Real-Time Intelligent Audio and Object Detection for Robotics Systems

By

Raghav Atal
Steve Barry
Venku Buragadda
Daniel Dobrzynski
J. Nick Gallagher
Marcus Himelhoch
Brian Kremzier
Kai Li
Abigail L. Matthews
Christopher B. Olen
Sasha Roberts
Audrey Salerno
Rushmi Ranjan Samal
Scott Shepard
Kevin Stutenberg
L. Victoria Waring

Advisor: Yuri Balasanov, Ph.D.

A Capstone Project Submitted to the University of Chicago in partial fulfillment of the requirements for the degree of Master of Science in Analytics

Chicago, Illinois

June 2020

The Capstone Project Committee for

Raghav Atal
Steve Barry
Venku Buragadda
Daniel Dobrzynski
J. Nick Gallagher
Marcus Himelhoch
Brian Kremzier
Kai Li
Abigail L. Matthews
Christopher B. Olen
Sasha Roberts
Audrey Salerno
Rush Samal
Scott Shepard
Kevin Stutenberg
L. Victoria Waring

certifies that this is the approved version of the following capstone project report:

Real-Time Intelligent Audio and Object Detection for Robotics Systems.

_____
Project Title

_____
Project Supervisor

_____
Program Director

Abstract

We present three separate object detection and audio classification schemes for robotics applications using the Raspberry Pi 4. Each application builds on advancements within the fields of object detection and audio classification to-date and takes advantage of the increased computational power of the Raspberry Pi 4. We ultimately test each application against a predetermined set of robotics tasks in the form of a scavenger hunt. Our findings suggest: the Raspberry Pi 4 can effectively manage five threads in parallel but it may perform optimally using only three; deep learning methods out-perform probabilistic approaches to audio classification; efficiency gains can be obtained from using portable USB accelerators and edge computing-optimized models; object detection models for robotics tasks must be trained on a wide variety of data given their high degree of bias toward their training data.

Keywords: Deep Learning, Object Detection, Audio Detection, Robotics, Parallel Processing, Multithreading, Transfer Learning

Executive Summary

We present three separate object detection and audio classification schemes for robotics applications using the Raspberry Pi 4. Each application builds on advancements within the fields of object detection and audio classification to-date and takes advantage of the increased computational power of the Raspberry Pi 4. We ultimately test each application against a predetermined set of robotics tasks in the form of a scavenger hunt. Our findings suggest: the Raspberry Pi 4 can effectively manage five threads in parallel but it may perform optimally using only three; deep learning methods out-perform probabilistic approaches to audio classification; efficiency gains can be obtained from using portable USB accelerators and edge computing-optimized models; object detection models for robotics tasks must be trained on a wide variety of data given their high degree of bias toward their training data.

Table of Contents

Introduction

The Raspberry Pi is one of the latest developments in the evolution of mobile computing technologies over the past few decades. Spurred on by ARM, a British hardware designer whose products were featured in approximately 95% of mobile phones by 2005, the first Raspberry Pi was made available to the public in 2012 (Dennis, 2015).

Concurrently, the growing availability of compute power coupled with deep neural networks has brought on the rise of deep learning. And with this increase in model complexity and computational power, deep learning practitioners have been able to address increasingly complex problems. Deep learning has proved consequential in robotics applications (Goodfellow, Bengio, & Courville, 2016), and indeed, the deep learning revolution has coincided with explosive growth within the robotics industry (Mordor Intelligence, 2019).

This research seeks to explore the capabilities of deep learning algorithms for automation tasks using relatively inexpensive mobile computing equipment such as the Raspberry Pi. Specifically, we hope to determine the most effective deep learning applications for object detection and audio detection on the Raspberry Pi and its compatible hardware interfaces.[1] In doing so, we hope to contribute to the growing numbers of commercial products available for home and light-industry automation (Dennis ,2015)

The fields of object detection, audio recognition, and robotics are broad, but much of this research is done on state-of-the-art platforms. Our research focuses on applying recent object detection and audio classification advancements to the more restrictive environment of the

[1] For this project, we will be relying on Dexter Industries' GoPiGo3 board, which communicates with the Raspberry Pi 4 over an SPI interface. More information about the GoPiGo robot can be found in the Methodology section.

Raspberry Pi. In doing so, we hope to train our robot to intelligently classify 12 classes of household objects, four colors of traffic cones, and 10 classes of audio samples.

We will begin with a brief review of the latest advancements in image recognition and audio detection, together with a description of the latest available technology from Raspberry Pi. We hope to take advantage of insights from these domains to propose the most effective, lightweight models for our Raspberry Pi. We will then detail the discrete robotics tasks toward which we will apply various image recognition, audio detection, and multithreading methodologies. Said tasks will ultimately inform the steps we take to train our models. Finally, we will review the performance of our various methodologies, provide the appropriate conclusions, and suggest directions for future research.

Background and Literature Review

Advancements in Object Detection

The past two decades have seen a revolution in computer vision, largely driven by the evolution of the Convolutional Neural Network (CNN). Beginning with early architectures such as Yann LeCun's *LeNet-5* in 1998, researchers in field of computer vision have taken advantage of increasing quantities of labeled image data, growing computational power, and deeper network architectures to vastly reduce the image classification error rates in competitions such as the ILSVRC ImageNet Challenge - a useful benchmark for measuring progress in computer vision. More recently, the winner of the 2015 ImageNet challenge was the Residual Network (or *ResNet*): a 152-layer deep CNN whose major contribution was the *skip connection*, which uses residual learning to speed up training considerably (Gerón 2017).

Object detection takes image recognition a step further by introducing the challenge of localization: Given the existence of one or more target objects within an image, how to simultaneously identify the class and location of one or more objects in an image? Early approaches use a sliding-window approach where a classifier is passed over distinct regions of an image, accepting only the inferences with the highest confidence scores to determine image location (Dalal & Triggs 2006; Felzenszwalb et al., 2010). Nonetheless, such approaches are slow on inference, as they require multiple, separate inferences prior to object detection.

You Only Look Once (YOLO)

First developed in 2015, YOLOv1 reframes object detection as a single regression problem by simultaneously predicting the classes and bounding boxes of target images (Redmon et al., 2015). In doing so, it substantially decreases inference time compared to many of its predecessors while maintaining a relatively high degree of accuracy.

During inference, YOLO divides a given image into an *S* x *S* grid and then predicts *B* bounding boxes with center coordinates (*x*, *y*) located in each of the *S* x *S* grid cells, along with each bounding boxes' height *h* and width *w*. Also, for each grid cell, it provides a confidence score for each of the *C* possible object classes. Thus, for each inference, YOLOv1 outputs an *S* x *S* x (*B* * 5 + *C*)-dimensional tensor. Ultimately, inferences are made by calculating the joint probability of a bounding box containing an object and the grid cell in which the bounding box is centered containing an object of a certain class and only retaining predictions above a certain threshold (Redmon et al., 2015).

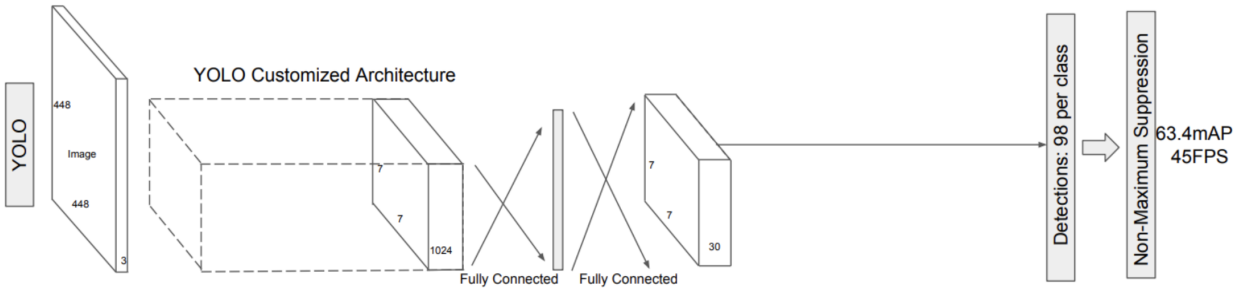


Figure 1: Model structure of YOLOv1 (Liu et al., 2016)

In the likely event that multiple bounding boxes detect the same object, YOLOv1 uses a technique called Non-Maximal Suppression (NMS), which removes certain duplicative bounding boxes depending on a given bounding box's probability and the extent to which it overlaps with other bounding boxes (Redmon et al., 2015).

Version three of YOLO (YOLOv3) realizes additional performance and efficiency gains over previous versions. Most notably, this model architecture provides an improved backbone classifier in DarkNet-53 and includes multi-scale predictions similar to SSD (Redmon et al., 2018). Darknet-53 is mainly composed of three-by-three and one-by-one filters with skip connections similar to the residual units in ResNet. Darknet-53 also has fewer *billion floating*

point operations (BFLOP) than ResNet-152, but achieves the same classification accuracy at half the computational time. Next, YOLOv3 does not use the softmax activation function typically used in object detectors. Instead, it uses individual logistic classifiers to determine the likelihood that an object belongs to a given label versus all other labels. This innovation leads to significant speed improvements. The updated version performs inferences three times faster than its predecessor with comparable accuracy (Redmon et al., 2018). YOLOv3's Feature Pyramid Network (FPN) also makes it better at detecting small objects vis-a-vis MobileNetV1 SSD. FPN consists of a bottom-up and a top-down pathway. The bottom-up pathway is the usual feed-forward convolutional network for feature extraction, where increasingly granular features are extracted at the deeper layers. The deeper the layer the greater semantic meaning captured by that layer (Hui, 2018).

The Single Shot Multibox Detector (SSD)

The single shot multibox detector (SSD) is an additional approach to object detection proposed by Liu et al. (2016). Similar to YOLOv1, SSD can detect multiple objects in a single image by separating the input space into a set of default bounding boxes of various sizes and aspect ratios. One major contribution of SSD is that it predicts object classes and offsets in bounding box locations using feature maps of varying scales from different layers throughout the network. Thus, all k bounding boxes corresponding to the n x m locations for each of f total feature maps, we output confidence scores for all classes c and 4 and bounding box offset predictions. The result of these contributions is an increase in mean accuracy precision (mAP) of from YOLOv1's 63.4% to a 74.3% and an increase in speed from YOLOv1's 45 frames per

second (FPS) to 59 FPS when using VGG-16 as SSD's base model. Furthermore, compared to other single stage methods, SSD has much better accuracy even with a smaller input image size.

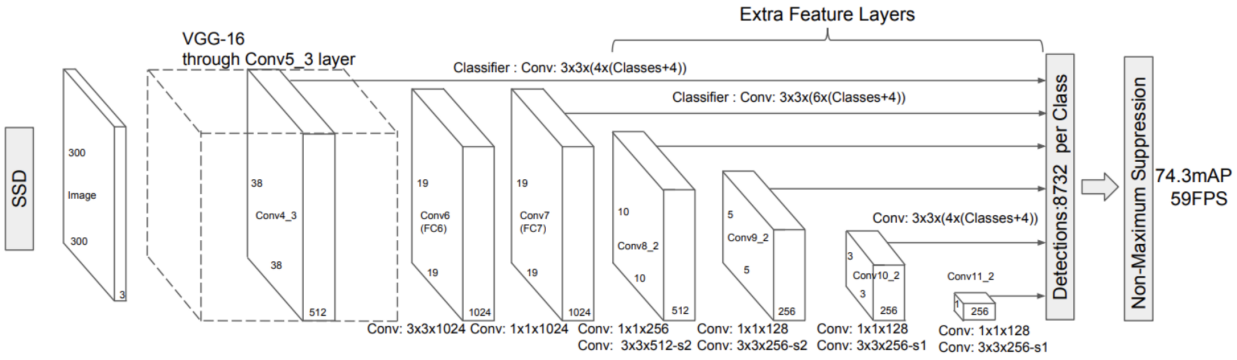


Figure 2: Model structure of SSD (Liu et al., 2016)

SSDs can be customized to use different base image classification models for feature extraction. MobileNet SSDs are specifically designed for use in mobile and embedded vision applications, by using either MobileNetV1 or MobileNetV2 as the base image classification model.

MobileNetV1 (Howard, et al., 2017) improves efficiency by offering an alternative to expensive, traditional convolutions that have to both filter and combine inputs into a new set of outputs all in one step. The so-called *depthwise separable convolutions* factorize the traditional convolution layer into two component layers: a depthwise convolution to apply a single filter per input channel and a 1×1 pointwise convolution to create a linear combination of the depthwise convolution output. The factorization has the effect of drastically reducing computation time and model size by placing 95% all of computations and 75% of parameters in the dense pointwise convolutions. The resulting MobileNetV1 model netted a 70.6% top-1 accuracy on ImageNet, with seven times fewer parameters and only a 1.1% loss in accuracy compared to a similar model using traditional convolutional layers.

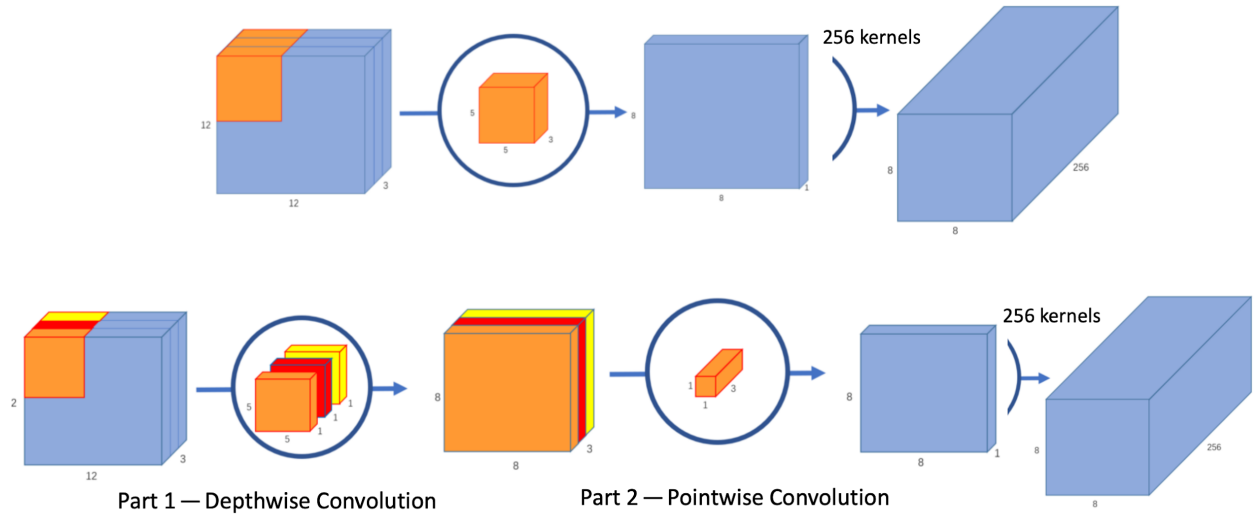


Figure 3: Depthwise separable convolution. Top: Standard Convolution. Bottom: Factorized Convolution (Wang, 2018)

Howard, et al. (2019) further improved upon MobleNetV1 with the scalable MobileNetV2. They introduce a novel layer module: an *inverted residual with linear bottleneck*. This adds linear bottlenecks between pairs of depthwise separable convolutions and skipped connections between the bottlenecks. The bottlenecks aim to encode the intermediate inputs and outputs, while the intermediate depthwise separable convolutions capture image features.

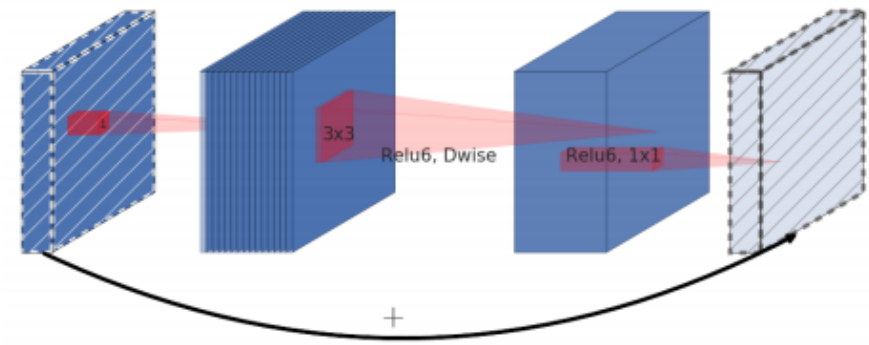


Figure 4: Inverted residual with linear bottleneck. Thickness in the figure denotes a relative number of channels. The hashed lines indicate bottlenecks, which are connected together (Howard, et al., 2019).

MobileNetV2 yields a top-1 classification accuracy of 72% on ImageNet data, outperforming MobileNetV1 by +1.4% with a 48% increase in efficiency. MobileNetV2 is further scalable with additional parameters to increase network width and resolution. Scalable networks are discussed in more detail in the next section.

Advancements Audio Detection

The past few decades have seen the advent of various probabilistic and machine learning approaches to audio detection problems. More recently, with the evolution of CNNs, approaches that have been proposed for image recognition have also achieved success within the audio detection domain.

Digital Representations of Audio Signals

When analog audio signals are captured as digital audio signals, they are represented as time series of sampled amplitudes with a known sampling rate. In order to properly analyze and recognize audio signals using learning algorithms, we need a way to also understand the frequency composition. A common method of representing changes in both amplitude and frequency of an audio signal is through a three-dimensional estimate of the *power spectral density* (PSD). Boashash (2016) describes how the PSD distributes the amplitude or power into the frequency components that make up the audio signal. The PSD estimate is generated from the squared magnitude of the windowed *Fast Fourier Transform* (FFT) of the audio signal. The FFT is a discrete Fourier transform of the audio signal used to convert the amplitude from the time domain into the frequency domain. A windowed FFT is also known as a *Short-Term Fourier Transform* (STFT), where the windowing allows the spectrum to be localized in time but results in some blurring of the spectrum across frequencies due to uncertainty relationships in the

frequency resolution. Therefore, the window length of the STFT needs to be tuned to optimize the trade-off between time resolution and frequency resolution.

Non-Linear Transformations and Condensed Representations of Audio Signals

For the purpose of audio recognition, it is common to transform the initial linear frequency scale to better emphasize the important frequencies to human hearing. A common scaling method is known as the Mel scale, which is linear below 1 kHz and logarithmic above 1 kHz. This scale is designed to better represent the human-perception of audio signals, where changes in frequency are represented in equal-pitch increments (Boashash, 2016).

Mel-frequency cepstral coefficients (MFCC) are condensed representations of the original time domain of an audio signal and have been popular in the speech recognition domain. MFCCs are obtained by mapping the PSD estimate of the audio signal onto the *mel scale*, taking the log of the resulting powers, and applying a discrete cosine transformation to transform back to the time domain (Gales & Young, 2008). Comparing various encoding schemes when applied to speech, Davis and Mermelstein (1980) find that MFCCs are significantly better than comparable methods at suppressing insignificant spectral variation at higher frequencies. Furthermore, MFCCs are particularly adept at providing compact representations of the original time domain, often capturing most of the relevant information in as few as six cepstral coefficients.

Image Representations of Audio Signals

Audio signals can also be represented as spectrograms images. A spectrogram is a plot of the power spectral density of the audio signal, with time on the *x*-axis, frequency on the *y*-axis, and the amplitude of the signal at a given time and frequency represented by the color intensity (with light colors indicating the higher amplitudes). Spectrograms can be further transformed into *mel spectrograms* by converting the linear frequency scale to a Mel scale.

Using Image Recognition Models for Audio Detection

The ability to represent audio signals as images allows us to apply computer vision approaches to the problem of audio detection. In other words, CNNs designed for image recognition can be used to classify spectrograms. As with many computer vision tasks, the process of training such a model can be aided by starting with a pre-trained CNN and applying transfer learning. We will review and consider a variety of families of scalable CNNs that are pre-trained on large, multi-class, image datasets such as ImageNet.[2]

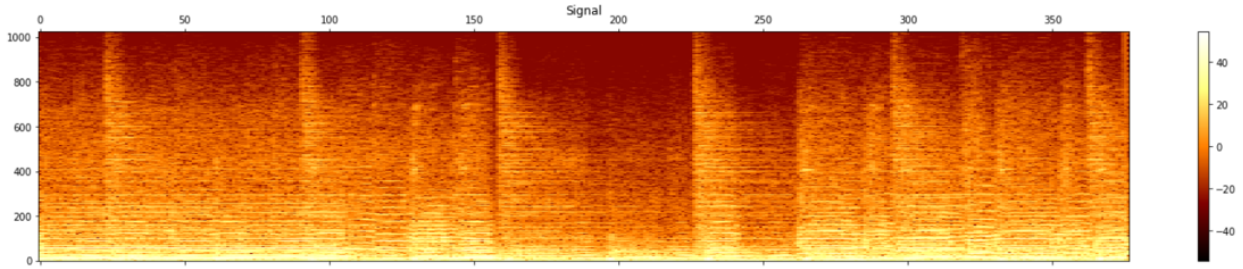


Figure 5: Spectrogram image

Scalable Convolutional Neural Networks

A common way to increase accuracy of a CNN is to increase the depth, width, or input resolution (or input shape) of the network - a technique known as scaling. Typically, only one dimension is scaled, and the most common dimension to scale up is the depth or number of layers. As a network is scaled up in size, its parameter space and resource needs grow along with accuracy, so scalable CNN families yield multiple sizes to best match a model to the data and resource requirements of the problem.

Early attempts to scale all three dimensions were unsuccessful, requiring tedious amounts of manual tuning while still yielding sub-optimal accuracy and efficiency (Tan & Le, 2019).

[2] Additional details on ImageNet can be found in the *Training Data* subsection of *Methodology.*

Furthermore, scaling improves accuracy at the cost of efficiency. As the parameter space of the model grows, so does the training and inference time, in addition to the amount of memory required.

In 2019, researchers at Google (Tan & Le, 2019) identified a new method of uniformly scaling up a CNN's network dimensions (width, depth, and input resolution) by a fixed set of scaling coefficients - a technique known as *compound scaling* - resulting in better accuracy and efficiency. Figure 5 below illustrates the differences between single-dimension scaling and compound scaling. Compared to traditional scaling methods, compound scaling results in a 0.7% increase in accuracy for ResNet and a 1.4% increase in accuracy for MobileNet. We discuss these two networks in further detail below.

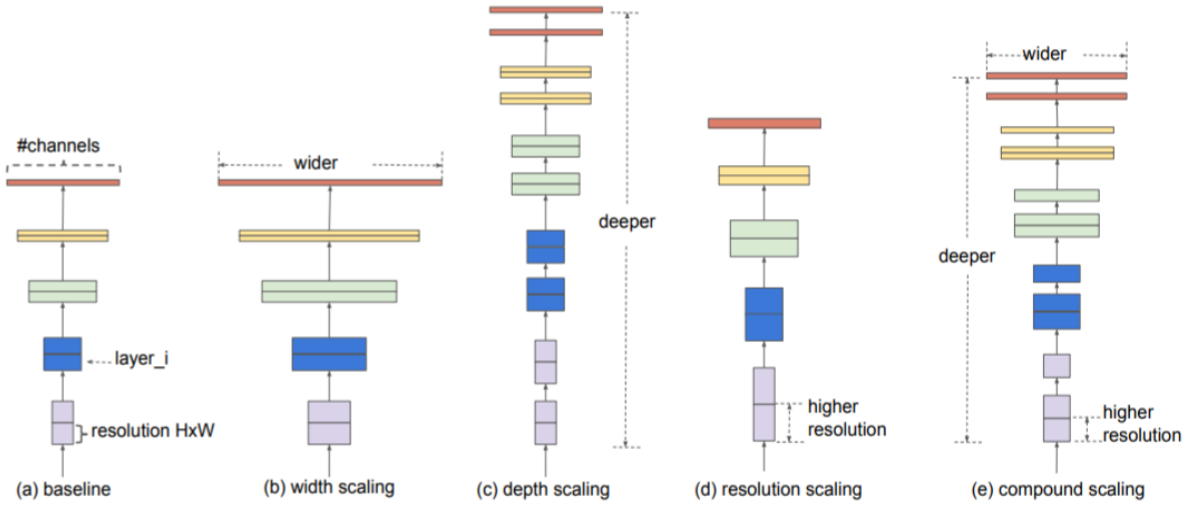


Figure 6: Model Scaling: (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio (Tan & Le, 2019).

ResNet

ResNet is a CNN developed to address the *degradation problem*. Degradation occurs when, as the complexity of the model increases, the training error increases for reasons other than overfitting. Strategies such as Xavier initialization and batch normalization can prevent

overfitting but additional learning difficulties may persist, resulting in worse performance than shallower networks (Gu et al., 2017). ResNet introduces the *residual block,* which uses shortcut connections, where the identity shortcut is added to the learned residual map to obtain the desired mapping. Research by Viet, Wilber, and Belongie (2016) suggests that one key characteristic that enables the training of deep networks is that residual networks avoid the vanishing gradient problem by introducing short paths that can carry gradient throughout the extent of the network. The network is then able to gradually restore the skipped layers as it learns the feature space. ResNet gained popularity when it won ILSVRC-2015. A 152-layer CNN trained using skip connections achieved 3.57% error on the ImageNet test set (He et al., 2016). ResNet remains popular today and its application has expanded from vision tasks to audio detection via the use of spectrograms (Hershey, 2017).

EfficientNets & EfficientNets-Lite

Embedded systems, such as microcomputers in robots and mobile devices do not always have the processing power to effectively use large, high-accuracy CNN architectures, such as ResNet. These low-power devices require specially optimized architectures to provide both accurate and fast inference.

Building on the efficiency gains of architectures such as MobileNetV2, Tan & Le (2019) use compound scaling to build a new family of eight scalable CNNs known as EfficientNet. Trained on the ILSVRC-2012 dataset, the baseline network, EfficientNet-B0, was developed using a multi-objective neural architecture search to optimize for accuracy and efficiency. Tan & Le's resulting architecture utilizes "mobile inverted bottleneck convolution" (MBConv) layers, which resulted in an architecture similar to MobleNetV2 but with compound scaling. They found that compared to MobileNetV2, EfficientNet-B0 yields a +5.3% increase in top-1 accuracy. The

largest version, EfficientNet-B7, had a top-1 accuracy of 84.4%, matching the state-of-the-art GPipe network but with 8.4-times fewer parameters and a 6.1-fold increase in speed.

In May of 2020, TensorFlow (Liu, 2020) released a family of five edge-device-optimized EfficientNets, named EfficientNet-Lite. EfficientNet-Lite tailored the EfficientNet network to allow for integer quantization, TPU compatibility, and TensorFlow Lite conversion of the model with minimal accuracy loss. Integer (*Int-8*) quantization significantly reduces the size and latency of the model, while eliminating floating-point incompatibility on many edge devices, and sacrificing only 1% of top-1 accuracy on ImageNet compared to *float-32* models (Liu, 2020). Comparing top-1 accuracy between EfficientNet-Lite and the original EfficientNet, integer-quantized EfficientNet-Lite4 (the largest) has a top-1 accuracy 80.2%, which is 0.1% less than EfficientNet-B2 (Liu, 2020; Tan & Lee, 2019).

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Figure 7: EfficientNet-B0 Architecture: Resolution, number of channels, and number of layers are compound scaled for larger versions of EfficientNet. Number of layers indicates the number of sequential layers of a given operator in a stage (Tan & Le, 2019).

Probabilistic Approaches: Hidden Markov Models (HMM)

Hidden Markov Models (HMMs) are doubly stochastic processes based on an underlying state sequence assumed to be Markovian (Fox, 2009). HMMs have been used in many if not all speech recognition systems since the 1970s. Beginning as discrete-word, speaker-dependent

systems, HMMs have gradually evolved to cover continuous-word, speaker-independent systems that can recognize large vocabularies (Gales & Young, 2008).

From Fox (2009), we begin by defining the hidden states: Let z_t denote the state of a Markov chain at time t and let π_j denote the state-specific transition distribution for state *j*. Then, for all $t > 1$:

$$z_t | z_{t-1} \sim \pi_{z_{t-1}}$$

We then define the initial transition distribution to determine the probability that the a given state is the initial state of a sequence:

$$z_1 \sim \pi^0$$

Finally, we define the distribution of the observation y_t conditioned on the state z_t as:

$$y_t | z_t \sim F(\theta_{zt})$$

Where $F(\bullet)$ is an indexed family of distributions for each hidden state and θ_i are the emissions parameters for state i. The resulting joint density for *n* observations is then given by:

$$p(z_{1:n}, y_{1:n}) = \pi^0(z_1) p(y_1 | z_1) \prod_{t=2}^{n} p(z_t | z_{t-1}) p(y_t | z_t)$$

To fit an HMM, we use the expectation-maximization algorithm or some sampling scheme such as Markov Chain Monte Carlo. In the former case, our objective, in simplistic terms, is to maximum the joint probability of the hidden state sequence and the data given the parameters.

In the end, some of the advantages of an HMM for audio classification over a deep learning approach include their ability to train on and performance inference on sequences of varying length, their interpretability, and their relative efficiency on inference.

Methodology

The diversity of image recognition and audio detection applications in the machine learning literature coupled with the complexity of real-time robotics tasks leads us to propose three separate approaches to a pre-specified robotics task. We begin by describing said task.

Scavenger Hunt Challenge

The robot is tasked with performing the Scavenger Hunt Challenge. Three separate trials of the same challenge will be performed. Each trial is set within a flat 10-square-foot space indoors, henceforth referred to as the robot's *environment*. Four cones of different colors are arranged within the *environment*, and a different object is hidden behind the green, purple, and yellow cones. The orange cone is designated as *home base*. A speaker is set up near the *environment* to broadcast sounds that the robot will identify during the challenge.

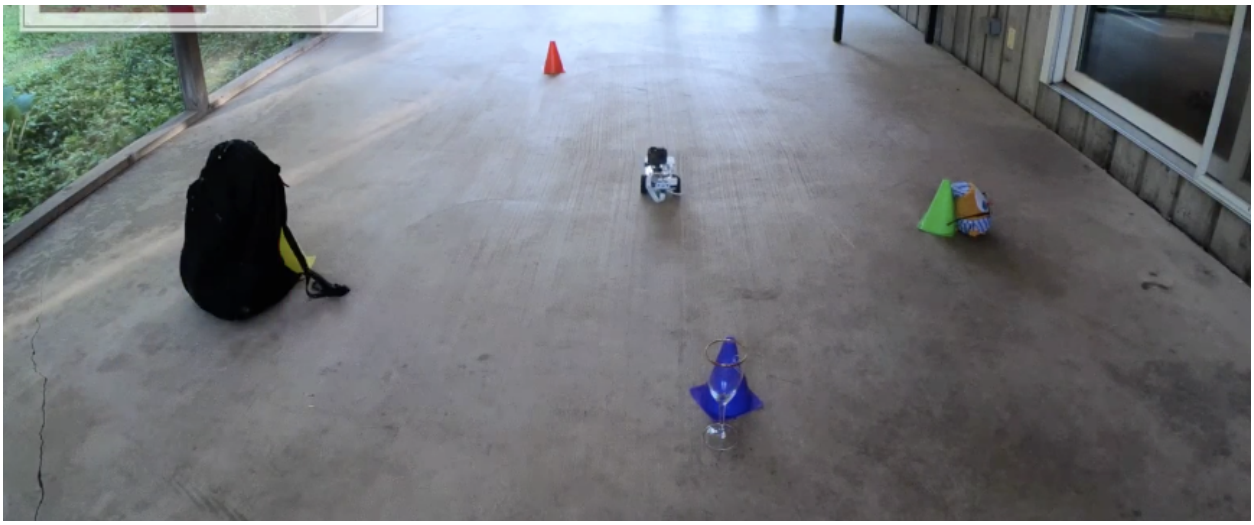


Figure 8: Scavenger Hunt Challenge Robotics Environment

Initially positioned within a one-foot radius of the *home base* cone, the robot is tasked with navigating to and identifying household objects placed behind the green, purple, and yellow cones. One small, one medium, and one large object will be randomly selected from the lists in Table X and placed behind each of these three cones prior to the start of the challenge. The robot

will need to first identify the object behind the green cone, then identify the object behind the purple cone, and finally identify the object behind the yellow cone. When the object behind each respective cone is identified, the class of the object, the color of the cone it was found behind, and the time of identification must be logged. After each successive trip to the green, purple, and yellow cones, the robot must return to and orbit the orange, *home base,* cone before setting out toward the next cone in the sequence. Each of the three trials will begin by broadcasting a pitch at a predetermined frequency and will end when the robot returns to the *home base* cone for the final time.

Small Objects			
Tennis Ball	Dice	Eyeglasses	Banana
Medium Objects			
Wine Bottle	Wine Glass	Boot	Book
Large Objects			
Suitcase	Backpack	Tennis Racket	Bicycle Helmet

Table 1. List of 12 possible classes of objects to be hidden behind the cones during the challenge. The objects are further divided into three parent classes: small, medium, and large objects.

Concurrently, four sounds from the *UrbanSound8K* dataset[3] (Salamon, Jacoby, & Bello 2014) will be played from a nearby speaker at 8 kHz at random times throughout the trial. When each sound is played, the robot will be required to identify the sound class and log the time the

[3] More information about the *UrbanSound8K* dataset can be found in the *Data* subsection.

classification was made. The four sounds will be selected at random from four of the following 10 sound classes:

***UrbanSound8K* Classes**	
Gun Shot	Drill
Dog Bark	Jackhammer
Car Horn	Children Playing
Street Music	Engine Idling
Air Conditioner	Siren

Table 2: *UrbanSound8K* Classes

Robotics Hardware and Software

Each trial has the following hardware specifications:

	Trial I	**Trial II**	**Trial III**
CPU:	Raspberry Pi 4 Model B, 4GB of RAM		
Robotics Platform:	Dexter GoPiGo3 Base Kit		
Hard Disk:	MicroSD Card, 32 or 64 GB		
Distance Sensor:	1 x Dexter Distance Sensor		
Camera:	1 x Raspberry Pi Camera		
Batteries:	8 x AA Batteries		
Servo:	1 x Dexter Servo Kit		
Microphone	1 x SiZheng Omnidirectional USB Microphone	1 x TKGOU Gooseneck Omnidirectional Condenser USB Microphone	1 x TKGOU Gooseneck Omnidirectional Condenser USB Microphone
Accelerator (Optional):	Intel Neural Compute Stick	Google Coral USB Edge TPU	None

Table 3: Robotics Hardware

Each trial has the following software specifications:

	Trial I	**Trial II**	**Trial II**
Operating System:	Rasbpian Buster, 64-Bit		
Python:	Version 3		
Deep Learning Framework:	TensorFlow 2+	TensorFlow 2+ (training) TensorFlow Lite (inference)	PyTorch (audio); TensorFlow 1.15 (image)

Table 4: Robotics Software

Training Data

We use transfer learning to obtain pre-trained models for the trials in which we rely on CNNs for object detection and/or audio detection tasks. The YOLOv3 and MobileNetV2 SSD object detection models were pre-trained on the Common Objects in Context (COCO) dataset (Lin et. al 2015). This is an object detection, segmentation, and captioning dataset of 1.5 million training instances with five captions per image and bounding boxes around the objects in each image, allowing users to train on objects in a variety of contexts. ResNet and EfficientNet-Lite image recognition models were pre-trained on ILSVRC-2012.

For the object detection tasks, we then further train our models to recognize the 13 object classes and four colored cones of the robotics challenge. For the 12 object classes, additional bounded images were obtained from the Google Open Images Dataset. Images of the colored cones were manually captured and bounding boxes were subsequently added.

For the audio detection tasks, we use data from the *UrbanSound8K* dataset (Salamon, Jacoby, & Bello 2014). This dataset consists of 8,732 sound recording excerpts taken from 1,302 original recordings organized into 10 classes (see Table 2). The recording excerpts range in

length between one and four seconds, and most - but not all - recordings were recorded in stereo at 44.1 kHz. Salamon, Jacoby, & Bello provided 10 stratified folds cross-fold validation. The predefined folds ensure that all of the excerpts from a given original sound file are contained in the same fold, thus avoiding any data leakage between training and validation.

Figure 8 contains a sample of mel spectrogram images for each of the 10 classes of *UrbanSound8K*. We can see significant diversity of spectrograms within each class, as well as similarity of spectrograms across the classes. The commonalities between certain classes of sounds, such as air conditioner and drilling, proved challenging during each of the trials.

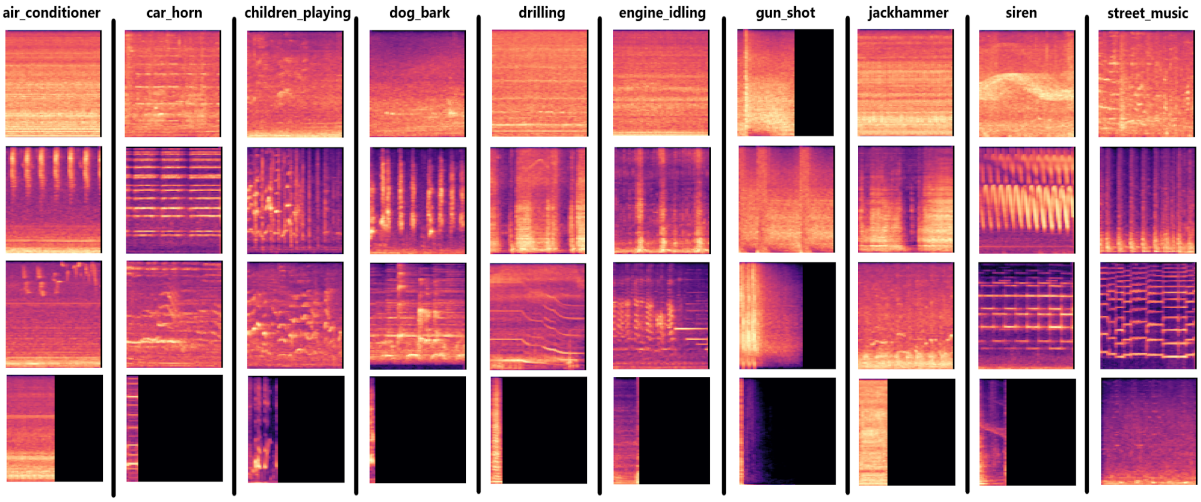


Figure 9: Mel spectrogram images of *UrbanSound8K* classes

During the robotics challenge, the audio signals obtained by the robot's microphone in its environment are constantly being augmented by the sounds of the robot - the turning of the motors and wheels, the buzzing of the servos, and the humming of the CPU. Given this constant noise, additional data augmentation and/or filtering measures are proposed in each trial below.

Trial I Models

Object Detection

Trial I relies on the OpenCV (Bradski & Kaehler, 2008) package to perform cone detection. We first convert an RGB input image to a Hue Saturation Value (HSV) image. Unlike RGB, which is defined in relation to primary colors, HSV is defined in a way that is similar to how humans perceive color. Under the HSV framework, *hue* encodes color information, where zero degrees corresponds to the red, 120 degrees corresponds to the green, and 240 degrees corresponds to the blue color on the color wheel. *Saturation* encodes the intensity or purity of a color, where for example, pink is less saturated than pure red. *Value* encodes the brightness of color, including shading and gloss components.

To filter for a particular color, we specify a range on the color wheel for said color. After we apply this filter, we then remove noise via *erosion* and *dilation*. The former decreases the size of objects and removes small anomalies by subtracting objects with a radius smaller than the structuring element. The latter increases the size of objects, filling in holes and connecting disjointed areas that are smaller than the size of the structuring element.

Next, we apply first a *median blur* to smoothen the image, which helps display the object of interest more clearly, and then a Canny edge detection technique to find the edges of the object of interest. At this point the object will look more like a cone shaped triangle.

Finally, we identify the contours of the image. Contours can be explained simply as a curve joining all the continuous points along a boundary, having the same color or intensity. We apply an approximate polygon technique to the objects of interest. This provides a more angled structure to the object. We apply *convex Hull* to specifically isolate cone-shaped objects.

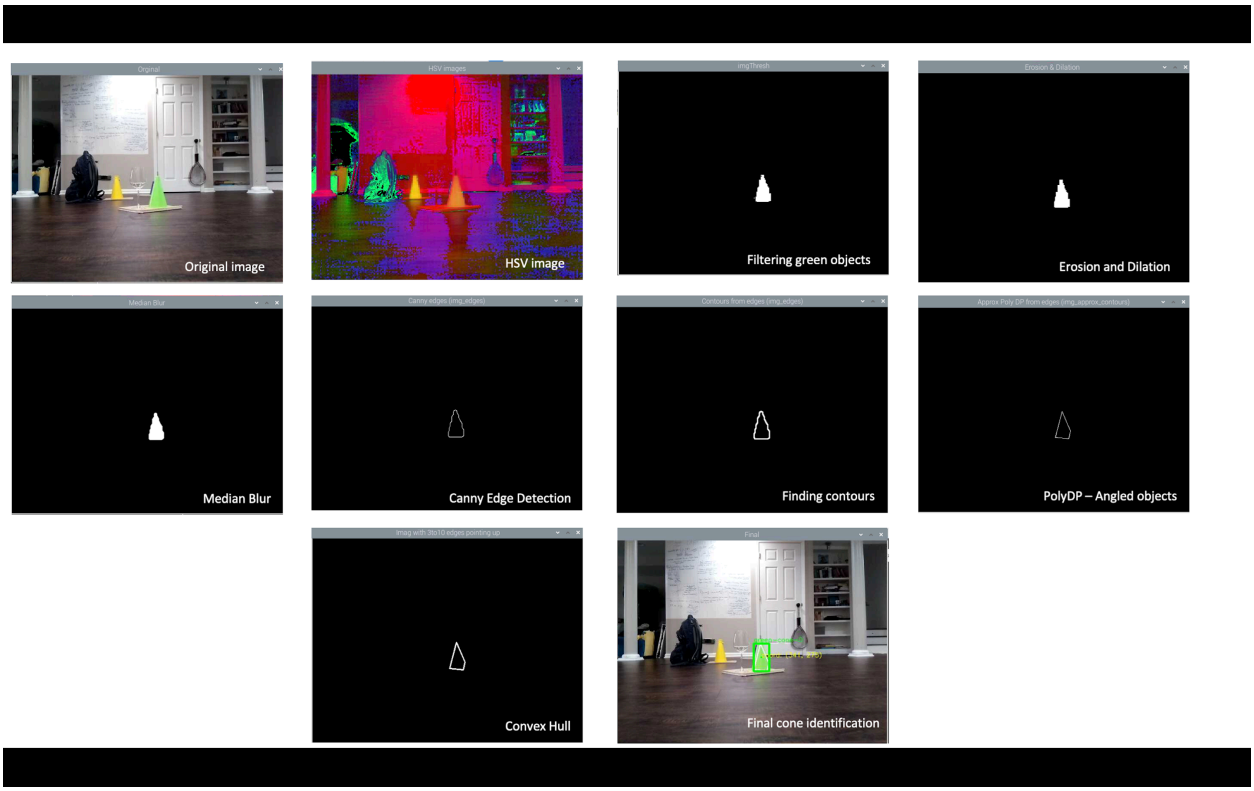


Figure 10: Cone identification with OpenCV

Trial I employs a pre-trained MobileNetV1 SSD for object detection. As previously discussed, MobileNet SSD is well-suited for real-time object detection given its speed and accuracy on relatively low-powered devices such as the Raspberry Pi 4. Also, unlike YOLOv3, it is easy to implement on the Raspberry Pi 4 with an *Intel Neural Compute Stick* (NCS) accelerator. To implement MobileNet SSD on the Raspberry Pi 4, we first need to create an environment compatible with our NCS. Intel developed the *OpenVINO* toolkit that allows for accelerated computing in its NCS devices. This toolkit and its dependencies are downloaded and configured to create the process of passing object recognition tasks through the USB port into the NCS. We subsequently activate the *OpenVINO* environment, download the pre-trained MobileNet SSD model weights, class labels, and model configuration, and compile the weights and configuration files (Rosebrock, 2018).

On implementation, MobileNet SSD automatically filters out inferences below a certain confidence threshold, constructs labeled bounding boxes around the objects detected in the image, filters inferences below a certain confidence threshold, and displays the resulting frame with class labels, bounding boxes, and prediction probabilities.

Audio Classification

Trial I uses an HMM for the audio detection component of the robotics challenge. As previously mentioned, one of the principal advantages of using HMMs for audio detection is that they can be trained on and tested with audio sequences of varying lengths. On the other hand, one of the chief disadvantages of HMMs in general is that they require the user to decide *a priori* the number of hidden states that compose the model. To attempt to remedy this, we employed *Self-Organizing Maps* (SOM) (Kohonen, 2001).

SOMs are a special class of neural networks used for unsupervised learning. SOMs are particularly adept at transforming high-dimensional, non-linear data into a low- (typically two) dimensional representation, preserving the topology of the original feature space (Asan & Ercan, 2011). As such, they present an useful unsupervised approach to determining *a priori* the number of hidden states.

Just as we will subsequently for training the HMMs, we train one SOM for each of the ten audio class labels. Moreover, in order to make our models robust to noise, we make two copies of the original 8,732 training instances and randomly mix them with pre-recorded robot noise resulting in approximately 26,000 training samples, 8,732 of which remained unchanged. We also maintain one corpus of training data where each recording was held at a 44.1 kHz sampling rate and create an additional copy of the entire training corpus where each sample

recording is downsampled to 22 kHz. Last, we convert all sample audio signals to MFCCs with 13 cepstral coefficients.

Training SOMs is a balance between reducing both topographic error and quantization error (Asan & Ercan, 2011), among other measurements. Given a map of N nodes, quantization error measures the average Euclidean distance between a training instance and its Best Matching Unit (BMU) node.[4] Topographic error, on the other hand, measures one minus the percentage of BMUs and second-BMUs that are adjacent in the node map. As such, such a measurement is appropriate for approximating topographical preservation of the original high-dimensional representation of the data. Furthermore, while quantization error decreases monotonically with the number of training iterations, topographical error behaves more erratically. In the end, we found that training on 20 iterations consistently produced a sufficiently low topographical error (less than 0.17 for nearly every label).

[4] For more details on the SOM training algorithm, see Asan & Ercan, 2011.

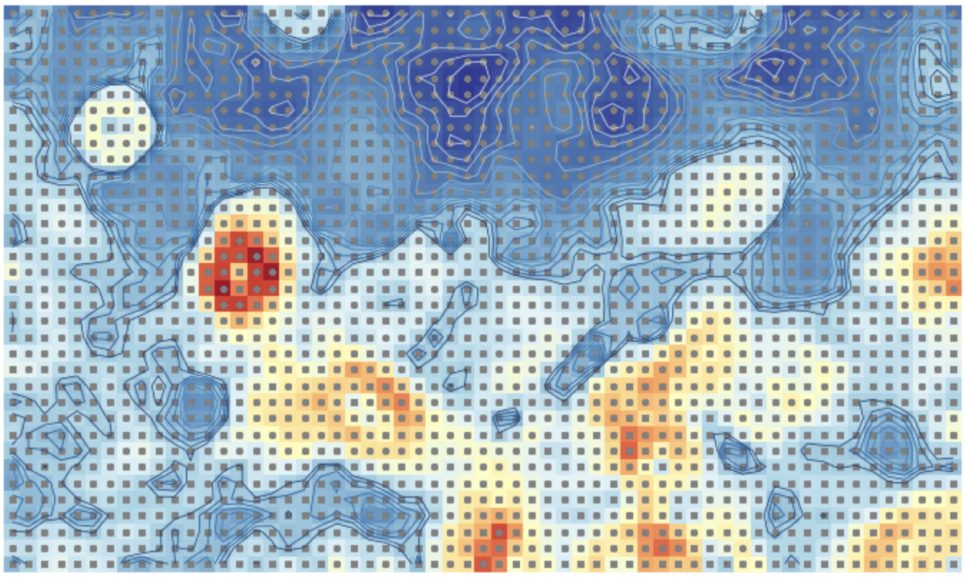


Figure 11: Contour plot of SOM for the "street music label". Dark colors represent 'hills' where adjacent nodes are far apart in Euclidean terms and bright colors represent 'valleys' where adjacent nodes are very close together.

In the absence of a clear objective function, unsupervised approaches such as SOMs do not produce obvious conclusions as to the ideal number of hidden states given a training set of MFCCs. That said, they can help indicate the complexity of the state space of one MFCC training set relative to the others. As such, we use observations from each SOM contour plot to better inform our decision as to the appropriate number of hidden states when training our HMMs for each audio label.

Given the ambiguity as to the appropriate number of hidden states, the effect of the sampling rate, and other hyperparameters such as the distributions of the hidden states, we opt to train a number of HMMs for each audio label and compare their recall scores:

Hyperparameters	*Number of Hidden States* (1)	12	Various	Various	Various
	Sampling Rate	40kHz	40kHz	40kHz	20kHz
	Hidden State Distribution (2)	Gaussian: Univariate	Gaussian: Multivariate	Gaussian: Multivariate	Gaussian: Multivariate
	EM Iterations (3)	200	2	50	250
HMMs	*Car Horn*	0.517510	**0.521401**	0.489637	0.501946
	Children Playing	**0.721667**	0.293333	0.413333	0.440833
	Engine Idling	0.501667	0.708333	**0.757778**	0.728333
	Drilling	0.511667	0.745000	**0.777778**	0.700000
	Siren	0.533214	0.601436	**0.771531**	0.659785
	Gun Shot	**0.691964**	0.256696	0.255952	0.272321
	Street Music	0.446667	0.656667	**0.754444**	0.689167
	Air Conditioner	0.343333	**0.790833**	0.671111	0.694167
	Jackhammer	0.728333	0.843333	**0.850000**	0.832500
	Dog Bark	0.485000	0.806667	0.747778	**0.813333**

Table 5: HMM recall scores. (1) When the table above indicates "various" hidden states, each label-specific HMM employed a different number of hidden states based on the SOM contour plots. (2) Univariate Gaussian distributions for the hidden states were used in cases when a single MFCC window was fed into the model as a training instance; multivariate Gaussian distributions were used in cases where MFCC windows for an entire sample audio recording were concatenated row-wise and fed into the model as a training instance. Thus, in the latter case, a single time step was multidimensional. (3) Each HMM was trained using the

expectation-maximization (EM) algorithm and validated by finding the highest log likelihood amongst all ten HMMs given the validation sample.

As can be seen in Table 5, each audio label-specific HMM performs differently depending on the hyperparameters chosen. Thus, on implementation, we choose the ten best-performing audio label-specific HMMs highlighted above.

Finally, we propose a sample criterion with which the robot will perform an audio classification using the data collected from the microphone's audio stream. Using a two-minute sample of robot noise, we measure the mean decibel level. We then choose an arbitrary threshold above this mean based on our observations of the decibel levels of concurrent robot noise and urban sounds broadcast from our speaker. On implementation, only when the decibel level surpasses this threshold does the robot send an audio sample from the stream to the HMM for inference.

Trial I Robotics Implementation

Real-time implementation of Trial I models is performed via five separate threads on the Raspberry Pi 4:

- *Thread 1: Main*
    - Controls robot locomotion and other movements.
    - Starts all other threads as needed.

- *Thread 2: Camera Feed*
    - Initializes and warms up the camera.
    - *Thread 1: Main* pushes images from *Thread 2: Camera Feed* to *Thread 3: Object Detection* queue.
    - *Thread 5: Live Video Feed* uses frames from *Thread 2: Camera Feed* to show a live feed to the user while the robot is moving.

- *Thread 3: Object Detection*
    - Pushes the object detection model on to the TPU.
    - *Thread 1: Main* captures pictures using *Thread 2: Camera Feed* and pushes the saved images path to the *Thread 3: Object Detection* queue.
    - Monitors its queue for any input image on which to perform inference. When an input image enters the queue, pushes the image to the TPU for inference and logs

the results.

- *Thread 4: Audio Detection*
    - Collects data streamed from microphone. If streamed audio data surpasses a predetermined decibel threshold, performs inference using the audio detection model and logs the results.

- *Thread 5: Live Video Feed*
    - Displays live feed using frames called *Thread 2: Camera Feed*.

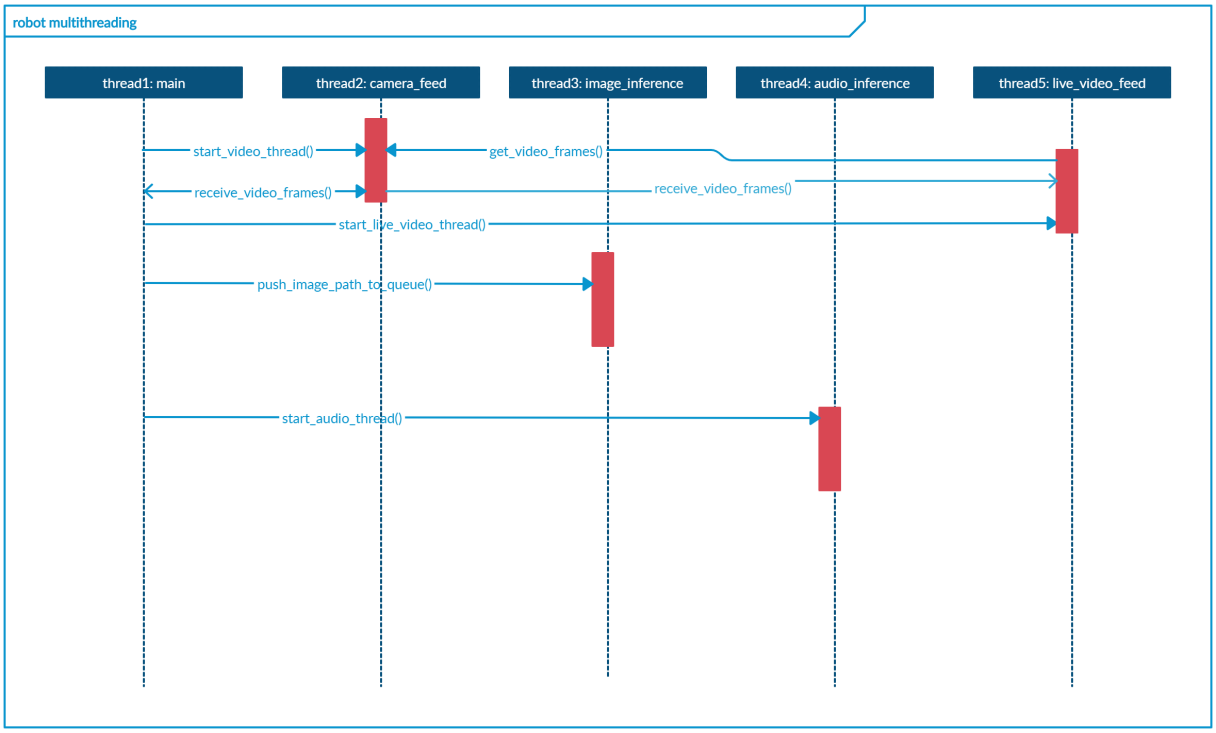


Figure 12: Trial I robotics implementation multithreading schema

Trial II Models

Trial II seeks to find the most efficient deep learning approaches to real-time object detection and audio classification. As such, we select models that can be converted to the lightweight TensorFlow Lite framework, making them both integer-quantizable and TPU-compatible. This significantly reduces model size, inference time, and framework dependencies.

Object Detection

Trial II uses MobileNetV2 SSD for both object and cone detection. Unlike YOLOv3, MobileNetV2 SSD is compatible with the Google Coral USB Edge TPU, which we found can increase inference speeds by a factor of 10. Using MobileNetV2 SSD, we can thus minimize object detection inference times and free up compute resources for the other components of the Scavenger Hunt Challenge. We note that although the TPU does increase power requirements, there was no obvious negative impact on the operations during the Scavenger Hunt Challenge.

We retrain the last four layers of two separate MobileNetV2 SSD models, respectively: one model to detect the cones and the other to detect the objects behind each cone. This ensures that both models are optimized for their respective objectives. To train our cone detection model, we captured 200 images of different colored cones using the Raspberry Pi camera. They were then manually labelled with bounding boxes and colors. To train our object detection model, we used 1,500 additional images of the 12 Scavenger Hunt Challenge objects and their corresponding bounding boxes from the Google OpenImages Dataset.

Out-of-sample recall scores for the object detection model are below. Not included are the results of the cone detection model, though we note that out-of-sample recall scores for cone detection surpassed 90%.

Type	Object Class	Validation Recall
Large	Suitcase	85.0%
	Backpack	81.2%
	Bicycle Helmet	87.4%
	Tennis Racket	95.0%
Medium	Book	79.5%
	Boot	95.5%
	Bottle	85.0%
	Wine Glass	93.9%
Small	Dice	99.2%
	Glasses	70.3%
	Tennis Ball	97.2%
	Banana	92.3%

Table 6: MobileNetV2 SSD Validation Recall Scores

Audio Classification

Trial II uses an EfficientNet CNN for audio classification and takes a filter-first approach to the training and signal processing. As such, the model is trained on the clean *UrbanSound8K* data, and during the robotics challenge, the robot's operating noise is filtered in real-time.

In order to mirror the pre-processing of the sound files to be broadcast during the robotics challenge, we pre-process the *UrbanSound8K* audio files by converting them to mono, downsampling them to 8 kHz, and applying peak normalization. The signals are then zero-right-padded to the maximum length of four seconds. Finally, each padded signal is converted into a

Mel spectrogram image of shape (300, 300, 3) with an FFT window length of 2,048 and 512 hops between successive frames. The 10 folds from the original data set are kept and used for cross-validation.

We use an EfficientNet-Lite4 CNN pre-trained on ILSVRC-2012 data to classify the Mel spectrogram images. Since the *UrbanSound8K* dataset contains 10 new classes, we apply a new classification header to the frozen feature vectors.[5] The classification header consists of a 0.2 dropout layer followed by a dense layer using softmax activation with an L2 regularization factor of 0.0001. The model is then quantized and converted to a TensorFlow Lite model.

Training of the classification header netted a 10-fold cross-validation top-1 accuracy of 78 percent. The resulting model is two-times faster than a similarly-trained ResNet-34 model, with only a two percent difference in top-1 accuracy on the *UrbanSound8K* data. Table 7 shows the out-of-sample metrics for one of the 10 validation folds.

[5] Although we experimented with unfreezing some of the top pre-trained layers, which resulted in higher top-1 accuracy scores, this had to be abandoned due to issues with quantization.

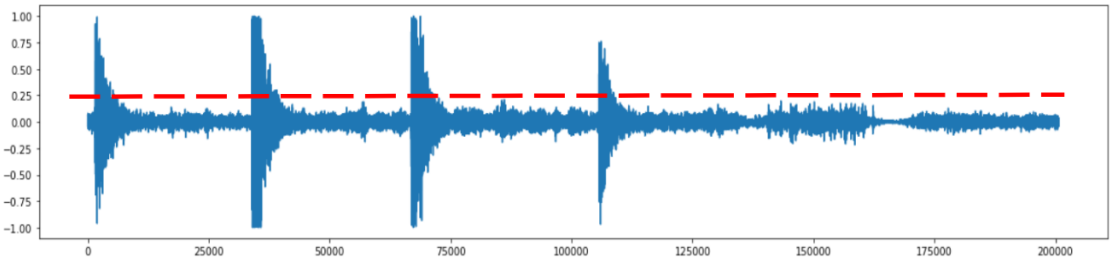
Audio Class	Precision	Recall	F1-Score
Air Conditioner	0.59	0.78	0.67
Car Horn	0.87	0.92	0.89
Children Playing	0.80	0.88	0.84
Dog Bark	0.75	0.93	0.83
Drilling	0.67	0.55	0.60
Engine Idling	0.86	0.52	0.65
Gun Shot	0.78	0.91	0.84
Jackhammer	0.84	0.80	0.82
Siren	0.85	0.91	0.88
Street Music	0.84	0.69	0.76
Top-1 Accuracy	**0.77**		

Table 7: EfficientNet-Lite4 Fold-1 Validation Metrics

Real-Time Audio Signal Processing and Noise Filtering

We begin the audio filtering process by analyzing the volume of the real-time audio stream, with the aim of recording data from the audio stream only when a certain volume threshold is surpassed. The audio stream is sampled at 44.1 kHz. We analyze the byte stream in chunks of 4,096 values, equivalent to roughly a tenth of a second. An effective way to measure the volume or 'energy' of a particular audio stream chunk is to calculate its root mean square (RMS). The RMS calculation is effective since it requires the squaring of all integer values, which will give equal weight to the positive and negative signal values. The mean RMS value of our sample recording of typical robot noise is less than 2,000 with a standard deviation of 250.

As a result, we set the threshold for triggering a recording to be a chunk with an RMS value that exceeds 3,000. Figure 12 depicts a four-second recording of a sample ‘dog bark’. We see four instances - each a separate bark - where the RMS of the chunk exceeds the typical robot noise threshold (red dashed line). When this happens, a 4.5-second recording begins when the threshold is first exceeded. Since the urban sound samples are no more than four second long, we use the last 0.5 seconds of the recording as the noise sample for filtering. This is useful because the exact character of robot noise depends on what the robot is doing when the urban sound



sample is broadcast.

Figure 13: Dog bark decibel levels versus RMS-determined noise threshold

We next propose a filtering technique for removing the ambient robot noise. We choose a spectral gating noise reduction algorithm originally proposed by The Audacity Team (2015). Spectral gating selectively filters the unwanted ambient noise while attempting to maintain the integrity of the target sound signal. Spectral gating takes two audio files as inputs: the raw audio segment to be filtered and a sample of the noise that we want removed from the raw segment. The result of the algorithm is a cleaned signal.

The algorithm begins by performing a Fourier transform of the noise signal, binning it into 1,025 distinct frequency bands. It then calculates the mean and standard deviation power for each frequency band. The resulting metrics will be used to calculate a threshold that represents

the peak power of noise expected at each frequency band (Figure 14). The aforementioned process is also applied to the raw signal to be filtered.

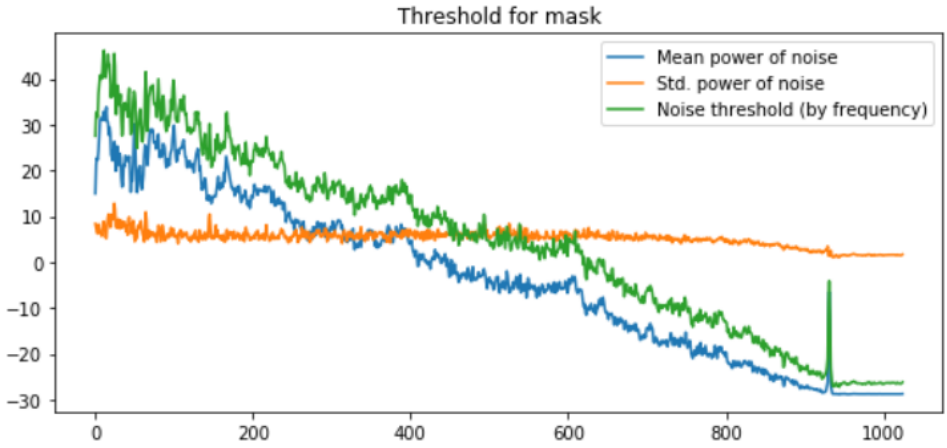


Figure 14: Noise power metrics by frequency band. The x-axis represents the count of frequency bands while the y-axis represents the power of the noise signal at the particular frequency bands. This view of the metrics matches deductions made in the previous spectrogram. The noise has higher power at lower frequencies and lower power at higher frequencies. The standard deviation of the noise is relatively small and consistent. We can also see a low power, but distinct signal coming from the motors at a higher frequency bin.

Next, the algorithm develops a filtering mask. The mask is built by comparing the original signal mean power at each frequency band to the noise threshold at each frequency band. If the power of the noise threshold is larger than the mean power of the original signal, then the frequency bin becomes part of the mask. This process is conducted separately for all frequencies at all time windows of the original signal. The previous step ensures that the mask is dynamic over time, meaning that certain frequencies are omitted from the mask at certain time periods when there are strong signals that are not suspected to be noise. The filtering algorithm smooths the mask over frequency and time, ensuring there are no sharp cutoffs or losses and that some portions of noise and signal will be able to bleed into the recovered signal.

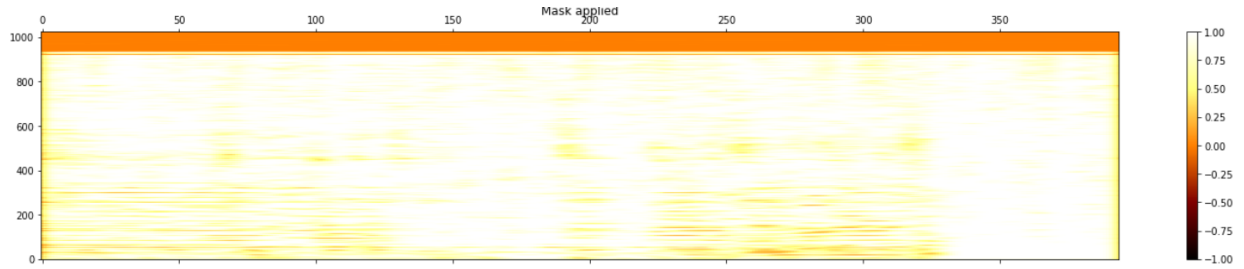


Figure 15: Temporal spectral density of mask for filter. The first portion of the signal has some unmasked portions seen, followed by a short pause where nearly everything is masked and then more signal is unmasked. The final portion of the clip seems to have no signal and is completely masked. There is also a clear mask at the high frequency bin where we anticipate a faint motor noise.

Finally, the algorithm applies the mask to the original signal over the time windows in the frequency domain. The resulting spectrogram is depicted at the bottom of Figure 16. Figure 16 compares the spectrogram of the filtered signal to the spectrograms of the original recording and the signal from the *UrbanSound8K* dataset. We see in the unfiltered spectrogram that much of the definition of the urban sound sample is obfuscated by the robot noise during the recording process. Filtering allows us to recover the definition and overall sonic characteristics of the urban sound sample, which can be confirmed by listening to the filtered sound. Much of the high-frequency definition (the top of the spectrogram) is lost, due to the compound effect of three factors: 1) the robot motor noise containing similar high frequencies, 2) the limited recording quality of the microphone and broadcast quality of the speaker, and 3) the decay of high-frequency sounds across distances when broadcast from the speaker.

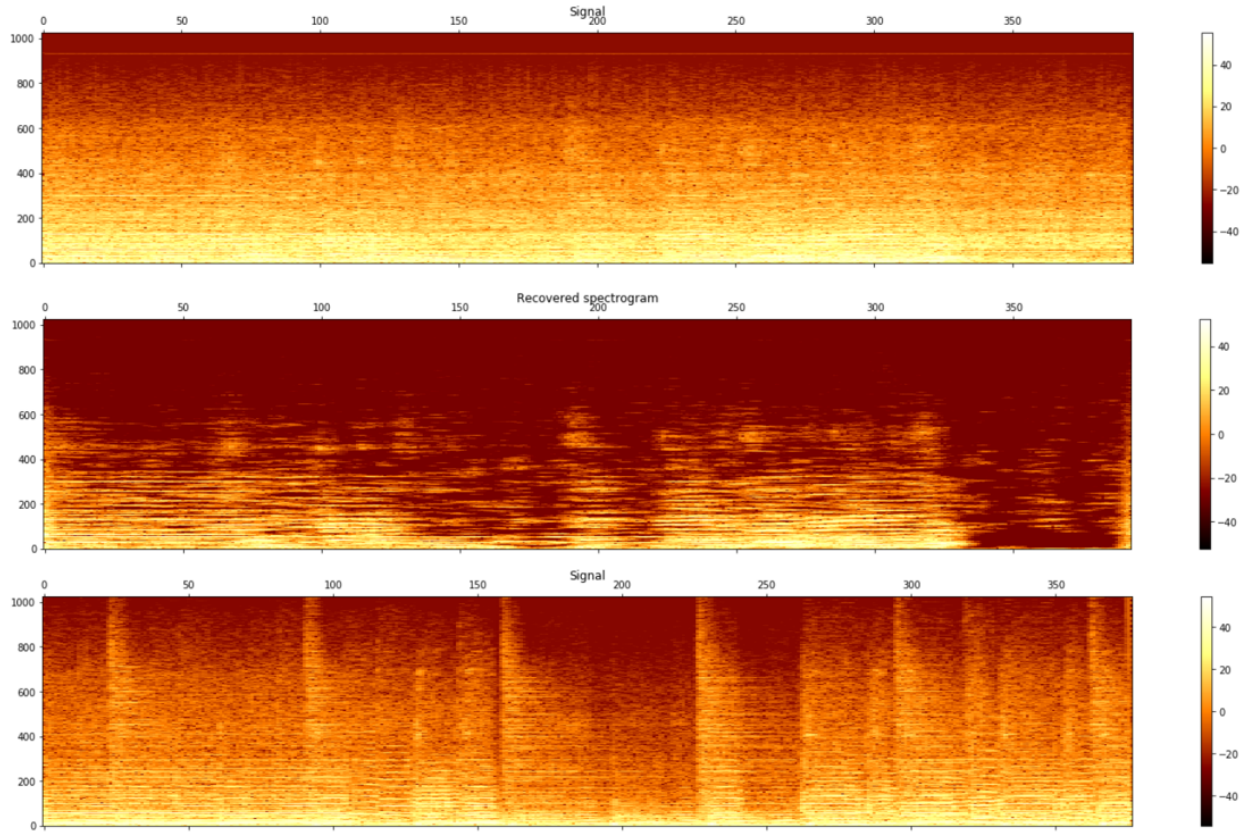


Figure 16: Temporal spectral densities in comparison. Temporal spectral densities of raw recording (top), recovered signal after noise filtering (middle), and the original 4-second audio clip from the *UrbanSounds8k dataset* (bottom)

Since the audio model requires a different spectrogram resolution than the noise filtering algorithm, we ultimately inverse Fourier transform the filtered spectrogram back to the time domain and pass on this result to the audio model for Mel spectrogram conversion and inference.

Trial II Robotics Implementation

Trial II uses a multithreading approach for implementation. This allows the robot to drive, identify objects, and recognize sounds. We chose multithreading over multiprocessing after preliminary tests showed that multiprocessing was not well supported on our hardware and software configuration. Real-time implementation of Trial II models was performed using two separate threads on the Raspberry Pi 4:

- *Thread 1: Main - Controls and Locomotion*
    - Operational control for the scavenger hunt and initialization of other threads
    - Directs robot speeds and movements based on programmed hunt objectives and responses from the cone detection in *Thread 2*.

- *Thread 2: Cone and Hidden Object Detection*
    - Controls the capture and processing of images and video from the camera.
    - Sends images to the Cone Detection model on TPU for inference and returns relative position of desired cone (if found) to *Thread 1*.
    - Sends video frames to the Hidden Object Detection model on TPU for inference and returns the class of the hidden object (if found).
    - Logs timing and results of cone and object detection and categorization

- *Thread 3: Audio Detection*
    - Initiates and controls continuous collection and processing of audio.
    - Sends filtered recordings to the Audio Detection model on CPU for inference and returns the predicted class of the sound heard.
    - Logs timing of audio events and resulting categorization.

Trial III Models

Object Detection

For cone detection, Trial III employs YOLOv3 via the Keras API for transfer learning, freeze the first 249 of 252 total layers, and fine-tune the model on a set of 500 manually captured and labeled images of orange, green, yellow, and purple cones in surroundings similar to those of the robotics *environment*. Altogether The YOLOv3 fine-tuning process consists of 100 epochs with early stopping and takes approximately 22 hours.

We find that the bounding boxes from YOLOv3 inferences effectively enables the robot to locate and navigate to the cones. We can derive the angle and distance from the robot to the cone using the bounding box coordinates and the properties of triangle similarity. We find that calculating the angle of attack and distance in such a way is preferred over the GoPiGo distance sensor given the relative inaccuracy of the sensor.

During the Scavenger Hunt Challenge, the robot needs to know the approximate distance from and angle to the cone. Let F be the focal length of the camera, P be the width of the pixels

of the cone calculated from bounding box coordinates, *D* be the distance to the cone, and *W* be the actual width of the cone (5.5 inches for the scavenger hunt challenge), triangle similarity is calculated as:

$$F = \frac{P * D}{W}$$

We propose the following formula (Figure Xi) to measure the angle of attack from the robot to the cone. Let *I* be the center of the image and *C* be the center of the predicted cone, measured in pixels. We divide the difference between *I* and *C* by the constant, 12.[6] A positive result indicates a clockwise rotation and a negative result indicates a counterclockwise turn:

$$R = \frac{I - C}{12}$$

We opted not to perform cone detection inferencing using the Google Coral USB Edge TPU. This is because we found the Keras API to be a necessary component of this implementation, given the ease with which it can translate the original Darknet implementation of YOLOv3 into Python. That said, unfortunately, Keras made converting the model to the TFLite format required by the Coral TPU infeasible because of the potential information loss when the original weights are quantized.

We use an ensemble approach for inference, where the robot captures 10 images per inference and selects the class predicted for the majority of the 10 images. The model has an average inference time of 1.55 seconds and an average accuracy score of 87%.

For object detection, we use MobileNetV2 for transfer learning. The benefits of the MobileNetV2 model include high accuracy and inference speed on the Coral Edge TPU. Training the model took approximately 10 hours for 100 epochs with early stopping after

[6] We determined that a horizontal movement of 12 pixels was equivalent to a GoPiGo3 rotation of one degree.

freezing the first 50 of 53 total layers. We downloaded 500 open source images representing our 12 Scavenger Hunt Challenge objects and manually labeled them.

Because MobileNetV2 can be converted to a TFLITE file format, we use the Google Coral USB Accelerator TPU to speed up our inferences. As with the cone detection model, an ensemble approach is used for inference. The robot takes 10 images and then selects the class predicted for the majority of the 10 images. The model achieves an average inference time of 330 milliseconds and an accuracy score of 91%.

Audio Classification

For audio detection, Trial III uses transfer learning from a ResNet34 CNN pre-trained on ImageNet data. Indeed, we find transfer learning using ResNet34 - a model trained on image data - to be much more successful than training a CNN on audio data from scratch. We implement ResNet34 using the FastAI package built on PyTorch. We first train the head to make predictions on the 10 audio classes from the *UrbanSound8K* dataset, and then we unfreeze the remaining layers of the model and train the whole model. We use FastAI's *fit_one_cycle* option that uses a cyclical learning rate (Smith, 2017), meaning the learning rate starts at some arbitrary low value, increases to a predetermined max value, and then decreases to zero. Training at a cyclical learning rate has been shown to require fewer epochs to adequately train. Prior to training, we augment *UrbanSound8K* data with noise generated from the GoPiGo3 while navigating the robotics *environment*. After mixing the sounds, we convert them to mono and downsample to an 8 kHz sampling rate. The volume of the GoPiGo3 noises is adjusted such that the urban sound samples are still audible after mixing, by adjusting the GoPiGo3 noise to have the same max volume as each urban sound sample. The clean urban sound samples as well as the augmented

urban sounds are converted into Mel spectrogram images and used to train the ResNet34 CNN. Table 8 shows the out-of-sample metrics for one of the 10 validation folds.

Audio Class	Precision	Recall	F1-Score
Air Conditioner	0.75	0.71	0.73
Car Horn	0.95	0.89	0.92
Children Playing	0.86	0.73	0.79
Dog Bark	0.80	0.87	0.83
Drilling	0.74	0.77	0.76
Engine Idling	0.91	0.71	0.80
Gun Shot	0.94	1.00	0.97
Jackhammer	0.78	0.94	0.85
Siren	0.89	0.94	0.91
Street Music	0.85	0.89	0.87
Accuracy			0.83

Table 8: ResNet34 out-of-sample metrics

Similar to Trial I, on implementation, sound is captured for inference from the real-time audio stream only when a certain volume threshold is surpassed.

Trial III Robotics Implementation

Real-time implementation of Trial IIII models is performed via three separate threads on the Raspberry Pi 4:

- *Thread 1: Signal Detection, Object Detection, and Robot Movements*
    - Waits for signal to start robot.
    - When signal is detected, initiates cone detection and robot movement program.
    - Sends cone location data to movement class, which directs robot to each respective cone.

    - When the robot is in position behind each target cone, detects objects and logs the results.

- *Thread 2: Audio Detection Recording*
    - Collects data streamed from microphone. If streamed audio data surpasses a predetermined decibel threshold, saves recording of audio to file.

- *Thread 3: Audio Detection Inference*
    - Performs inference using the audio detection model and logs the results.

Findings

Trial I

For Trial I, the robot was able to accurately classify the medium and large-sized objects. Nevertheless, it struggled to classify the eyeglasses. This may be because MobileNet often struggles to identify smaller objects vis-a-vis approaches such as YOLOv3. It also may be because our MobileNetV1 SSD was not trained on standalone eyeglasses, but rather eyeglasses worn on a face.

Trial I Results					
Object Detection			**Audio Classification**		
Timestamp from Start (ms)	**Prediction**	**Actual**	**Timestamp from Start (ms)**	**Prediction**	**Actual**
59,449	Wine Glass	Eyeglasses	61,123	Dog Bark*	Dog Bark
143,180	Wine Glass	Wine Glass	160,258	Car Horn*	Car Horn
238,464	Backpack	Backpack	200,860	Children Playing	Street Music
281,409	End at Orange Cone		245,878	Children Playing	Gun Shot
			259,458	Car Horn	Siren

Table 8: Trial I Results. *Although the HMM was consistently able to correctly classify Dog Barks and Car Horns during preliminary tests on live-streamed audio data, given the unusual timestamps recorded from Trial I, we cannot unequivocally affirm that those two audio classes were correctly inferred.

Trial I's HMM performs inference efficiently, but was completely unable to classify certain sounds (e.g. Gun Shot and Drilling) and often struggled to differentiate other sounds (e.g.

Children Playing versus Street Music and Car Horn versus Siren). That said, the aforementioned sounds that it struggled to differentiate often had noticeably overlapping features. For example, the sounds of traffic could be heard in many of both the Siren and Car Horn samples of the *UrbanSound8K* dataset. Finally, we were unable to get consistent decibel readings with which to reliably trigger audio classification when (and only when) an urban sound sample was broadcast from our speaker. We note that while Trial I employed a flat, tabletop omnidirectional microphone, Trials II and III both employed a gooseneck omnidirectional condenser microphone. Indeed, the extra proximity of the tabletop microphone to the robot noise may have made audio classification more challenging.

Trial II

For Trail II, the robot successfully navigated the course with limited errors. The large, medium, and small hidden objects were correctly classified, with one caveat: the eyeglasses were only identifiable when placed on an image of a face. This is because all of the eyeglasses samples in the training data were of eyeglasses worn on a face.

Trial II Results					
Object Detection			**Audio Classification**		
Timestamp from Start (ms)	**Prediction**	**Actual**	**Timestamp from Start (ms)**	**Prediction**	**Actual**
31,577	Eyeglasses**	Eyeglasses	14,349	Dog Bark	Dog Bark
86,983	Wine Glass	Wine Glass	32,423	Car Horn	Car Horn
139,538	Backpack	Backpack	58,219	Street Music	Street Music
156,255	End at Orange Cone		77,971	Dog Bark	Gun Shot
			101,102	Siren	Siren

Table 9: Trial II Results. **Eyeglasses were placed on an image of a face during the challenge

The Trial II audio model successfully classified four out of five urban sounds played during the change, misclassifying only the Gun Shot as a Dog Bark. This is possibly due to the short length of the single-shot Gun Shot sample. In general, the audio model was less reliable at classifying shorter sounds, which is not surprising since the spectrograms corresponding to the shorter sound sample were right-zero padded and CNNs often struggle with correctly learning information at the edge of images.

Trial III

For Trial III, the robot successfully classified every object except for the eyeglasses. The audio model successfully classified the Dog Bark, Car Horn, and Street Music samples, but failed to differentiate the Children Playing from the Gun Shot samples and the Drilling from the

Siren samples. Thus, although the Trial III's ResNet outperformed Trial II's EfficientNet-Lite4 during validation in terms of top-1 accuracy, EfficientNet-Lite4 performed better during the Scavenger Hunt Challenge.

Trial III Results					
Object Detection			**Audio Classification**		
Timestamp from Start (ms)	**Prediction**	**Actual**	**Timestamp from Start (ms)**	**Prediction**	**Actual**
40,973	None	Eyeglasses	14,631	Dog Bark	Dog Bark
87,073	Wine Glass	Wine Glass	33,113	Car Horn	Car Horn
157,753	Backpack	Backpack	59,096	Street Music	Street Music
164,923	End at Orange Cone		78,501	Children Playing	Gun Shot
			102,080	Drilling	Siren

Table 10: Trial III Results

Furthermore, we note from the timestamps that compared to Trials I and III, Trial II overall completes its audio classifications and object detections the fastest. As far as audio classification, this rather clearly shows that Trial II's edge computing-optimized audio classification model out-performs Trial III's ResNet model in terms of efficiency, particularly considering Trial II's audio model includes an additional 0.5 second delay for filtering. It is more

difficult to compare the efficiency of Trial II and Trial III's object detection models from the timestamps because these logs are a function not only of inference time but also of the time that it takes the robot itself to navigate the robotics *environment*. Nevertheless, it is suggestive that, while Trial II and Trial III's object detection log times are fairly consistent, Trial II ultimately outperforms Trial III by around 17,000ms.

Summary and Conclusions

Our three approaches to implementing object detection and audio classification for robotics applications on the Raspberry Pi yield varying results. We summarize our finding as follows:

- The object detection algorithms employed in this research, like any other machine learning algorithm, are limited by the data to which they are exposed. As such, in the case of the eyeglasses object class, said object detection algorithms are hard pressed to make a correct classification with a high degree of certainty if the eyeglasses are *not* worn on a human face.
- Deep learning approaches to audio classification out-perform probabilistic approaches such as HMMs when tested on validation and in real-time for robotics applications, though they do not do so substantially, particularly given the additional compute power required by deep learning models on inference. That said, we make the caveat that part of the difference in performance between these two approaches during the Scavenger Hunt Challenge may be due to the difference in microphones employed by the robots.
- We see a notable difference in the amount of time it takes the robot to compete the Scavenger Hunt Challenge between Trial I, which employs a robotics application with five total threads, and Trial II and III, both of which employ a robotics application with only three threads. This is in addition to the fact that while Trial I and Trial II use an accelerator for inference (NCS in the former case and Coral TPU in the latter), Trial III does not rely on an accelerator.

- We see a notable different in the inference times of Trial II's edge-computing optimized EfficientNet-Lite4 audio classification model and Trial III's ResNet audio classification model.
- Although Trial III's ResNet outperformed Trial II's EfficientNet-Lite4 during validation in terms of top-1 accuracy, EfficientNet-Lite4 performed better during the Scavenger Hunt Challenge.

One key limitation of this study is that we only performed one live Scavenger Hunt Challenge for each of the three Trials. Given the often-tenuous conditions of the robotics *environment*, including the microphones' sensitivities to varying noise conditions and the cameras' sensitivities to varying light conditions, it would be useful to perform multiple tests of each Trial and study the variance of the performance of the three Trials. Moreover, by using two different types of microphones, we limited our ability to effectively compare the accuracy of our audio classification models. Future research should ensure that all sensor equipment (e.g. cameras, microphones, distances sensors) is consistent across Trials.

Across all three Trials, our audio classification models had a more difficult time classifying shorter sound samples such as the Gun Shot. As such, future studies may consider training audio classifications models – be they neural networks or HMMs – using sample lengths that match the average length of the shortest audio class. In the case of the *UrbanSounds8K*, this would mean that four-second sound samples would need to be divided into multiple one or two-second sound samples to match the average length of the Gun Shot sound sample.

Next, because each of our deep learning models for audio classification only unfroze a minimal number of layers – if any – when employing transfer learning, future studies may wish to experiment with unfreezing more layers.

Finally, we discuss how one of the greatest challenges of HMMs is the requirement that the user decide *a priori* the number of latent variables. As a result, future research may consider exploring the use of a variant of a Hierarchical Dirichlet Process HMM (HDP-HMM) as proposed by Fox (2009) in order to determine the appropriate number of hidden states conditioned on the data via Gibbs sampling or a related method. Such HDP-HMMs may rival or even out-perform deep learning approaches to audio classification.

In the end, with the growing computational power of mobile computing devices such as the Raspberry Pi in addition to the availability of small USB accelerators such as the ones employed in this research, we see ample opportunity for the growth of cheap and effective intelligent devices going forward.

References

Asan, U. & Ercan, S. (2011). An Introduction to Self-Organizing Maps. In Cengiz Kahraman (Eds.), *Computational Intelligence Systems in Industrial Engineering with Recent Theory and Applications* (pp. 299-319). Atlantis Press.

Audacity Team. (2015, March 30). *How Audacity Noise Reduction Works*. Audacity. Retrieved from https://wiki.audacityteam.org/wiki/How_Audacity_Noise_Reduction_Works

Boashash, B. (2016) *Time-Frequency Signal Analysis and Processing: a Comprehensive Reference*.

Bochkovskiy, A, Wang, CY, & Liao, HY (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. Retrieved from https://arxiv.org/pdf/2004.10934.pdf

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV*. O'Reilly Media, Inc.

Dalal, N. & Triggs, B. 2005. Histograms of oriented gradients for human detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 886-893. https://ieeexplore.ieee.org/document/1467360.

Davis, S. & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4): 357-366. doi: 10.1109/TASSP.1980.1163420.

Dennis, A. K. (2015). *Raspberry Pi Home Automation with Arudino* (2nd ed.). Packt Publishing.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9): 1627-1645, 2010.

Gales, M. & Young, S. (2008). The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends in Signal Processing*. 1(3), 195-304.

Gerón, A. (2017). *Hands on Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Gu, J., et. al. (2018). Recent advances in convolutional neural networks. Pattern Recognition, 77, 354–377. doi: 10.1016/j.patcog.2017.10.013

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi: 10.1109/cvpr.2016.90

Hersey, S., et al. (2017). CNN Architectures for Large-Scale Audio Classification. *arXiv preprint arXiv:09430v2, 2017.*

Howard, A. G., et al. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861v1, 2017.*

Hui, J. (2018, March 28). What do we learn from single shot object detectors (SSD, YOLOv3), FPN & focal loss (RetinaNet)? Retrieved from medium.com

Kathuria, Ayoosh (2018) *What's New in Yolov3.* Retrieved from https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

Kohonen, T. (2001). *Self-Organizing Maps* (3rd ed.). Springer-Verlag.

Lin, T. Y., Maire, M., Belongie., S. et. al. (2015). Microsoft COCO: Common Objects in Context. Retrieved from https://arxiv.org/pdf/1405.0312.pdf

Liu, R, et al. (2020, March 16). Higher accuracy on vision models with EfficientNet-Lite. *TensorFlow Blog.* https://blog.tensorflow.org/2020/03/higher-accuracy-on-vision-models-with-efficientnet-lite.html

Liu, Wei et al. (2016). “SSD: Single Shot MultiBox Detector.” Lecture Notes in Computer Science (2016): 21–37. Crossref. Web.

Mordor Intelligence. (2019). Robotics Market Size, Growth, Analysis: Growth, Trends, and Forecast (2019 - 2024). Retrieved from https://www.mordorintelligence.com/industry-reports/robotics-market.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. Retrieved from https://arxiv.org/pdf/1506.02640.pdf

Redmon, J. & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. Retrieved from https://pjreddie.com/media/files/papers/YOLOv3.pdf

Rosebrock, Adrian (2018) Real Time Object Detection on the Raspberry Pi with the Movidius NCS. Retrieved from https://www.pyimagesearch.com/2018/02/19/real-time-object-detection-on-the-raspberry-pi-with-the-movidius-ncs/

Sainbury, T. (2018, July 7) *Noise reduction using spectral gating in python.* Retrieved November 20, 2019 from https://timsainburg.com/noise-reduction-python.html

Salamon, J., Jacoby, C., & Bello, J.P. (2014). A Dataset and Taxonomy for Urban Sound Research. *Proceedings of the 22nd ACM international conference on Multimedia,* 1041-1044. https://dl.acm.org/doi/10.1145/2647868.2655045.

Sandler, M., Howard, A., et al. (2019) MobileNetV2: Inverted Residuals and Linear Bottlenecks. Retrieved from https://arxiv.org/pdf/1801.04381.pdf

Smith, L. (2017). Cyclical Learning Rates for Training Neural Networks. *arXiv preprint arXiv:1506.01186v6, 2017.*

Tan, M & Le, V. Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Retrieved from https://arxiv.org/pdf/1905.11946.pdf

Veit, A., Wilber, M., & Belongie, S. (2016). Residual Networks Behave Like Ensembles of Relatively Shallow Networks. Retrieved from https://arxiv.org/pdf/1605.06431.pdf

Wang, C. (2018, August 13). A Basic Introduction to Separable Convolutions. Retrieved from towardsdatascience.com