Edge Analytics on Resource Constrained Devices

Sean Savitz

School of Computer Science and Informatics,
Cardiff University, UK
E-mail: savitzsa@cardiff.ac.uk

Charith Perera

School of Computer Science and Informatics,
Cardiff University, UK
E-mail: pererac@cardiff.ac.uk

Omer Rana

School of Computer Science and Informatics,
Cardiff University, UK
E-mail: ranaof@cardiff.ac.uk

Abstract: Video and image capture cameras have become an important type of sensor within the Internet of Things (IoT) sensing ecosystem. Camera sensors can measure our environment at high precision, providing the basis for detecting more complex phenomenon in comparison to other sensors e.g. temperature or humidity. This comes at a high computational cost on requirements of CPU, memory and storage resources, and requires consideration of various deployment constraints such as lighting and height of camera placement. Using benchmarks, this work evaluates object classification on resource constrained devices, focusing on video feeds from IoT cameras. The models that have been used in this research include MobileNetV1, MobileNetV2 and Faster R-CNN that can be combined with regression models for precise object localisation. We compare the models by using their accuracy for classifying objects and the demand they impose on the computational resources of a Raspberry Pi. Various IoT deployments are investigated by comparing the probability scores of classifying chosen objects using different camera placements. We conclude that the Faster R-CNN model that is configured with the InceptionV2 regression model has the highest accuracy. However, this is at the cost of additional computational resources. We found that the best model to use for object detection functionality on the Raspberry Pi is the MobileNetV2 model paired with the SSDLite regression model. This results in the highest accuracy and probability score for object classification, in comparison to other mobile-friendly models considered in this work, whilst using the least amount of computational resources.

Keywords: Internet of Things, Edge Computing, Edge Analytics, Resource Constraint Devices, Camera Sensing, Deep Learning, Object Detection.

Reference to this paper should be made as follows: Sean Savitz, Charith Perera, Omer Rana, (2021) 'Edge Analytics on Resource Constrained Devices', *International Journal of Computational Science and Engineering*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes:
Sean Savitz graduated with a First class BSc degree in Computer Science at Cardiff University, UK. He is currently reading an MSc in Cyber Security at the University of Southampton.

Charith Perera is a Senior Lecturer at Cardiff University, UK. His research interests are the Internet of Things, Sensing as a Service, Privacy and Sensing Infrastructure.

Omer Rana is a Professor of performance engineering at the School of Computer Science and Informatics, Cardiff University, UK. His research interests include distributed systems and scalable data analysis.

1 Introduction

Internet of Things (IoT) [1, 2] refers to the ever-growing network of physical objects that feature an IP address

for internet connectivity [3]. This provides connectivity and communication between objects and other internet-enabled devices and systems. Over the last five years a drastic increase in the use of IoT devices, particularly within the average consumer household, has occurred. A major influence has been the rise of cloud-based solutions [4] to process large amounts of data generated from IoT devices, as well as the low cost of entry into the 'smart' industry. IoT devices have been deployed across a number of applications, such as smart parking, structural health monitoring, smart lighting etc.

Cameras are becoming an important type of sensor within the IoT sensing ecosystem [5]. Camera sensors measure our surroundings at a high level of detail and provide the basis for detecting more complex phenomenon in comparison to other sensors e.g. temperature or humidity. However, this comes at a high computational cost, requiring significant CPU, memory and storage capacity. Cameras and sensors are vital to an IoT network and their sensing accuracy has continued to increase over recent years. As a result, they have contributed to new advancements, as well as the development of useful applications such as Crowd-Based Mobile Cloud [6]. This technology aims to utilise the ever growing amounts of data that are collected from sensors in smartphones in order to provide *meaningful* insights about user behaviour [7]. These include identification of complex social patterns, as well as detecting location change patterns to predict a user's future location and activities. Another example of how this can be applied in practice with an environmental focus is to aid scientists with detecting temperature, Nitrogen Dioxide (NO_2) levels and other chemical leaks using mobile sensors in real-time, from varying locations. [8].

We investigate various mechanisms for deploying resource constraint IoT cameras to detect objects using different deep learning pre-trained models. To achieve this, we develop an end-to-end open-source IoT stack, that can be deployed and reconfigured for different purposes over time. Our primary focus is to implement and deploy a Raspberry Pi4 based IoT camera network that transmits real time video data to a selection of neural network (NN) models that have been pre-trained on the MS-COCO image dataset. A performance evaluation of various NN models and their suitability for deployment on resource-constrained IoT sensing devices is reported. Our research utilises the following models: MobileNetV1 [9], MobileNetV2 [10] and Faster R-CNN [11]. Each model behaves distinctly in a network architecture, training strategy and optimisation function. As a result, models such as Faster R-CNN that provide state-of-the-art accuracy comes at a computational cost (requiring greater number of computational resources to execute). On the other hand, mobile-friendly models are smaller in size and require less memory and are therefore more suited to resource-constrained environments.

We use these models to produce a comparative analysis of the accuracy of detecting object(s) in real-time. Another contribution of our research is to develop a resource monitoring agent that compares a broad range of resource measurements and includes the performance, RAM usage, CPU load and CPU usage of each model on a Raspberry Pi (RPi) device (used as a proxy for an IoT device).

This paper is structured as follows: Section 2 presents related work along with a context for this paper. Section 3 discusses the selection of our approach and the experiments that have been performed. Section 4 describes the edge-based analytics system implementations that we compare, with an evaluation of these systems in Section 5. Section 6 includes a discussion of the results and the key observations from these results, followed by concluding statements in Section 7.

2 Background and Related Work

The Internet of Things (IoT) concept was coined by a member of the Radio Frequency Identification (RFID) development community and has grown in use significantly in recent years [12]. It is widely accepted that IoT is an extension of the internet into the physical world for interaction with physical entities. An IoT device is defined as 'a hardware component connecting an entity to the digital world' [13]. Commonly agreed characteristics of IoT devices across the industry include their smaller/ embedded size, their mobility, and their use of Cellular or Wi-Fi communication methods. Although recently, there has been an increase in popularity of new communication methods in IoT networks, such as Zigbee, Z-Wave, Bluetooth Low Energy (LE), Narrow Band IoT (NB-IoT), LoRAWAN, SIGFox, etc. IoT devices often incorporate small, low-cost and highly efficient single motherboard computer designs, utilising cloud computing facilities to process large data [14]. Practical deployment of IoT devices remains a challenge [15], i.e. identifying whether to place cameras and sensors at ground level or at a height, in object detection tasks remains an error prone process.

2.1 Computer Vision and Deep Learning

Current computer vision applications make significant use of deep learning techniques (as illustrated in Figure 1), primarily using convolutional neural networks (CNNs) [16] for identifying and recognising objects. A CNN is the most representative model of deep learning [17]. A typical CNN architecture, such as VGG16, encompasses a multi-layered architecture, as presented in Figure 2.

As illustrated in Figure 2, each of the three layers of a CNN are known as a feature map – a 3D matrix of pixels which represent different colour channels (e.g. RGB). Different types of transformations are conducted on these feature maps, such as filtering and pooling. The filtering operation (or convolution) convolutes a

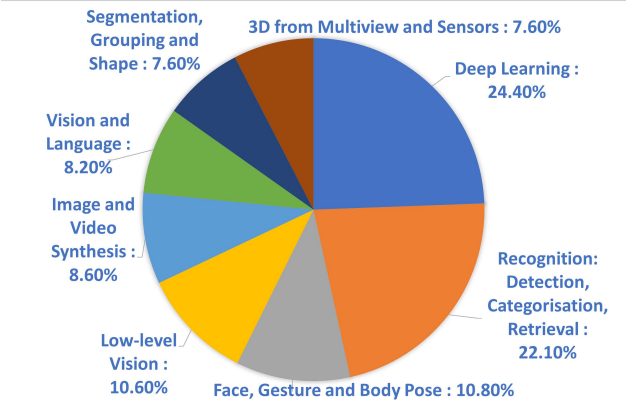

Figure 1 The Breakdown of Subject Areas in the 1300 Computer Vision Accepted Papers in 2019 [18]

filter matrix (learned weights) with the values of a receptive field of neurons and incorporates a non-linear function such as ReLU to obtain final responses. Its hyperparameters include the filter size F and stride S. Stride alters the amount of movement of the input image or video; a stride of 1 is a shift of one pixel of the image at a time. The pooling layer incorporates a down-sampling operation that carries out spatial invariance on the convolution layer. This function typically uses maximum or average pooling that summaries the responses of a receptive field into one value, thus producing more robust feature descriptions. Interleaving between the convolution and pooling layers leads to an initial feature hierarchy that can be fine-tuned in a supervised approach with the use of fully connected layers to adapt to varying visual tasks. VGG16 often has 13 convolutional layers, 3 fully connected layers, 3 max-pooling layers and a softmax layer that is incorporated for the classification of objects.

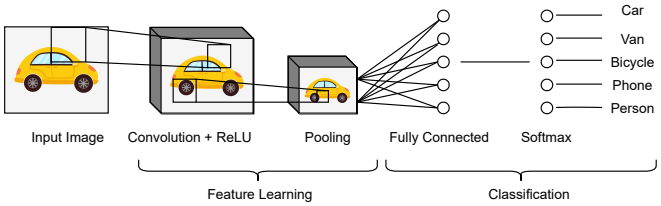

Figure 2 A CNN that Processes an Input Image and Classifies the Object [11]

Detecting an object using a CNN involves localising the object in the image and identifying which class this object belongs to [10]. These locations are typically represented by bounding boxes that are placed around each detected object, e.g. in pose estimation [19], vehicle detection [20, 18] and video surveillance [21]. In [22], machine learning is used to detect and classify human behaviour interactions in a visual surveillance task. Other research in [23] describes detection of curb ramps on Google street view, with a focus on presenting to wheelchair users streets that have adequate accessibility. Other examples include autonomous driving, focusing on identifying pedestrians and cyclists to prevent collisions [24]. However, object detection can be computational expensive, especially in comparison to image classification [18]. This complexity results from the numerous potential object locations, called 'proposals' that must be processed. The exact location of an object must be refined in real-time for precision. Nevertheless, various open-source and large-scale image datasets are available that can be used in conjunction with deep learning techniques to support object detection.

A number of realistic data sets exist to support object detection, e.g. 'Labelled Faces in the Wild' [25] and detection of pedestrians with the 'Caltech Pedestrian Dataset', the latter contained 250,000 frames with a total of 350,000 bounding boxes, and 2300 unique annotated pedestrians [26]. In addition, various 'benchmark' datasets, called the PASCAL Visual Object Classes (PASCAL VOC), have also been reported. These datasets contain 20 categories of objects that are spread over 11,000 images [27]. More recently, the ImageNet dataset was developed, which is organised according to the WordNet hierarchy. Although ImageNet continues to grow in size, it currently contains 21,841 synsets with 14,197,122 images, averaging 650 images per synset [28].

We use a selection of neural network models that have been pre-trained on the Microsoft Common Objects in COntext (MS COCO) dataset. This image dataset contains 91 common object categories with over 200K images and 2.5M labelled instances, in which 82 objects have more than 5K labelled instances. In comparison to the ImageNet dataset, MS COCO has fewer categories, despite having a greater number of object instances per category. MS COCO strives to find non-iconic images containing objects in their natural context and therefore provides images with a high level of contextual information in a wide variety of environments. MS COCO contains an average of 7.7 object instances per image, which is greater in comparison to other influential datasets such as ImageNet that averages 3.0 and PASCAL VOC that averages 2.3 [29]. This provides our selected neural network models that have been pre-trained with the MS COCO dataset, with the ability to learn detailed object patterns for precise 2D localisation [29].

Low-light environments cause complications in object detection. Existing studies have primarily focused on image enhancement [30] or improving night vision surveillance that requires costly hardware [31]. Remarkably, less than 2% of the images provided by the influential image datasets, including MS COCO, are captured in low-light [32]. New developments in 2018, however, led to the creation of the Exclusively Dark dataset [32], which contains 7,363 low-light images that possess especially low-light environments to twilights.

2.2 Neural Network Models for Mobile Devices

Modifying deep neural network architectures to create an optimal balance between accuracy and performance, whilst reducing its computational resource requirements for execution, has been an integral part of research in recent years [10]. This has led to major improvements over early designs such as ResNet [33], GoogLeNet [34] and VGGNet [35]. Modifying connectivity structures in internal convolutional blocks has also improved accuracy, such as with the introduction of sparsity [36] or using the ShuffleNet model [37].

Neural network models such as Faster R-CNN are computationally intensive to execute. Computational cost can also be high due to the use of optimisation techniques to adapt the neural network architecture – which can involve the use of evolutionary search techniques (e.g. genetic algorithms) or reinforcement learning [38]. The availability of mobile-friendly neural network models, such as MobileNetV1 and MobileNetV2, require fewer computational operations and and are therefore more tailored to resource-constrained environments [10]. In sections 2.2.1 and 2.2.2, MobileNetV1, MobileNetV2 and Faster R-CNN are presented, and compared in section 2.2.3.

2.2.1 MobileNet Architecture

Sandler et al. [10] introduce MobileNetV2, which extends MobileNetV1 using depthwise convolution as efficient building blocks, providing two new features: linear connections between layers, as well as shortcut connections between bottlenecks [10].

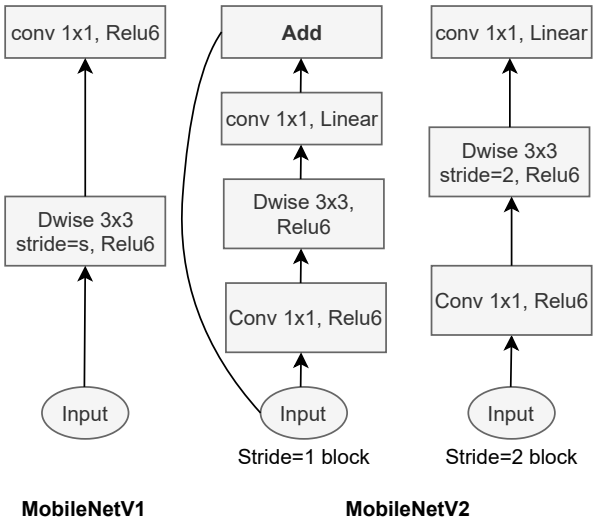

Figure 3 MobileNetV1 and MobileNetV2 Convolutional Blocks [10]

MobileNetV1 has two layers. The first layer performs lightweight filtering by applying a single convolution filter per input channel, called a depthwise convolution. The second layer is a 1x1 convolution, called a pointwise convolution which is responsible for building new features through computing linear combinations of input channels. In MobileNetV1, ReLU6 is used, making use of low-precision computation. On the other hand, MobileNetV2 utilises two types of blocks; one is residual with a stride of 1 and the other with a stride of 2, which is used for downsizing. The first layer comprises of a 1x1 convolution with ReLU6, the second layer is the depthwise convolution, with the final layer involving another 1x1 convolution without non-linearity.

The size of the models, number of parameters and computational complexity of these two MobileNet models are compared in [10, 9]. They use MACs (multiply-accumulate) operations to measure the size of the models by identifying how many calculations are used in processing a single 224x224 pixel image. They suggest that MobileNetV2 is much smaller in size than MobileNetV1 and is less computationally expensive.

Table 1 Size and Computational Cost: MobileNetV1 vs. MobileNetV2

Version	MACs (millions)	Parameters (millions)
MobileNetV1	569	4.24
MobileNetV2	300	3.4

MobileNetV1 and MobileNetV2 are typically paired with a Single Shot Detector (SSD), which is used in the regression task in object detection. It aids in locating the part of the image that has the highest probability of containing an object. This is achieved by combining prediction across multiple feature maps and applying convolution filters to detect objects. SSD is faster and more accurate than previous state-of-the-art single shot detectors, such as YOLO [39]. Previous work [10, 40], incorporates a mobile-friendly variant of SSD with MobileNetV2, namely SSDLite, that replaces the regular convolutions in SSD with depthwise separable convolutions. MobilNetV2+SSDLite is 35% faster than MobileNetV1 SSD in which the computational cost, in terms of Multiply-Accumulate and size of the model is significantly decreased, whilst achieving the same accuracy [10].

2.2.2 Faster R-CNN Architecture

Ren et al. [41] present their Faster R-CNN model, an extension to two existing algorithms [42, 43]. Faster R-CNN utilises a region proposal based framework, based on scanning the entire scenario and then focusing on the region of interest. Faster R-CNN is composed of two modules; a deep convolutional neural network that proposes regions and a Fast R-CNN network that uses the proposed regions to classify objects. .

Table 2 Size & Computational Cost of MobileNetV2 with SSD and SSDLite for predicting 80 Classes [10]

	Params	MAdds
SSD	14.8M	1.25B
SSDLite	2.1M	0.35B

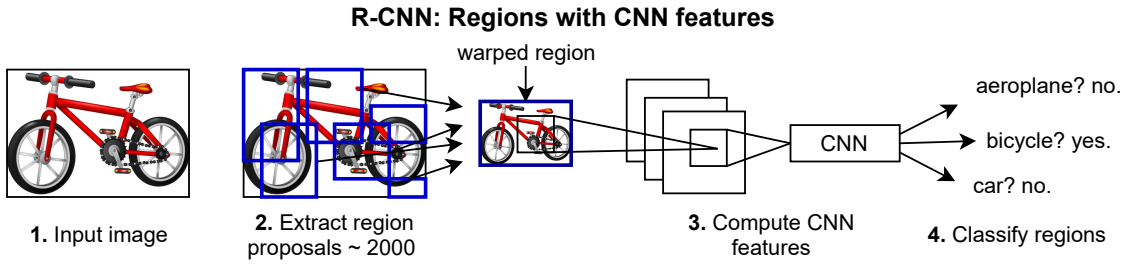

Figure 4 The R-CNN Model Architecture [42]

As presented by [41, 42], R-CNN adopts a selective search method that extracts precisely 2000 candidate region proposals from each image. This significantly decreases the number of regions that need to be classified. Each proposed region is warped into a fixed resolution and fed into a CNN module that outputs a 4096-dimensional feature vector as the final representation. In this case, the CNN acts as a feature extractor that provides a high-level, semantic feature representation of each region proposal. These features are fed into a Support Vector Machine (SVM) that classifies the presence of the object in the 2000 region proposals. The final stage involves predicting offset values which are used to adjust the bounding boxes that represent the identification of an object to increase precision.

However, the generation of the 2000 regions proposal with selective search is time-consuming at around 2 seconds and R-CNN cannot be utilised for detection in real-time as it takes around 47 seconds per image [44]. As a result, Girshick et al. [43] addresses limitations of R-CNN by developing a faster object-detection algorithm called Fast R-CNN.

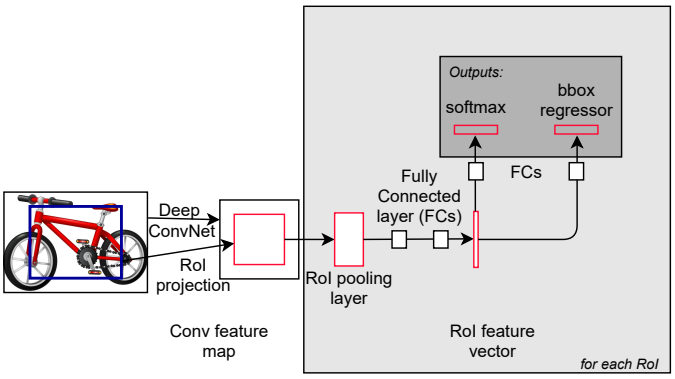

Figure 5 The Fast R-CNN Architecture [43]

Instead of extracting 2000 region proposals, the Fast R-CNN network processes the entire image through several convolutional and max-pooling layers to produce a convolutional feature map. Region proposals are identified and then warped into squares. For each object-proposal, a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map that is fed into a sequence of fully connected layers. From the RoI feature vector, a softmax layer is used to predict the class of the proposed region, as well as the offset values to localise the object in the image [11, 43]. As a result, training in Fast R-CNN is on average 8.75 hours, compared to R-CNN's 84 hours.

Table 3 Performance Comparison Carried out by TensorFlow [41]

Model Name	Speed (ms)	COCO mAP
MobileNetV1 + SSD	30	21
MobileNetV2 + SSD	31	22
MobileNetV2 + SSDLite	27	22
Faster R-CNN + InceptionV2	58	28

Using selective search to find region proposals in R-CNN and Fast R-CNN is a particularly slow and time-consuming process that affects the performance of the network. As a result, Ren et al. [41] introduces Faster R-CNN in which their goal was to create a model that shared computation with a Fast R-CNN network, whilst removing the use of selective search.

Similar to Fast R-CNN, Faster R-CNN processes the entire image through several convolutional and max-pooling layers to produce a convolutional feature map. The feature matrix is then passed into the Region Proposal Network (RPN) layer, which is a convolutional network that simultaneously predicts object bounds and an *objectness* score at each position. The RPN is trained to generate high-quality region proposal and uses 'attention' mechanisms to inform the unified network where to find these, which are subsequently used by Fast R-CNN for detection. As a result, the region proposals created from the RPN are fed into the RoI pooling layer that extracts a fixed-length feature vector from the feature map.

The final stage of Faster R-CNN's architecture is an R-CNN whose goal is to: (i) classify region proposals into classes, and (ii) adjust the bounding boxes that surround the identified object for precision. Faster R-CNN has a test-time of 0.2 seconds on average, compared to R-CNN's 49 seconds for R-CNN and and 2.3 seconds for Fast R-CNN [44].

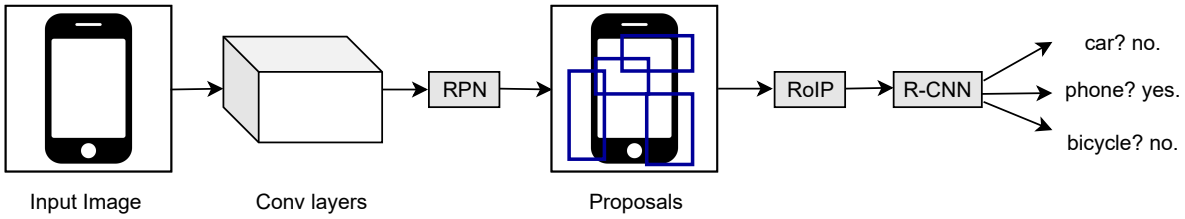

Figure 6 The Faster R-CNN Architecture [41]

2.2.3 Comparison of the Models

In this section we compare MobileNetV1, MobileNetV2 and Faster R-CNN – identifying the probability of classifying a chosen object(s) correctly, as well as the impact of each model on computational resources. TensorFlow 'Model Zoo' [41] provides a collection of neural network models that have been pre-trained on an abundance of image datasets. Selected neural network models are pre-trained on the MS COCO dataset and paired with regression models that allow for more precise object localisation. These models include MobileNetV1+SSD, MobileNetV2+SSD, MobileNetV2+SSDLite and Faster R-CNN+InceptionV2. To compare these models, TensorFlow [45] uses benchmark tests to find the speed (latency) of each model whilst inferencing on 600x600 images, as well as the mAP (mean average precision), which is a detector-specific measurement for MS COCO. Their investigation found that MobileNetV2+SSDLite was the fastest model and Faster R-CNN + InceptionV2 had the highest detection accuracy.

Sandler et al. [10] compare MobileNetV1 and MobileNetV2 models that focused on accuracy, size, computational cost and performance. They ran each model on a Google Pixel1 phone and classified objects using the ImageNet dataset. Their results found that MobileNetV2 was more accurate, smaller in size with fewer parameters, required fewer computational resources and had a lower latency compared to MobileNetV1, as presented in Table 4.

[10] compares MobileNetV1+SSDLite and MobileNetV2+SSDLite models that have been pre-trained on the COCO dataset. They found that the MobileNetV2+SSDLite model achieves competitive accuracy with significantly fewer parameters and reduced computational complexity. Multiply-Add operations are used in their work to measure the impact of each model on a Google Pixel 1 phone. Hollemans [46] has identified the performance of the MobileNetV1 and MobileNetV2 models by monitoring the number of frames per second. They run each model on various Apple devices, by running inferences on 224x224 pixel images using a double-buffering approach, combining the use of a CPU and a GPU. We instead focus on resource demands of a RPi – a more realistic environment for many IoT devices.

In [47], the performance of the Faster R-CNN model using the InceptionV2 regression model is investigated. The pairing of these two models leads to a reduction is size of the Faster R-CNN model [11]. Running Faster R-CNN+InceptionV2 on a laptop with Nvidia GeForce GTX 1060 GPU resulted in a frame rate of 1.6 seconds. In comparison, MobileNetV1 and MobileNetV2 models achieved significantly higher performance with an average of 161 and 199 frames per second, respectively [46].

Based on this survey, we have found that although Faster R-CNN is a more computationally complex model than MobileNetV1 and MobileNetV2, its network architecture provides higher accuracy in an object detection task. Our approach uses a monitoring agent to quantify resource requirements for model execution on a RPi, for the classification of objects within a live video feed. The processing power, small size and low-cost of a RPi has made it desirable for use in an IoT network. RPi4 has had a full-chip redesign in which benchmark tests have shown a significant increase in performance [48]. Ansari et al. [49] present a security alarm system that monitors motion and sets off an alarm if a person is detected. Subsequently, the camera module that is connected to the RPi takes photos and video which is sent to a cloud server. The Meter Maid Monitor project [50] uses an RPi with a camera module that forwards live traffic video into a convolutional neural network, in order to detect whether a ticket warden is present. In the event that there is more than 75% chance of identifying a ticket warden, a text message is sent to the user.

3 Systems Design

The design of our IoT system incorporates three stages; Data Collection, Data Processing and Data Presentation. The Data Collection stage uses sensors,

Table 4 Performance Comparison: MobileNetV1 vs. MobileNetV2 on the ImageNet Dataset [10]

Model Name	Top 1 Accuracy (%)	Parameters (Millions)	MAdds (Billions)	Speed (ms)
MobileNetV1	70.6	4.2	575	113
MobileNetV2	72	3.4	300	75

Table 5 Performance Comparison: MobileNetV1+SSDLite Vs. MobileNetV2+SSDLite on the COCO Dataset [10]

Model Name	Speed (ms)	Parameters (Millions)	MAdds (Billions)	Speed (ms)
MobileNetV1 + SSDLite	22.2	5.1	1.3	270
MobileNetV2 + SSDLite	22.1	4.3	0.8	200

Table 6 The Maximum Frames Per Second (FPS) When Using MobileNetV1 and MobileNetV2 Models' On Apple Devices [46]

Version	iPhone 7	iPhone X	iPad Pro 10.5
MobileNetV1	118	162	204
MobileNetV2	145	233	220

actuators and cameras as edge devices [51, 52]. Data Processing is carried out by a central hub, also realised using a Raspberry Pi 4 (RPi) device – which subsequently acts as a gateway in our IoT system as illustrated by Figure 7. A Python script was deployed on the RPi for processing a real-time video feed using the three models: MobileNetV1, MobileNetV2 and Faster R-CNN. These models are pre-trained on the MS COCO dataset for object detection. The Python script automates the collection of probability scores for object classification across the three models. Our work made use of a camera module that connects to the RPi and feeds live video into each of the selected models. A GrovePi+ was also incorporated that fits on to the RPi via the GPIO pins, to provide a connection between the RPi and edge devices [53]. Finally, the data was processed and presented in a user-friendly interface in the Data Presentation stage, explored further in section 3.2.

3.1 IoT Cloud Platform

The available memory within an RPi is constrained by the inserted microSD card. As a result, in our IoT system the RPi processes data at the edge which is then sent to the cloud. To provide seamless connectivity in the data stack of our IoT network, the original plan was to utilise Eclipse Kura on the gateways and Eclipse Kapua as the cloud platform. However, due to limited RPi4 documentation, we utilised the open-source IoT cloud platform 'ThingsBoard' (thingsboard.io) – which combines scalability, fault-tolerance and performance, in addition to support for data presentation in a user-friendly manner using custom widgets. A lightweight communication protocol was needed to provide data exchange between ThingsBoard and the RPi. We used the Message Queuing Telemetry Transport (MQTT) protocol as it is a lightweight publish/ subscribe messaging transport protocol and is useful for connections in remote locations where a small code footprint is required or where network bandwidth is at a premium. We are processing the images on the edge (i.e., RPi) and only sending the extracted information (meta data) in text format to the cloud (i.e., ThingsBoard).

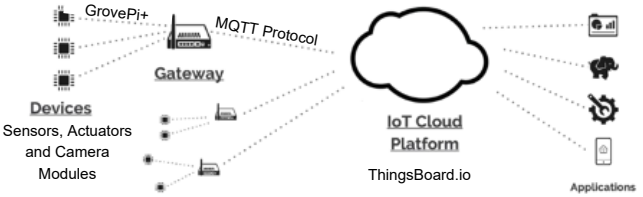

Figure 7 Overall IoT System Architecture

We used *MobileNetV1*, *MobileNetV2* and *Faster R-CNN* in this work. These models are typically paired with regression models that aid the precise localisation of objects. We have used variants of the aforementioned neural network models, namely MobileNetV1+SSD, MobileNetV2+SSD, MobileNetV2+SSDLite and Faster R-CNN+InceptionV2, that permits this research to be aligned with other literature. The pairing of Faster R-CNN and InceptionV2 was incorporated as this was the variant of the Faster R-CNN model that was smallest in size. Selecting models that have been pre-trained on the MS COCO dataset was attractive as MS COCO presents images with more contextual information than other influential datasets and contains more labelled object instances per image than ImageNet or PASCALVOC. In section 3.1.1, the objects that we have chosen to classify in our investigation are discussed.

3.1.1 Objects Chosen

We developed a Python script to identify bicycles, road vehicles, e.g. cars, trucks, and buses, and mobile phones for the purposes of this investigation. This section discusses the potential use cases and motivations for our choices.

Bicycle detection: The detection of bicycles is paramount to collision avoidance systems within vehicles on the road. Volvo developed a cyclist detection system [54] that incorporates a grille-mounted radar system and a high-resolution camera to monitor the location of cyclists around the car. The radar detects the objects and tracks the vehicle's distance to them whilst the camera determines the type, size and height of the object. In the event that collision risk is imminent, the system sounds an audible alarm, causing the breaks to be automatically applied.

Vehicle detection: The detection of vehicles on the road, such as cars, trucks, and buses can also be used to analyse traffic patterns [20, 18]. Sravani et al. use their traffic monitoring data to investigate and analyse queue-lengths, traffic-flows at junctions, speed distribution data, as well as space and time occupancy rates on the road [20]. In turn, this data can be used

by structural engineers during the creation of new roads or to determine where to place speed restrictions and traffic lights on the road. Distinguishing between road vehicles also has its use in restricted areas such as bus lanes. For the purpose of detecting road vehicles, our work investigated the probability score of each model classifying large toy cars, trucks and buses, as shown in Figure 8.

Mobile phone detection: Mobile phones can be used to take pictures or video. The detection of mobile phones has its use in a vicinity where security is of utmost importance and is therefore not allowed. In the event that a phone is detected, then an alarm will ring informing staff where the phone is located. Another use of a mobile phone detector is in the workplace that reminds members of staff that they have left their phone behind when they leave the office.

Figure 8 Bicycle, Mobile Phone and Road Vehicles Used in this Study

3.2 Comparative Analysis of Models

To measure the performance of each model, we used number of frames per second (FPS) as a metric – an approach that has been utilised in [46, 47]. We partially adopted the measurements used in [55], by considering RAM usage, CPU load and CPU usage. We also continuously observed the CPU core temperature of the RPi when running each model to ensure that the temperature did not increase to over 85^{o}C. Such high temperature would lead to a significant decrease in performance in what is termed as throttling [56]. This would have therefore led to unreliable results and an invalid outcome. The resulting comparative analysis is presented in Figures 12, 13, 14, 16 and 17.

3.3 Deployment Scenarios

The placement of IoT sensors also has an influence on the analysis outcome. We placed the camera and light sensor of the IoT system 80cm away from the chosen

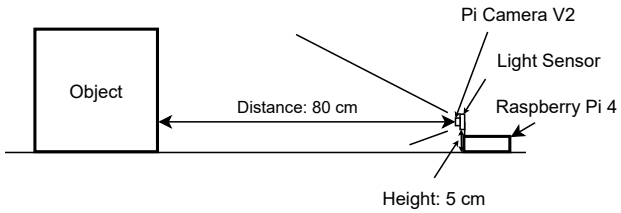

Figure 9 Camera and Light Sensor Placed 80cm Away From the Object and on the Ground

object(s), as illustrated in Figure 9. We also investigated placing the camera at a height of 2.5 metres at a range of distances away from the chosen object(s) during testing. This is important in practice when having to determine whether to place edge devices at ground level or at a height, e.g. at the end of a long pole, for optimal object detection. These placement scenarios provide an important insight into best practices for the deployment of edge devices in an IoT network, especially where these edge devices include video cameras.

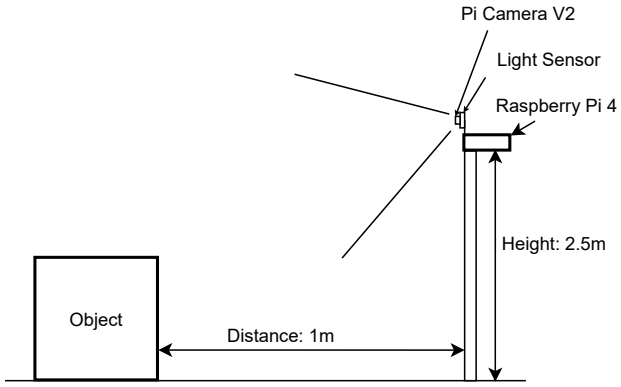

Figure 10 Camera and Light Sensor Placed at a Height of 2.5 metres

A light sensor was also used to assess the quantity of light and its impact on object classification accuracy. Low lighting would otherwise lead to a reduced probability score as a result, as only 2% of the MS COCO dataset contains images captured in a low-light environment. The importance of white (vs. yellow) light for improving accuracy of face recognition has also been highlighted by [57].

4 Implementation

We used the following hardware throughout this project; a Raspberry Pi4 running the Raspbian operating system, a GrovePi+, a Pi Camera V2 module and a light sensor. Python version 3.6 was used for the numerous libraries required for deep learning and computer vision functionality. In particular, the TensorFlow library was vital, which is an open-source machine learning framework to develop and train models, while the OpenCV library was used for image recognition and identification.

Table 7 Metrics Used for Comparative Analysis

Metric	Description
Probability Score	Measures the probability and confidence that chosen object(s) have been detected (Array of 10 Floating Points between 0 and 1).
Performance (FPS)	Measures the number of image frames per second sent to the neural network model for processing (Frames per Second (FPS)).
RAM Usage	Measures the amount of memory currently used when running each neural network model.
CPU Load	Measures the number of processes which are executed or waiting to be executed by the CPU.
CPU Usage (%)	Measures CPU utilisation. This is the amount of time required to process instructions while each neural network model is used.
CPU Temperature (^oC)	Measures increase in CPU temperature, caused by running each neural network model.

Table 8 Commands Used in the Resource Monitoring Agent

Resource Measurement	Command
RAM Usage	free-m
CPU Load	uptime
CPU Usage	mpstat
Temperature	vcgencmd measure_temp

The Python script utilises a category index that maps integers that are directly associated with objects from the MS COCO dataset to the name of that object. For example if the neural network predicts that the object detected is '2', this will therefore correspond to a 'bicycle'. The first feature that we developed incorporates these integers as an input to the system and provides functionality only to detect those object(s) that may be required for a particular purpose. This permits the code to be dynamic in nature. This can be expanded to any of the 91 objects presented in the MS COCO dataset.

A 2-dimensional 10x10 matrix is used to store the integers that relate to the objects that have been detected. A 2-dimensional 10x10 matrix is also used to store the probability score of each object that has have been detected. During detection, the object's integer value with the highest probability score automatically shifts to the first element of the first row of the matrix. This is therefore presented as 'classes[0][0]' and 'scores[0][0]'. We also output the exact location of the detected object using a multidimensional array, as well as the current number of detected objects.

In order to identify the demand imposed by each of the selected neural network models, a resource monitoring agent was developed that collects measurements from the computational resources of the RPi4 while objects are detected. This data is, subsequently, stored in a dictionary for easy extraction with the use of keys.

The value from the light sensor is also monitored, which connects to the Raspberry Pi via the GrovePi+. This aids in ensuring that the amount of light present stays consistent throughout our investigations. The final stage of the IoT stack involves sending the data that has been processed and collected on the RPi to the cloud platform, ThingsBoard.

All the information collected is stored in a dictionary and includes the following: the current frames per second, the probability score of classifying the chosen object(s), the value from the light sensor, the RAM used, the CPU load, the CPU used and the CPU temperature. The dictionary is then converted into a JSON object which is the format requirement of ThingsBoard. JSON is one of the formats accepted by the Thingsboard.

Figure 11 Placement of Objects, Edge Devices and the Raspberry Pi 4 in the Comparative Analysis Investigation

5 Evaluation

The following models that have been pre-trained on the MS COCO dataset were used: MobileNetV1+SSD (MV1SSD), MobileNetV2+SSD (MV2SSD), MobileNetV2+SSDLite (MV2SSDLite) and Faster R-CNN+InceptionV2. This section discusses the results of the tests that were carried out to evaluate the performance and accuracy of the models

Through a series of evaluations, the probability scores that chosen object(s) are classified by the selected neural network models were compared. This investigation also includes numerous tests that compare the impact that each selected neural network model has on the computational resources of a RPi4. We use all objects that have been discussed previously: bicycles, road vehicles, e.g. cars, trucks and buses, and mobile phones. In this test, we placed the camera and light sensor of the IoT network at ground level, at exactly 80cm away from the object(s) chosen for detection, as presented in Figure 11.

To ensure that our tests were repeatable and reliable, we ran the Python script that feeds real-time video into each neural network model for precisely 5 minutes. The results were then averaged every 30 seconds, producing 10 data points. A tape measure was used to measure the exact 80cm distance between the camera and the

chosen object(s) during detection. We ensured that the temperature of the RPi4 did not exceed 85^{o}C – as this would lead to throttling and a subsequent decrease in performance. We maximised the amount of light present, thereby ensuring that light was not a limiting factor in each model's probability scores for classifying objects. As previously explained, we used white light throughout by using a lamp pointing to each object, as well as the dimmer lights in the room which maximised the light present. By ensuring that there were no other tasks running on the RPi, the computational resource measurements collected reflected the execution of the neural network model. For comparison, resource measurements of the RPi at rest are also included in the results.

5.1 Object Classification Scores

As illustrated in Figure 12 results show that MV1SSD consistently had the lowest probability score for classifying all three objects compared to the other three models. We found that this model's accuracy was between 0.75 and 0.84, and its mode was 0.82. In addition, when using the MV1SSD model for detection, we observed its highest mean probability score was for road vehicles and bicycles tests with 0.82, while the value for mobile phone detected averaged at 0.76. For the MV2SSD model, the probability score ranged between 0.80 and 0.90 for classifying all three objects, with 0.86 the most frequent outcome. The highest mean probability score while utilising the MV2SSD model was generated in the road vehicles test with 0.88, compared to 0.86 in the bicycle test and 0.82 in the mobile phone test. The MV2SSDLite model (for all three objects) resulted in a probability score ranging between 0.80 and 0.91 and a mode of 0.86. While incorporating the MV2SSDLite model, the highest mean probability score was recorded during the classification of road vehicles with 0.90 compared to 0.87 whilst classifying bicycles and 0.84 for classifying mobile phones. In comparison to the other three models, the Faster R-CNN+InceptionV2 model classified the three objects with the highest probability score, with a range between 0.91 and 1, and 0.99 the most common result. In this investigation, the highest mean probability score while using the Faster R-CNN+InceptionV2 model was for the mobile phone test with 0.99, compared to 0.97 for the road vehicles test and 0.95 for the bicycle test. As a result, Faster R-CNN+InceptionV2 achieved an average mean probability score while classifying all three objects that were 21% higher than MV1SSD, 14% higher than MV2SSD and 11% higher than MV2SSDLite. Overall, we have found that the highest mean probability score across all models was achieved during the classification of road vehicles with 0.89, compared to the classification of bicycles with 0.88 and the classification of mobile phones with 0.85.

5.2 Comparing the Performance of Each Model

As shown in Figure 13, the MV2SSDLite model had the best performance across all three investigations, with a mean number of frames per second (FPS) of 1.51, that equates to 0.66 seconds per frame, and a mode of 1.52 across all tests. In comparison, we observed that utilising the MV1SSD model resulted in an average FPS of 1.46 (0.68 seconds per frame) and a most frequent outcome of 1.44 FPS. The MV2SSD model had a mean FPS of 1.21 (0.83 seconds per frame), with 1.17 FPS being the most common result. Undeniably, the worst performing model throughout this investigation was achieved by Faster R-CNN+InceptionV2, with a mean FPS of just 0.17. The consequence of this is that MV2SSDLite attained an 788% increase in FPS over Faster R-CNN+InceptionV2, a 3% increase in FPS over MV1SSD and a 25% increase in FPS over MV2SSD.

5.3 Comparing RAM Used

As shown in Figure 14, the results of this investigation found that utilising the MV2SSD model to classify our chosen object(s) consistently used the most computational resources in relation to the RAM used, with a mean of 670.97MB across the entire investigation. In addition, the Faster R-CNN+InceptionV2 model also used RAM with an average of 645.99MB, while the MV1SSD model resulted in an average RAM usage of 486.79MB. In this study, the least amount of computational resources used in connection to RAM was the MV2SSDLite model, with an average of 456.29MB.

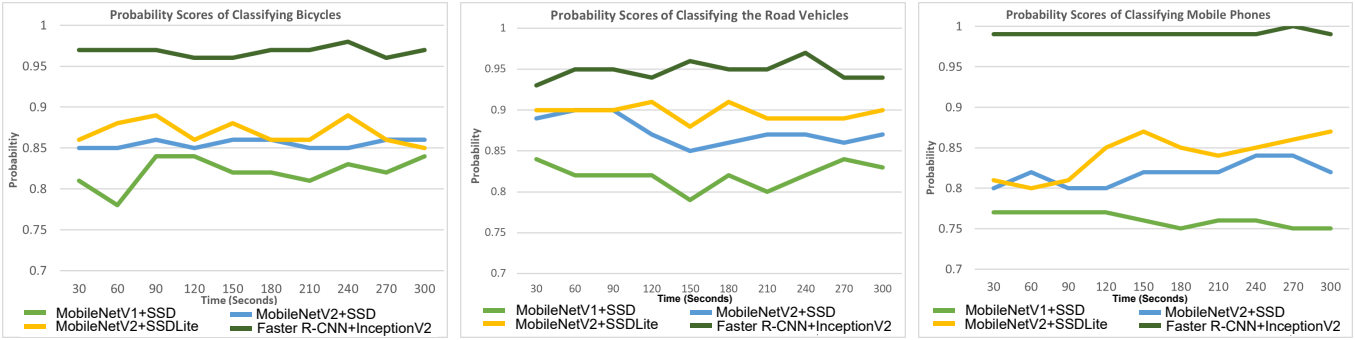

Figure 12 Probability Scores of each Model Classifying Bicycles, Road Vehicles and Mobile Phones

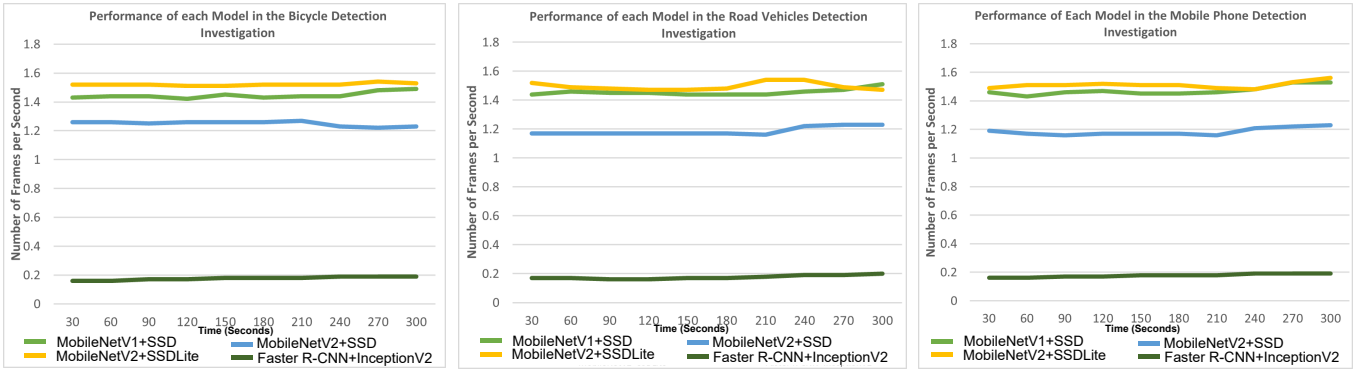

Figure 13 Performance of each Model During the Classification of Bicycles, Road Vehicles and Mobile Phones

As a result, MV2SSD utilised 47% more RAM than MV2SSDLite model, 38% more than MV1SSD model and 4% more than Faster R-CNN+InceptionV2 model.

5.4 Comparing CPU Load

As shown in Figure 15, our results show that the MV1SSD used the least computational resources in relation to the CPU load, with a calculated mean of 1.42 processes during the classification of the three objects. Furthermore, the lowest CPU load across all three investigations was achieved with the MV1SSD model with 2.05 processes compared to the larger values of 2.2 processes, 2.8 processes and 3.72 processes collected from the MV2SDDLite, MV2SSD and Faster R-CNN+InceptionV2 models, respectively. A CPU mean load of 1.48 processes was generated while utilising the MV2SSDLite model, whilst an average of 1.62 processes was recorded with the MV2SSD model and a mean of 2.54 processes was observed while using the Faster R-CNN+InceptionV2 model. Therefore, the Faster R-CNN+InceptionV2 model consumed the most CPU resources. As a result of this investigation, we have found that the MV1SSD model imposed a demand on the CPU load that was 4% less than the MV2SSDLite model, 12% less than the MV2SSD model and 44% less than the Faster R-CNN+InceptionV2 model.

5.5 Comparing the CPU Used by each Model

In this study, as shown in Figure 16, we found that the MV2SSDLite model utilised the least number of computational resources compared to the other three models, with an average of 15.64% for the entire investigation. In comparison, higher averages were observed with the other models in which a mean of 15.67%, 15.92% and 19.35% were recorded for the MV1SSD, MV2SSD and Faster R-CNN+InceptionV2 models, respectively. In addition, we observe that the MV2SSDLite model has a CPU demand of 0.2% less than the MV1SSD model, 2% less than the MV2SSD model and 19% less than the Faster R-CNN+InceptionV2 model. On the other hand, the Faster R-CNN+InceptionV2 model has the greatest CPU usage. This results in the Faster R-CNN+InceptionV2 model using 24% more of CPU than MV2SSDLite, 23% more CPU than MV1SSD and 21% more CPU than MV2SSD. The highest CPU usage achieved within the MV2SSDLite model was 16.5% compared to 17.5% from the MV1SSD model, 18.07% from the MV2SSD model and 22.76% from the Faster R-CNN+InceptionV2 model.

5.6 IoT Deployment & Placement

This test investigates where to place edge devices in an IoT system for optimal object detection. This test provides an insight into the best practices for the

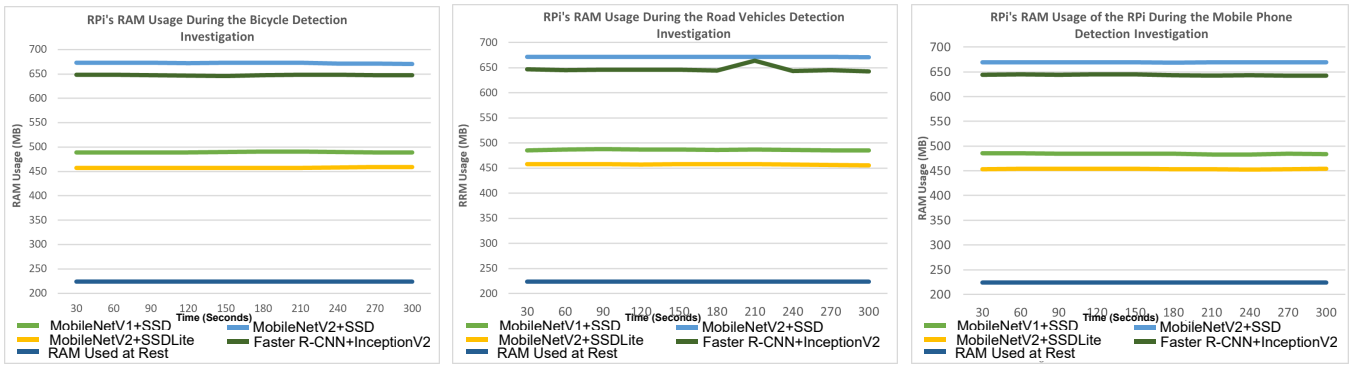

Figure 14 RAM Usage of RPi During the Classification of Bicycles, Road Vehicles and Mobile Phones

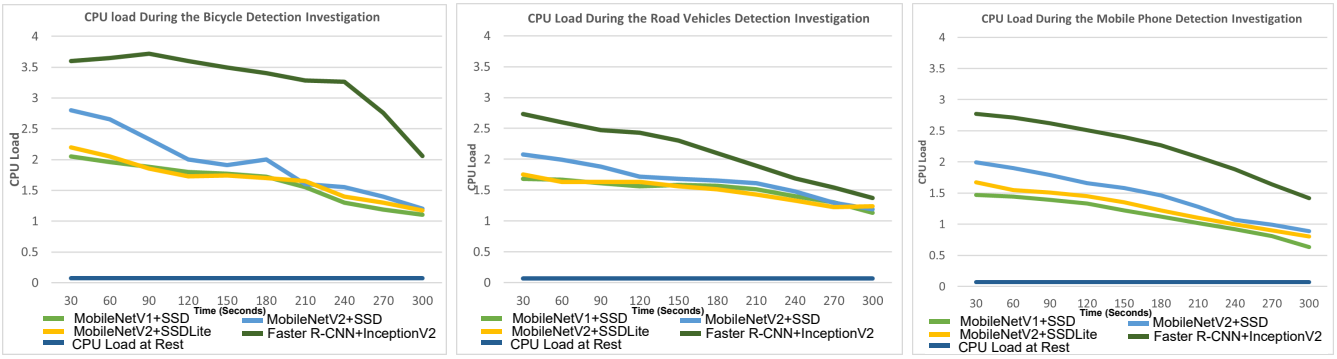

Figure 15 CPU Load of RPi During the Classification of Bicycles, Road Vehicles and Mobile Phones

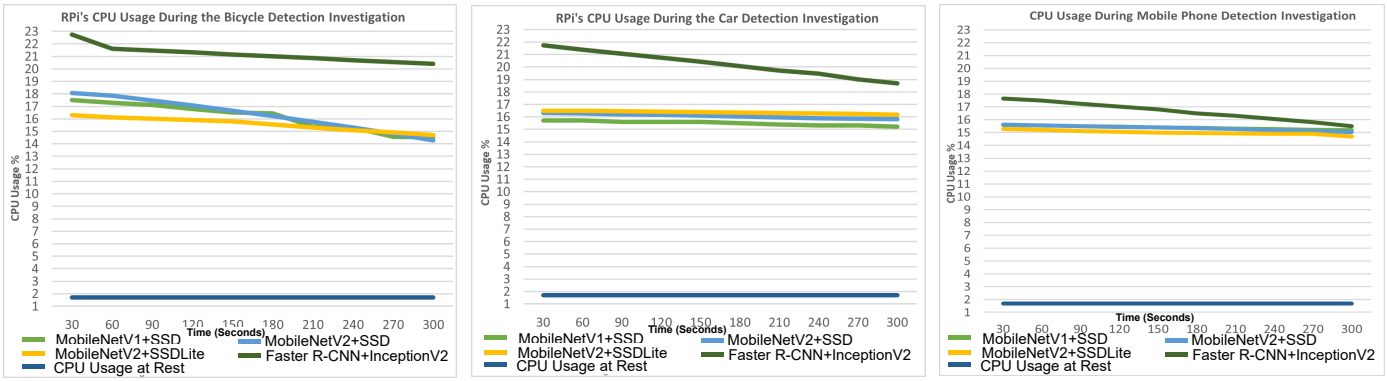

Figure 16 CPU Usage of RPi During the Classification of Bicycles, Road Vehicles and Mobile Phones

deployment of cameras and sensors in an IoT network. We initially deployed our Python script on the RPi to automate the collection of probability scores when the chosen object(s) are classified by each model. We use these probability scores to investigate where to deploy cameras in an IoT network by identifying whether placing cameras at a height leads to an increased classification probability score. The Pi camera used to feed live video into each model only has a resolution of 8 megapixels. Therefore, we found that the neural network models were having difficulty with smaller objects, such as the mobile phone and toy vehicles, when observed from a distance. As a result, we have only used a bicycle for classification in this test.

We primarily placed the camera at ground level at an incrementally increasing distance of 1metre, 1.5metres and 2metres from the bicycle. We then placed the camera at the height of 2.5metres at the same distances away from the bicycle, for comparison. For repeatability, the script that submitted real-time video to each neural network model was executed for 3minutes in each experiment. We then found the mean probability score of each model classifying the bicycle at each distance. The amount of light available was maximised using a white light lamp that ensured that light was not a limiting factor in this investigation. A tape measure is used to ensure that we were placing the camera at exact distances away from the bicycle.

From the results, we can observe a similar trend, i.e. placing the camera at a distance of 1.5metres away from

Table 9 Mean Probability Scores of Each Model Classifying Bicycles When the Camera is Placed at Ground Level

Network Model	1m	1.5m	2m
MobileNetV1+SSD	0.79	0.86	0.59
MobileNetV2+SSD	0.83	0.88	0.64
MobileNetV2+SSDLite	0.89	0.95	0.68
Faster R-CNN+InceptionV2	0.91	0.96	0.71

Table 10 Mean Probability Scores of Each Model Classifying Bicycles When the Camera is Placed at 2.5metres

Network Model	1m	1.5m	2m
MobileNetV1+SSD	0.90	0.96	0.79
MobileNetV2+SSD	0.95	0.98	0.85
MobileNetV2+SSDLite	0.98	0.98	0.88
Faster R-CNN+InceptionV2	0.93	0.98	0.89

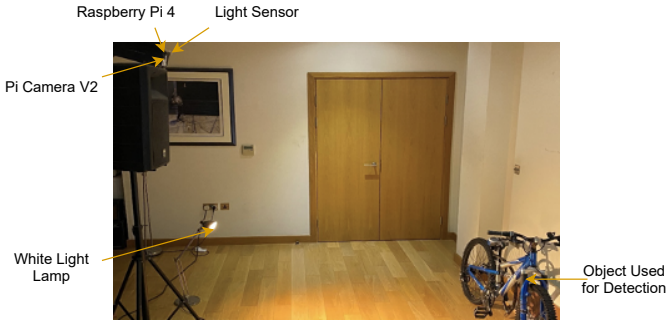

Figure 17 Edge Devices Placed at a Height of 2.5metres in the Deployment Investigation

the bicycle led to the highest probability score with a calculated mean of 0.91 on the ground, compared to 0.98 at a height of 2.5metres. On the contrary, placing the camera at a distance of 2metres away from the bicycle resulted in the worst probability scores with a generated average of 0.66 on the ground and 0.85 at a height of 2.5metres. Placing the camera at a height of 2.5metres consistently resulted in a higher probability score of classifying bicycles than at ground level, at all distances. The overall average of the probability score of classifying the bicycle with the camera at ground level was 0.81, compared to 0.92, with the camera placed at a height of 2.5metres. We also found that the largest increase in probability score between placing the camera at height compared to the ground was at a distance of 2.5metres away from the bicycle at 29%.

6 Discussion and Lessons Learnt

Based on comparative analysis, we find that the Faster R-CNN+InceptionV2 model attains the highest probability score for classifying chosen object(s) in comparison to the other three models. This is achieved by the model's deep neural network architecture that incorporates a region proposal network (RPN) and a Fast R-CNN. This results in higher accuracy in predicting and classifying objects. The MobileNetV2 + SSD (MV2SSD) and MobileNetV2 + SSDLite (MV2SSDLite) models have a higher probability score for classifying chosen object(s) than the MobileNetV1 + SSD (MV1SSD) model. This improvement results from MobileNetV2's architecture that involves linear bottlenecks between layers and shortcut connections. Our results closely align with those of [11], which found that the Faster R-CNN+InceptionV2 model's COCO mAP (mean average precision) measurement was 28, compared to MV1SSD's 21, MV2SSD's 22 and MV2SSDLite's 22. Overall, this investigation was very successful as all models were able to classify the chosen object(s) to a high probability score, with 0.75 the lowest score recorded whilst using the MV1SSD model.

In the performance investigation, the computational complexity of using the RPN and Fast R-CNN in the Faster R-CNN+InceptionV2 model's architecture resulted in an extremely poor performance during the classification of the three objects. Previous work in [47] that used Faster R-CNN+InceptionV2 recorded the number of frames per second (FPS) as 0.67, compared to our research that averaged just 0.17FPS. However, [47] uses a laptop GPU while running their tests, compared to our investigation that uses a Raspberry Pi4 with a Broadcom VideoCore VI. In addition, we observed that even the use of the mobile-friendly models led to a maximum of 1.56FPS during the classification of mobile phones whilst using the MV2SSDLite model. In comparison, the investigation in [46] has found that the number of FPS reached up to 204FPS with the MobileNetV1 model and 220FPS with the MobileNetV2 model. To reach these high measurements, however, their study incorporated a double-buffered approach and GPU acceleration to improve performance. Therefore, the current trend in industry is to deploy these models on servers with large GPUs to improve their performances on the RPi.

It was observed that the demand imposed on the CPU load and CPU usage by the Faster R-CNN+InceptionV2 model's deep neural network architecture was much higher than the mobile-friendly models. Interestingly, however, MV2SSD used the most computational resources in relation to the RAM in the investigation. This is contrary to the study in [10], that found that MobileNetV2 used significantly less Multiply-Adds than MobileNetV1. MV2SSDLite consistently used the least computational resources in relation to all measurements. This is caused by MobileNetV2 pairing with the mobile-friendly variant of SSD, namely SSDLite, that replaces the regular convolutions in SSD with depthwise separable convolutions. The large difference between the computational cost of MV2SSD and MV2SSDLite that we have observed is consistent with [10] that found that the number of Multiply-Adds in MV2SSDLite was significantly less than MV2SSD. In the IoT deployment investigation, we found that placing the camera at a height of 2.5metres consistently resulted in higher probability scores for classifying a bicycle than placing the camera on the ground. Placing the camera at height leads to the camera having a wider view of the object. This results in a higher probability score of classifying the correct object as more features of the object can be considered.

- The Faster R-CNN+InceptionV2 model achieves the highest accuracy and probability score when classifying objects.
- The MV2SSDLite model achieves the highest accuracy and probability score in comparison to other mobile-friendly models.
- Overall, the Faster R-CNN+InceptionV2 model has the highest demand on the computational resources of the RPi, in particular CPU load and CPU usage.

- The MV2SSDLite model has the smallest computational cost.
- Placing the camera at a height of 2.5metres led to higher object classification probability scores, compared to placing the camera at ground level.

7 Conclusion

This work investigates analysis of video streams that have been generated from IoT cameras that are placed at the edge of the network. A key focus is to benchmark leading machine learning/inferencing algorithms that can be executed in proximity to these cameras on Raspberry Pi devices. This is realised by feeding live video through variants of a selection of well-known neural network models that have been pre-trained on the MS COCO image dataset – utilising MobileNetV1+SSD, MobileNetV2+SSD, MobileNetV2+SSDLite and Faster R-CNN+InceptionV2.

We conclude that the MobileNetV2+SSDLite is the best model to use for object detection tasks on the RPi. This model has the highest accuracy and probability score for classifying objects compared to other mobile-friendly models. It also achieved the best performance of 1.51FPS, whilst using the least amount of RAM and CPU of the RPi in comparison to all the models considered.

Our approach for IoT deployment provides a basis for calibrating camera location and its distance from the object of interest. Placing the camera at a height of 2.5metres consistently led to a higher probability score for all models for classifying a bicycle – an increased accuracy of 14% compared to placing the camera at ground level.

Acknowledgement

This work is also partly supported by EPSRC Capital Equipment award for ECR and PETRAS 2 (privacy, ethics, trust, reliability, acceptability, and security) IoT Centre of National Excellence (EP/S035362/1).

References

[1] Shu Chen, Nanxi Chen, Jiayi Tang, and Xu Wang. Cognitive fog for health: A distributed solution for smart city. *International Journal of Computational Science and Engineering*, 22(1):30–38, 2020.

[2] C. Perera, Y. Qin, J.C. Estrella, S. Reiff-Marganiec, and A.V. Vasilakos. Fog computing for sustainable smart cities: A survey. *ACM Computing Surveys*, 50(3), 2017.

[3] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *Communications Surveys Tutorials, IEEE*, 16(1):414 – 454, 2014.

[4] Ce Li, Tan He, Yingheng Wang, Liguo Zhang, Ruili Liu, and Jing Zheng. Pipeline image haze removal system using dark channel prior on cloud processing platform. In *International Journal of Computational Science and Engineering*, volume 22, pages 84–95. Inderscience Publishers, 2020.

[5] Mohammed Amine Benmahdjoub, Abdelkader Mezouar, Larbi Boumediene, and Youcef Saidi. Smart embarked electrical network based on embedded system and monitoring camera. *International Journal of Computational Science and Engineering*, 22(1):15–29, 2020.

[6] Mahmood Hosseini, Constantinos Marios Angelopoulos, Wei Koong Chai, and Stephane Kundig. Crowdcloud: a crowdsourced system for cloud infrastructure. *Cluster Computing*, 2019.

[7] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Dimitrios Georgakopoulos, and Peter Christen. MOSDEN: An internet of things middleware for resource constrained mobile devices. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, pages 1053–1062. IEEE, jan 2014.

[8] Jerry Gao, Jing Zhao, Xiaojun Yin, Sean Chen, and Jesse Huang. Crowd-Based Mobile Sensor Cloud Services - Issues, Challenges and Needs. In *Proceedings - 2017 International Conference on Computational Science and Computational Intelligence, CSCI 2017*, 2018.

[9] Hartwig Adam Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. In *Computer Vision and Pattern Recognition*, 2009.

[10] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[11] Gopikrishna Yadam. Object Detection using TensorFlow and COCO Pre-Trained Models — by Gopikrishna Yadam — Object Detection using TensorFlow and COCO Pre-Trained Models, 2018.

[12] Keyur K Patel, Sunil M Patel, and P G Scholar. Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges. *International Journal of Engineering Science and Computing*, 2016.

[13] Mohamed Abomhara and Geir M. Køien. Cyber security and the internet of things: Vulnerabilities, threats, intruders and attacks. *Journal of Cyber Security and Mobility*, 2015.

[14] Jamilson Dantas, Eltton Araujo, Paulo Maciel, Rubens Matos, and Jean Teixeira. Estimating capacity-oriented availability in cloud systems. *International Journal of Computational Science and Engineering*, 22(4):466–476, 2020.

[15] Charith Perera, Chi Harold Liu, and Srimal Jayawardena. The Emerging Internet of Things Marketplace from an Industrial Perspective: A Survey. *IEEE Transactions on Emerging Topics in Computing*, 3(4):585–598, 2015.

[16] Fei Gu, Zhihua Xia, Jianwei Fei, Chengsheng Yuan, and Qiang Zhang. Face spoof detection using feature map superposition and CNN. *International Journal of Computational Science and Engineering*, 22(2-3):355–363, 2020.

[17] Zhong Qiu Zhao, Peng Zheng, Shou Tao Xu, and Xindong Wu. Object Detection with Deep Learning: A Review, 2019.

[18] Priya Dwivedi. Latest Computer Vision Trends fomr CVPR 2019, 2019.

[19] Alexander Toshev and Christian Szegedy. DeepPose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[20] Ch Sravani, A.Janar Dhana, M.Vara Lakshmi, and Y Kyathi. Gmm Based Vehicle Traffic Analysis On Roads. *International Journal of Engineering Trends and Technology*, 2017.

[21] Neha Patil, Shrikant Ambatkar, and Sandeep Kakde. IoT based smart surveillance security system using raspberry Pi. In *Proceedings of the 2017 IEEE International Conference on Communication and Signal Processing, ICCSP 2017*, 2018.

[22] Nuria Oliver, Barbara Rosario, and Alex Pentland. A Bayesian computer vision system for modeling human interactions. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1999.

[23] Kotaro Hara, Jin Sun, Robert Moore, David Jacobs, and Jon E. Froehlich. Tohme: Detecting curb ramps in Google Street View using crowdsourcing, computer vision, and machine learning. In *UIST 2014 - Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, 2014.

[24] Peter Fairley. Self-driving cars have a bicycle problem [News]. *IEEE Spectrum*, 2017.

[25] Erik Learned-Miller, Gary B. Huang, Aruni RoyChowdhury, Haoxiang Li, and Gang Hua. Labeled faces in the wild: A survey. In *Advances in Face Detection and Facial Image Analysis*. 2016.

[26] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, 2009.

[27] Mark Everingham, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International Journal of Computer Vision*, 2010.

[28] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. 2009.

[29] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.

[30] Ankit Chourasiya and Neha Khare. A Comprehensive Review Of Image Enhancement Techniques. *International Journal of Innovative Research and Growth*, 2019.

[31] J Sanchez-Yanez, Jonathan Cepeda-Negrete, and Raul E Sanchez-Yanez. Experiments on image enhancement for night-vision and surveillance. Technical report, 2015.

[32] Yuen Peng Loh and Chee Seng Chan. Getting to know low-light images with the Exclusively Dark dataset. *Computer Vision and Image Understanding*, 2019.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[36] Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The Power of Sparsity in Convolutional Neural Networks. feb 2017.

[37] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[38] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 2017.

[39] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016.

[40] Hongxiang Fan, Shuanglong Liu, Martin Ferianc, Ho Cheung Ng, Zhiqiang Que, Shen Liu, Xinyu Niu, and Wayne Luk. A Real-Time Object Detection Accelerator with Compressed SSDLite on FPGA. In *Proceedings - 2018 International Conference on Field-Programmable Technology, FPT 2018*, 2018.

[41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[42] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014.

[43] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[44] R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms — by Rohith Gandhi — Towards Data Science, 2018.

[45] Tensorflow.org. Performance measurement — TensorFlow Lite, 2020.

[46] Matthijs Hollemans. MobileNet version 2, 2018.

[47] Madhawa Vidanapathirana. Real-time Human Detection in Computer Vision — Part 2 — by Madhawa Vidanapathirana — Medium, 2018.

[48] Rob Zwetsloot. Raspberry Pi 4 specs and benchmarks. Technical report, Raspberrypi.org, 2019.

[49] Aamir Nizam Ansari, Mohamed Sedky, Neelam Sharma, and Anurag Tyagi. An Internet of things approach for motion detection using Raspberry Pi. In *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things, ICIT 2015*, 2015.

[50] Matt Richardson. Meter Maid Monitor: parking protection with Pi - Raspberry Pi, 2016.

[51] Guto Leoni Santos, Demis Gomes, Judith Kelner, Djamel Sadok, Francisco Airton Silva, Patricia Takako Endo, and Theo Lynn. The internet of things for healthcare: Optimising e-health system availability in the fog and cloud. In *International Journal of Computational Science and Engineering*, volume 21, pages 615–628. Inderscience Publishers, 2020.

[52] B. Alturki, S. Reiff-Marganiec, and C. Perera. A hybrid approach for data analytics for Internet of Things. In *ACM International Conference Proceeding Series*, volume Part F1327, 2017.

[53] Badraddin Alturki, Stephan Reiff-Marganiec, Charith Perera, and Suparna De. Exploring the Effectiveness of Service Decomposition in Fog Computing Architecture for the Internet of Things. *IEEE Transactions on Sustainable Computing*, 2019.

[54] Jay Ramey. How does pedestrian and cyclist detection work? Autoweek explains, 2017.

[55] Adisorn Lertsinsrubtavee, Anwaar Ali, Carlos Molina-Jimenez, Arjuna Sathiaseelan, and Jon Crowcroft. Picasso: A lightweight edge computing platform. In *Proceedings of the 2017 IEEE 6th International Conference on Cloud Networking, CloudNet 2017*, 2017.

[56] Shahriar Shovon. Raspberry Pi Temperature Monitor, 2018.

[57] Peter Raynham and Torunn Saksvikrønning. White light and facial recognition. *Lighting Journal (Rugby, England)*, 2003.