 with TensorFlow 2", 2020, [online] Available: https://pylessons.com/YOLOv3-TF2-introduction/

[5] Adrian Rosebrock, "Increasing webcam FPS with Python and OpenCV", 2015, [online] Available: https://www.pyimagesearch.com/2015/12/21/increasing-webcam-fps-with-python-and-opencv/

7. PUBLICATIONS BY THE CANDIDATE

[1] Detection Dataset, “Monkey, Cat and Dog detection” on Kaggle Dataset [online] Available: https://www.kaggle.com/tarunbisht11/yolo-animal-detection-small

[2] Code for Object Detection using YOLOv3, “Object Detection using YOLOv3 in TensorFlow 2” on GitHub [online] Available: https://github.com/tarun-bisht/object-detection-yolov3

[3] Code for Object Detection using Object Detection API, “TensorFlow Object Detection” on GitHub [online] Available: https://github.com/tarun-bisht/tensorflow-object-detection

[4] Code for security camera, “Intelligent Security Camera” on GitHub [online] Available: https://github.com/tarun-bisht/security-camera

[5] Code Notebook and discussion for generating training data, “Create your Image dataset” on Kaggle and Google Colab [online] Available: https://www.kaggle.com/data/186747

[6] Notebook to train an efficientdet_d0 using transfer learning and object detection API, “Detect Monkey Cat and Dog” on Kaggle [online] Available: https://www.kaggle.com/tarunbisht11/detect-monkey-cat-and-dog

8. ACKNOWLEDGEMENT

I would like to express my gratitude and appreciation for all teachers of Computer Science Department whose guidance, support, faith and encouragement has been invaluable throughout this study. I would like to thank our HOD, Dr. Ashish Mehta for coming up with this wonderful idea for project.