A Low-Cost Embedded Computer Vision System for the Classification of Recyclable Objects

Karl Myers[1, 2] and Emanuele Lindo Secco[2]

[1] Study Group Liverpool, UK
[2] Robotics Lab, School of Mathematics, Computer Scinece and Engineering, Liverpool Hope University, UK
15008411@hope.ac.uk, seccoe@hope.ac.uk

Abstract. Due to rapid urbanization, increasing population and industrialization, there has been a sharp rise in solid waste pollution across the globe. Here we present a novel solution to this inefficiency, by the use of embedded Computer Vision (CV) in the Material Recovery Facilities (MRF).
The proposed architecture employs software (i.e. Tensorflow and OpenCV) and hardware (i.e. Raspberry Pi) as an embedded platform in order to classify daily life objects according to their visual aspect. The CV system is trained using modules contained within the TensorFlow API with two datasets, namely the TrashNet and a combination of the TrashNet and a set of web images.
This solution allows greater accuracy, with a baseline performance of 90% which drops to 70% when deployed on the embedded platform, due to the quality of the images taken by an integrated camera for the real-time classification. The speed results are also promising with a baseline speed of 10 FPS at simulation level, which drops to 1. 4fps when running on the platform.
Such a system is cheap at less than £ 100, it is perfectly adequate to be used to identify recyclables in the MRF for sorting.

Keywords: Computer Vision, Tensor Flow Low-cost prototype.

1 Introduction

Rapid urbanization, the increasing human population and industrialization have increased environmental pollution across the globe. With the increase in activities such as production and marketing, the use of natural resources has increased. However, with this increase there has been an inevitable and significant increase in the solid waste products produced. Due to these consumption tendencies the increasing levels of such waste products are now and will continue to be detrimental to the environment and ultimately to the humans and animals that live there.

Due to this there has been increasing research into the retrieval of such waste from the environment such as the SeaVax solution, a roaming, satellite controlled aluminium platform powered by the sun and wind. It operates like a giant vacuum cleaner, shredding and compressing waste products into its hopper.

The use of such retrieval methods has significantly increased the removal of waste from the environment by many thousands of tons. However, the issue with these systems is they do not discriminate between what waste they retrieve which poses the question “Do we have the capacity to handle such an increase in waste?”

According to the Department for Environment, Food & Rural Affairs (DEFRA), the UK alone generated 222.9 million tons of solid waste in 2017 and of this 47% was recycled. The amount of waste generated is expected to rise by at least 20% by 2020, and the UK aims to increase the recycling rate to 50% in the same year (DEFRA, 2018).

The methods of recycling may differ country by country. As an example, in the UK it is known as single-stream recycling. Single stream recycling refers to a system in which all kinds of recyclables such as paper, metals, and glass are put into a single receptacle by the consumer. Afterwards the recyclable waste is collected and sent to a *Material Recovery Facility* (MRF) for sorting and processing.

At the MRF a combination of mechanical and manual methods are employed to sort the materials into their respective classes. In general, the sorting process is as follows:

1. All of the materials are placed on a conveyor and non-recyclables are manually removed by operatives
2. The remaining waste moves to a triple deck screen. There, all cardboard, containers and paper are removed.
3. The remaining materials, pass under a powerful magnet to remove tin and steel cans.
4. A reverse magnet, called eddy current, causes aluminium cans to fly off the conveyor and into a storage container.
5. Finally, the remaining plastics such as bottles are removed, crushed and bailed.
6. All through the previous steps, operatives have to continually monitor the materials coming from each stage to remove and sort the materials the mechanical process misses.

The main benefit of single-stream recycling is increased recycling rates. The increased rates are due to individuals or consumers not having to do the sorting themselves and are more likely to participate in the curbside recycling programs. From the collection point of view, costs for the hauling process are reduced versus separate pickups for different recycling streams, or the haulier having to place different materials into various truck compartments.

However, the most notable disadvantages of the single-stream recycling system is that it has led to a decrease in the quality of recovered materials and the cost of recycling is inherently higher due to the human resource cost. For these reasons, the industry trend is towards state of the art MRFs and a move away from legacy or "dirty" MRFs which are much more labour intensive.

Therefore, the answer to the question is no, at present we do not have the capacity to handle an increase in waste and moreover we do not have the capacity to handle the waste we are retrieving now. Hence, efforts must be made to streamline the waste sorting process and also efforts into intelligent retrieval of waste to further ease the pressure on the MRFs must be increased.

The aim of this study is to develop a multi-purpose embedded object detection CV system that could be used both in the sorting process at the MRF when integrated with, for example, robot arms and for intelligent retrieval and sorting when integrated into a retrieval method.

To do this the system must:

- Be able to accurately identify the five classes of recyclables; glass, plastic, metal, paper, cardboard.
- Be fast enough to be used for real-time retrieval.
- Be easily integrated with other equipment, with a view to fully automate the sorting process and to facilitate intelligent retrieval of waste to further ease the pressure on the MRF.
- Be as cheap as possible, as the cheaper the system, the more likely it would be that the system would be adopted in the future.

In order to match such requirements, the following objectives are defined:

- Train a *Computer Vision* (CV) model that is accurate and fast enough to be used for both sorting and retrieval.
- Develop programs for two modes of operation; sorting in the MRF and also intelligent retrieval
- Implement the system on an embedded platform which can be easily integrated into either system

2 Materials and Methods

2.1 Hardware

The Raspberry Pi is a low cost, single-board computer that can be connected to a monitor, via HDMI, using a standard keyboard and mouse as an interface just as any other PC. It can also be controlled remotely via a PC or laptop using software such as VNC Viewer. It is a very capable device that enables people of all backgrounds to explore computing and to learn how to program in languages such as Python. It is has the ability to do everything you would expect a low range desktop computer to do, from internet browsing and playing HD video, to general computing such as creating spreadsheets and word-processing.

Furthermore, the Raspberry Pi has the ability to interact with the outside world with the availability of a wide range sensors and modules. Due to this, it is has been adopted for use by both researchers and hobbyists for use in such things as IoTs, and digital maker projects.

For this project, the Raspberry Pi 3 Model B+, Shown in Figure 1, will is used as the platform on which the system will be deployed. The full specifications of the board are shown in table 2. It was chosen firstly due to the success recorded by [1]using its pre-decessor, its ease of use and moreover, its ultra-low-cost.

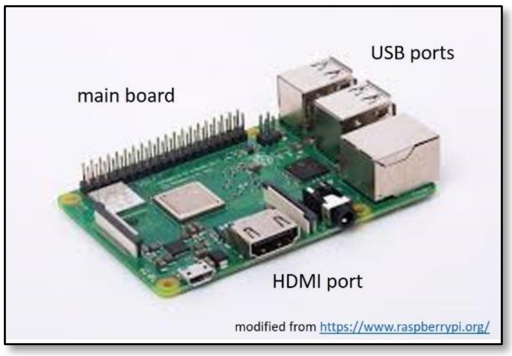


Figure 1 - The embedded PC

The optical sensors are used to capture the image data, which is processed and used for inference. Two sensors will be used and compared. The first sensor is the PiCam module, shown in Figure 2(a). The PiCam module is the new official camera board released by the Raspberry Pi Foundation.

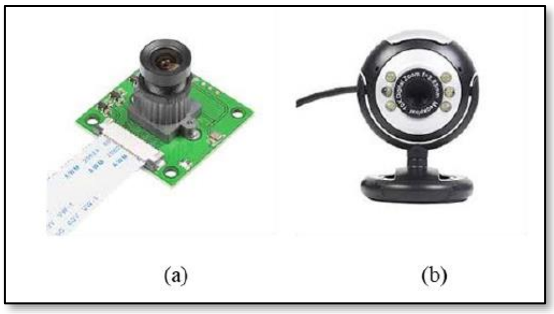


Figure 2 - The optical sensors

The module contains a medium quality 5 Megapixel Sony IMX219 image sensor designed as a custom add-on board for Raspberry Pi. It features a fixed-focus lens and is capable of 2592 x 1944 pixels resolution on static images, and supports 1080p30, 720p60 and640x480p90 video. It connects to the Raspberry Pi by way of one of the small sockets on the boards upper surface and uses the dedicated MISI interface, specially designed for interfacing to cameras.

The second is a standard mid-range 12MP web camera, shown in figure 3-2(b). It features a multi-focus lens and is capable of 3280 x 2464 pixel resolution on static images, it supports 1080p and 720p video and connects to the Raspberry Pi via USB.

For this project, the two sensors will be used and compared to determine which sensor performs best in terms of image quality and speed for both single capture and inference and real-time capture and inference.

They were both chosen for comparison due to the success in use of the PiCam by (Phadnis et al., 2018) and the success in the use of a Web Camera by [2] .

2.2 Software

a. TensorFlow

Machine Learning (ML) is a complex and convoluted discipline due to the designing and implementation of ML models. However, the designing and implementation of ML models is a far less difficult and daunting task than it used to be, due to the research into and development of ML frameworks such as *Google's TensorFlow* that ease the training of models, serving predictions, and the refining of results.

TensorFlow is an open source library for numerical computation and large-scale ML created by the Google Brain team. TensorFlow bundles together a wide range of machine learning and deep learning models and algorithms which can be easily adapted and re-trained for any purpose; from object detection and recognition to Optical Character Recognition (OCR). It uses Python to provide an easy to use front-end API for building applications with the framework, while executing those applications in high-performance C++.

It works by allowing developers to create dataflow graphs. Dataflow graphs are structures that describe how data travels through a graph or a series node where each node in the graph represents a mathematical operation, and each connection or edge between nodes is a tensor or in other words a multidimensional data array. TensorFlow provides all of this for the developer by way of an API built using the Python language.

For this project, both Tensorflow and Tensorflow API will be installed on the windows machine and will be used for training the model and the development of the single capture and inference and real-time capture and inference programmers. They will also be installed on the Raspberry Pi and will be used to run the said programmers.

TensorFlow and TensorFlow API were chosen for backend and frontend development due to the ease of use and the availability of a wide range of pre-built and trained models. Moreover, however, it was chosen due to the success in use demonstrated by [3], [1], and [2] in their research that was explained and evaluated earlier.

b. OpenCV

OpenCV is an open-source computer vision and machine learning software library containing more than 2500 optimized algorithms. It was primarily developed to provide a common infrastructure for CV applications and to accelerate the use of machine perception and is mainly aimed at real-time computer vision applications.

In general OpenCV is used for the processing of real-time images. It contains functions and algorithms for reading and writing images, the detection of faces and their features, detecting shapes such as circles and rectangles, text recognition in images such as number plates, modifying image quality and colors amongst many others. In general it is utilized for the processing of real-time images.

For this project, OpenCV will be used for the capturing of frames, processing the images so they can be inferred upon by Tensorflow and also to output the resulting images with the inference results. It was chosen due to the success and ease of use demonstrated in the research by [1] described earlier.

c. VNC Viewer

Virtual Network Computing (VNC) viewer is a type of software that makes it possible to control another computer over a network connection. Key presses and mouse clicks are transmitted from one computer to another, thus allowing remote control of a desktop, server, or other networked device without being in the same location.

VNC works on a server/client model its operating procedure is as follows: A VNC viewer is installed on the local computer, or client-side, and also the remote computer, or server-side The server side then transmits a matching display of the server-side display to the viewer and interprets commands such as keystrokes and mouse click and movement coming from the client-side and transmits them and carries them out on the remote computer.

VNC Viewer will be used as the main user interface for communication with and working on the Raspberry pi. It was chosen as the interface due to its ease of use and because it removes the need to connect the Rpi to peripheral equipment such as monitor, keyboard and mouse. It will also be useful as after the system is implemented users will need to perform regular maintenance on the system in terms of updates. Furthermore, it would also simplify the task of changing the program that the system is running, for example, changing the program to single image capture and inference to real-time capture and inference.

2.3 Model

The class of model that will be used for this project is class of efficient models called MobileNets. MobileNets are specifically designed CV models for use on hardware that has significant constraints on memory and processing power such as mobile devices and embedded platforms.

MobileNets, developed by [4], are based on an efficient architecture that utilises depth-wise separable convolutions to build streamlined deep neural networks. They introduce two simple hyper-parameters that efficiently trade-off between speed and accuracy.

The hyper-parameters allow the model developer to choose the right sized model for their application determined by the constraints of the problem, for example, memory and processing power. In their research, they perform extensive experiments on resource and accuracy tradeoffs, and from this, they determine that MobileNet shows strong performance compared to other popular models used on ImageNet classification.

They then demonstrate the effectiveness of MobileNets across a range of applications and use cases including object detection, finegrain classification and face attributes amongst others.

In their research, they explain the MobileNet structure as being built on depthwise separable convolutions, as mentioned earlier, except for the first layer which is full convolution. Defining the network in such simple terms they are able to explore different network topologies to find a good network.

For this study, both MobileNet V1 and MobileNet V2 will be trained using the same dataset and compared using the COCO metrics described earlier. Specific metrics will be loss and mAP. Both models will then be deployed on the embedded platform and will then be tested and compared using qualitative and quantitive techniques to determine which model performs best.

Primarily, MobileNet was chosen as the base model due to it being specifically designed for use on embedded platforms and also, because of the success in the research undertaken by [1], [2] and [5]as mentioned earlier.

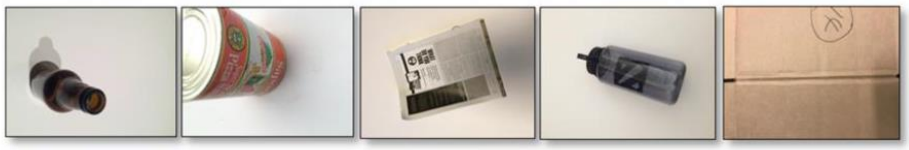


Figure 3 - Samples of TrahNet Data

2.4 Data sets and data preparation

a. Data sets

The data used to train the above models is a combination of two datasets; TrashNet and Images scraped from Google images.

TrashNet dataset contains a total of 2500 images of the five classes of recyclable waste; paper, glass, plastic, metal and cardboard. The images in the dataset consist of photographs of rubbish taken on a white background. The different exposure and lighting selected for each photograph include the variations in the dataset. The dataset is nearly 3.5GB in size with each image being resized to 512x384. The content of the dataset is as follows: 604 images of paper, 601 images of glass, 410 of metal, 482 of plastic and 403 of cardboard. A sample of the data is shown in Figure 3.

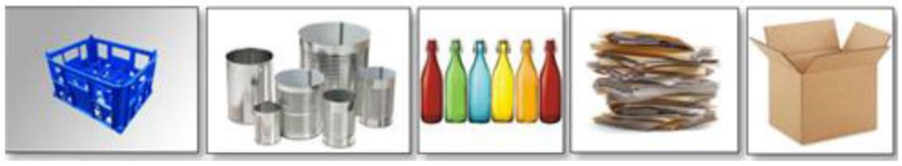


Figure 4 - Google Image Dataset Sample

TrashNet was used as this was the largest repository of data available; however, the dataset does have issues. As explained earlier in the research performed by [1] to get the best results in terms of accuracy the dataset used for training must be as varied as possible such as images of different sizes, containing multiple objects of both objects for classification and objects of no interest. TrashNet, however, contains images of single objects, it is in sequential order, and the images are all of the same sizes.

For this reason, a further 1000 images, 200 of each class, were scraped from google images. A sample of which is shown in Figure 4. To determine the effectiveness of the TrashNet dataset MobileNet V1 was first trained with TrasNet, and then Mobilenet V1 and V2 are trained using the two datasets combined.

b. Data preparation

ANN's use a method of training called supervised learning. In general terms, supervised learning is where the system is given input and output variables to learn how they are related or mapped together. The ultimate goal is to produce an accurate mapping function so when a new input is given; the algorithm can then predict the output. This training method is an iterative process where each time the algorithm makes a prediction, it is corrected or given feedback until it reaches an acceptable performance level. The training data used for supervised learning includes a set of examples with input subjects and desired output, also denoted as the supervisory signal. In an application of supervised learning for image processing, for example, an AI system is provided with labelled pictures of objects in categories such as car or person. After a sufficient amount of iterations, the system should be able to discriminate between and categorize the unlabeled images, at which time the training is complete.

Hence, all the images in the datasets described earlier must be labelled with a description of the object or objects in the image and their location and size in terms of pixels and pixel coordinates. To do this, software call *LabelImg* was used. LabelImg is a graphical image annotation tool, written in Python and uses Qt for its graphical interface.

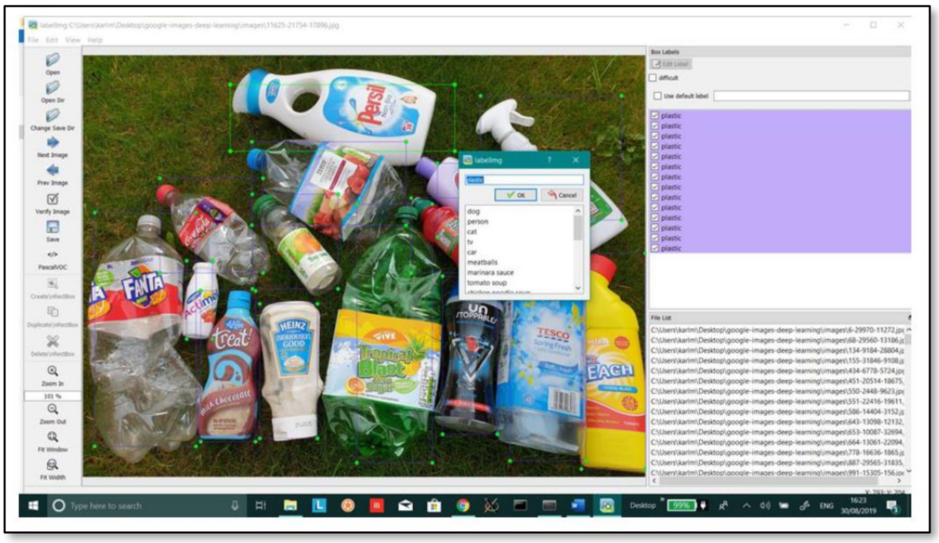


Figure 5 - Example of labelling images in LabelImg

The procedure for preparing and labelling the image data is as follows:

1. TrashNet and Google image datasets are combined and then randomized using a .bat script. This step is performed due to the research findings of [1] where they determined that non-sequential data would give better results in terms of accuracy due to reducing overfitting of the model to the data.
2. The combined dataset is then opened in l*abelImg*, a box is drawn around each object present in each image and the object is annotated with the name of the class it belongs to. An example of which is shown in Figure 5.
3. An XML document is then produced for each image containing the annotations, size and positions of the objects in PASCAL VOC format.
4. The data is then split into training and testing sets in a ratio of 80:20, respectively.
5. The XML documents in the training and testing datasets are then converted to CSV format using a conversion script.
6. Finally, the CSV files are then converted to TFRecord format, using a further script, as this is the format that Tensorflow requires.

3 Results and Discussion

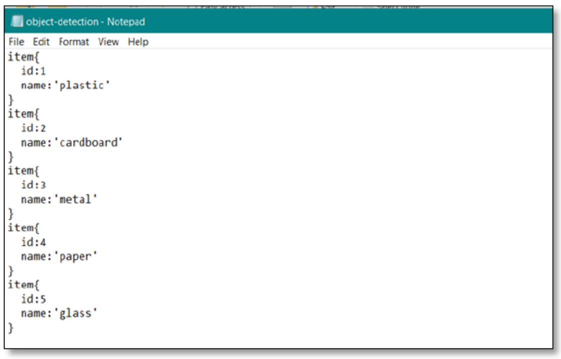
This paragraph summarizes the training procedure, the implementation of the code and then the results of the testing.

3.1 Training

As mentioned previously, the training method employed in this study is transfer learning. Transfer learning is defined by [6] as leveraging knowledge attained from previously learnt domains and applying it to new domains and tasks.

As humans, we have an inherent ability to transfer knowledge across tasks. Hence, what knowledge we acquire while learning about one task, we can utilize this knowledge to solve related tasks. Therefore, the more related the task, the easier it is for us to cross-utilise this knowledge to learn and perform a new task. For example, if we know how to ride a bicycle, it is easier to learn to ride a motorcycle. In this scenario, we do not learn everything from scratch when attempting to learn new aspects or topics. We transfer and leverage our knowledge from what has been learnt previously.

Traditionally, however, conventional ML and DL algorithms have been designed to work in isolation. Meaning, these algorithms are trained to solve specific tasks. Therefore, the models have to be built from scratch once the feature-space distribution changes [6]. Transfer learning is the idea of overcoming the isolated learning paradigm and utilizing the knowledge acquired for one task to solve related ones.



```
item{
  id:1
  name:'plastic'
}
item{
  id:2
  name:'cardboard'
}
item{
  id:3
  name:'metal'
}
item{
  id:4
  name:'paper'
}
item{
  id:5
  name:'glass'
}
```

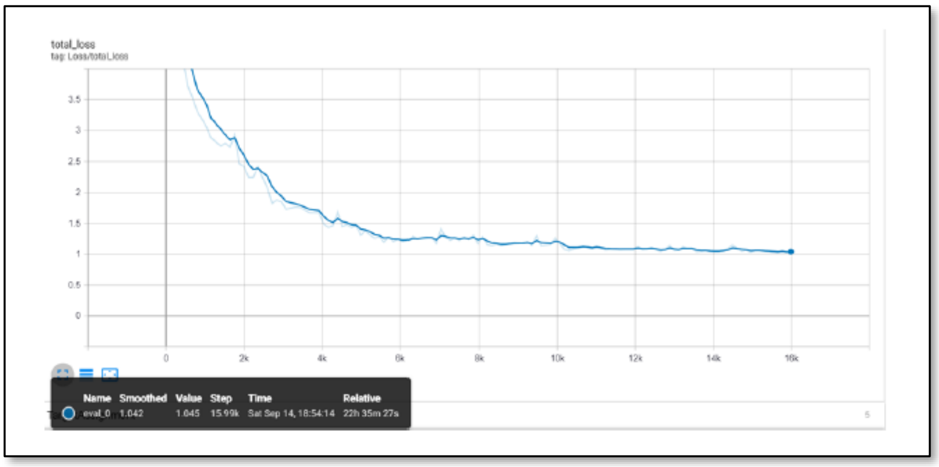
Figure 6 - .pbtxt File Containing Class Labels and Integer IDs

The motivation behind transfer learning is the increasing research into True *Artificial General Intelligence* (AGI). Where researchers and data scientists believe transfer learning is essential to further progress towards AGI (Ling et al., 2015). Furthermore, Andrew Ng, a renowned data scientist, was quoted in an article by [7] saying “After supervised learning , transfer learning will be the next driver of machine learning commercial success”.

As previously mentioned, for this project, TensorFlow API, specifically the modules contained in the API, will be used to perform transfer learning on MobileNet, the procedure is as follows:

1. Download the pre-trained model/models and their associated config files.
2. Create a .pbtxt file containing the class labels and the integer values associated with them. As shown in Figure 6.

3. Modify the config files by:
    a. Changing the number of classes
    b. Adding paths to the directory where the .pbtxt file is stored
    c. Adding paths to the directory where the config file is stored
4. Start the training by running TensorFlow APIs model_main module with command-line arguments to which directory the model and config files are stored and where to store the output graph and checkpoint files.
5. Monitor the training via Tensorboard, specifically the loss
6. When the loss metric stabilises as close too One as possible stop the training. Shown in Figure 7.



Name	Smoothed	Value	Step	Time	Relative
eval_0	1.042	1.045	15.99k	Sat Sep 14, 18:54:14	22h 35m 27s

Figure 7 - Loss Metric from Tensorboard

As mentioned earlier, this process is repeated three times; the first MobileNet V1 is trained with the TrashNet Dataset only and the remaining Two Both V1 and V2 will be trained with a combination of TrashNet and Google images datasets. This is to detemnine, firstly, the effects of data quality with a comparison of MobileNet V1 with both TrashNet and a combination of the two datasets and to compare MobileNet V1 and V2 to determine which performs best in terms of accuracy and speed.

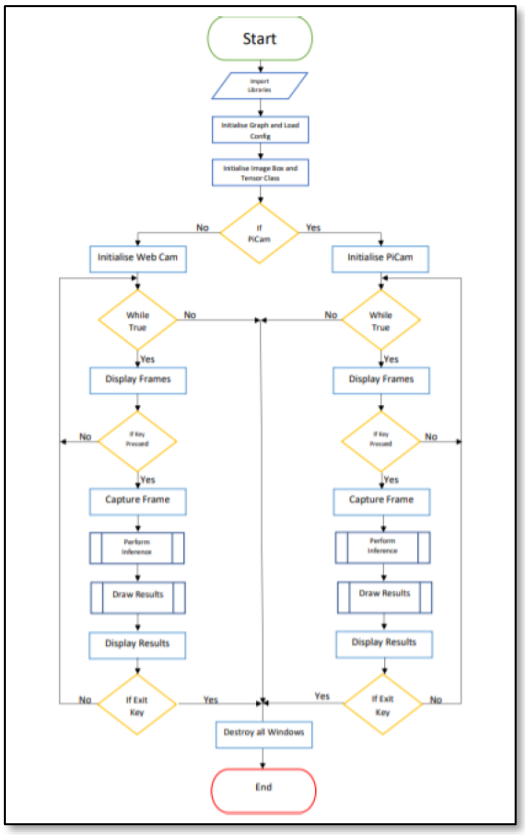


Figure 8 - Single-Capture and Inference Program Design

3.2 Implementation

a. <u>Deployment</u>

In order to deploy the trained models, the top layer or classification, as described above, must first be frozen and the whole inference graph exported for use. This is performed, again, using a module that comes included with Tensorflow API. The module is named *export_inference_graph* and is run with command-line arguments that first describe the input type, in this case an image tensor and then paths to the directories in which the saved chekpoints are stored and the config file as described earlier, and the path to the directory where the new graph will be exported.

As mentioned earlier, the models will be deployed on the windows machine, used for training, and quantitative tests will be performed. The quantitative tests will firstly be used to determine if the use of TrashNet dataset effects the accuracy of the model and secondly too attain a baseline on which the results of deployment testing on the Raspberry Pi will be compared against.

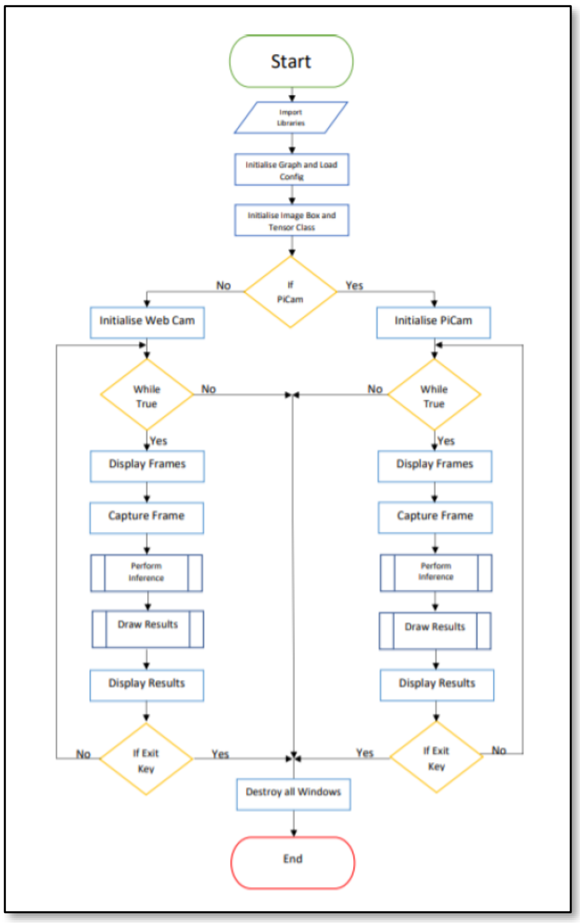


Figure 9 - Real-Time Capture and Inference Program Design

b. Program design

Two programs were designed for this project, the first being single image capture and inference, and the second is real-time capture and inference.

Single Image Capture and Inference

The single-Inference and capture program, shown in Figure 8, is designed to be used with both the PiCam and web camera. When running the program the user can select the web camera for use with the command-line argument --USB, when the program loads it first imports the necessary libraries and then initialises the frozen inference graph and the box and tensor classes. Next, depending on the users choice, either the web camera or the PiCam is initialised. the program then initiates a continuous loop, where the frames from captured from the camera are displayed on the screen. If the capture key is pressed, a frame is captured, the inference is run on that frame, the results are drawn and the results are displayed. If the exit key is then pressed, the program will exit the loop and remove the display windows.

Real-Time Capture and Inference

The real-time capture and inference program, shown in Figure 9, works in nearly exactly the same way. However, the user interaction is removed, therefore, the frame capture and inference is continuous and the results are drawn and displayed in real-time. The program, again, can be exited by pressing the exit key.

3.3 Testing

After deployment, the models were tested for both accuracy and speed. As mentioned earlier, a base-line accuracy and speed were attained for comparison with the results from testing on the Raspberry Pi. For deployment testing on the Raspberry Pi, both accuracy and speed were determined using both the PiCam and web Camera, again, for comparison.

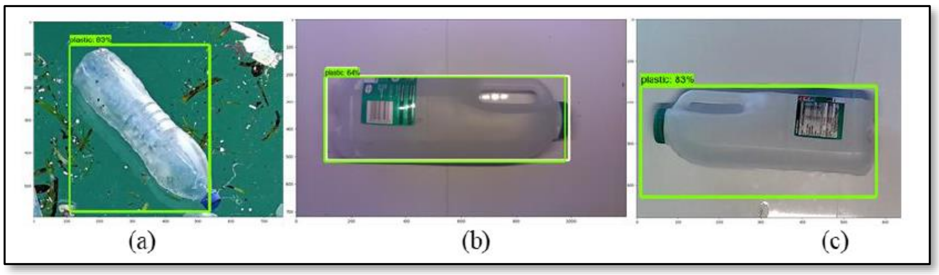


Figure 10 - Sample of Accuracy Image Outputs

a. <u>Accuracy</u>

To determine a base-line accuracy on the windows machine 50 images, 10 of each class, were scraped from google images as per the procedure explained earlier. They were then run through a dedicated accuracy-test program. The accuracy-test program works in nearly the same way as the single-inference and capture program. However, there is no user interaction, the program utilizes a for loop which iterates through each of the fifty images and outputs the image with inference and confidence score, shown in Figure 10, and also writes the inference results and confidence scores to a txt file for processing.

For deployment accuracy, 50 images, again ten of each class, were taken with both the Picam and web camera and were run through the same program as the baseline testing and the results were outputted in the same way. Sample results for the PiCam and web camera are shown in Figure 10 (b) and (c) respectively.

To determine the accuracy of the models, the accuracy of each class was calculated using the following expression, where the accepted value is the total number of possible correct results and the error is the total number of incorrect results. Therefore it holds:

$$\% \; accuracy = [\; (\; accepted\_value - error \;) \; / \; accepted\_value \;] \; x \; 100\%$$

The average accuracy across each class was then calculated using the following expression, where $\bar{x}$ is the average accuracy, n is the number of terms and x_i is the value of each term in the list of numbers being averaged.

$$\bar{x} = 1 / n \;\; \Sigma_{i=1,n} \, x_i$$

b. Speed

To determine a base-line speed for the models on the windows machine an item of one of the classes was placed in the view of the windows machine's camera and the base-line speed script was run. The script iterates through 500 ticks or 500 iterations were inference, drawing results and displaying results takes place, and at the end of each tick the calculated *Frames Per Second* (FPS) is outputted to a text file for processing. For deployment accuracy, the above process was repeated on the raspberry pi using both the Picam and Web camera for comparison.

The FPS was calculated using the following expression, where *FPS* is the calculated speed, n is the number of frames to be considered, 1 in our case, t_1 is the time at the beginning of the iteration and t_2 is the time at the end of the iteration.

$$FPS = n \; / \; (\; t_2 - t_1 \;)$$

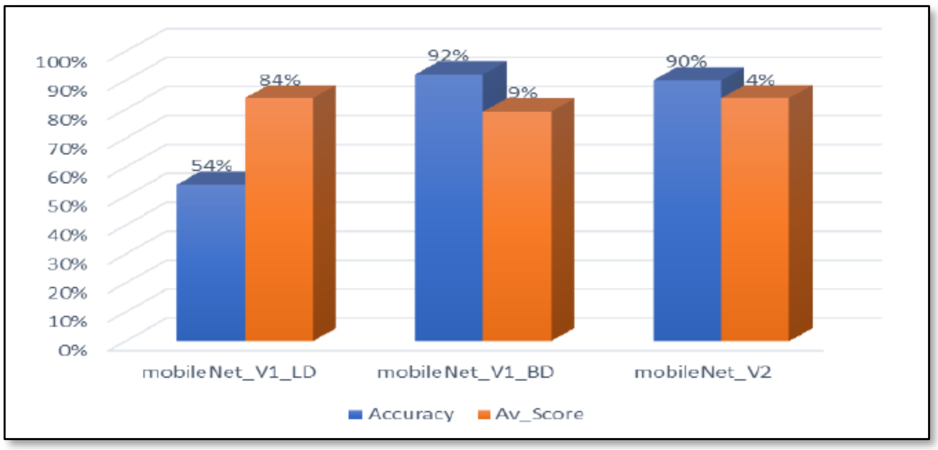


Figure 11 - Base-Line Accuracy Results

3.4 Results

a. <u>Base-Line Accuracy</u>

The base-Line accuracy results for MobileNet V1 with TrashNet only, MobileNet V1 with a combination of both datasets and MobileNet V2, shown in Figure 11, show an accuracy of 54%, 92% and 90% respectively.

The results for V1 with TrashNet and with a combination of both datasets clearly illustrate that the use of TrashNet alone has a significant impact in terms of accuracy, with a drop of nearly 40%. Therefore, the determination that data must be non-sequential and be varied in terms of sizes in the research by [1] is correct and due to this MobileNet V1 and V2 were both trained on a combination of the two datasets as to increase accuracy.

The comparison of both MobileNet models in terms of accuracy, however, does not show any significant difference in accuracy. Therefore, both models were deployed on the Raspberry Pi to determine if deploying the models on the Raspberry Pi using the PiCam and web camera has an effect on the accuracy of the models.

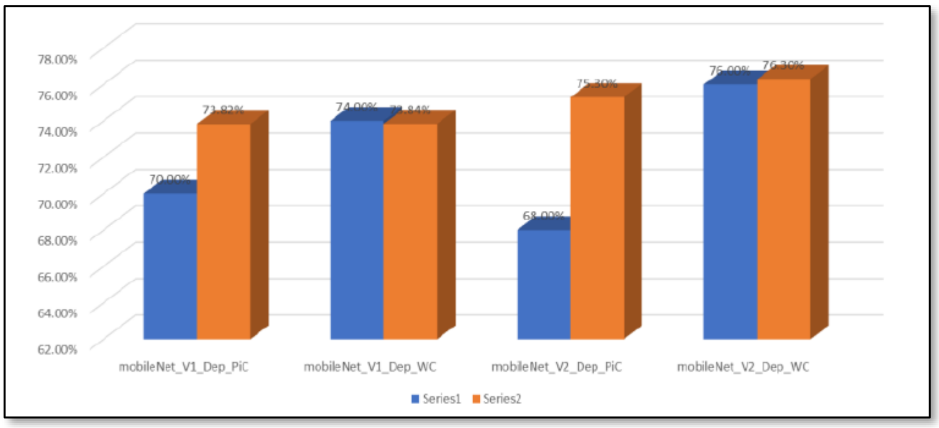


Figure 12 - Deployment Accuracy Results

b. Deployment Accuracy with PiCam and Web Camera Accuracy

The deployment accuracy results, shown in Figure 12, show the accuracy of both V1 and V2 using the PiCam and web Camera. The results of V1 show an accuracy of 70% with the PiCam and 74% using the web camera. The results for V2 show an accuracy of 68% with the PiCam and 76% with the web Camera.

Again, the differences in accuracy between the models are minimal; therefore, the models perform equally well, and either would be suitable for use in the system.

The results in comparison with the base-line however, show a drop in accuracy of around 20% when both V1 and V2 are deployed on the Raspberry Pi. However, this drop in the accuracy is not likely to be due to the Raspberry Pi. It is more likely that the drop in the accuracy is due to the image quality being impacted by a combination of varying light intensities and the focusing of the lens.

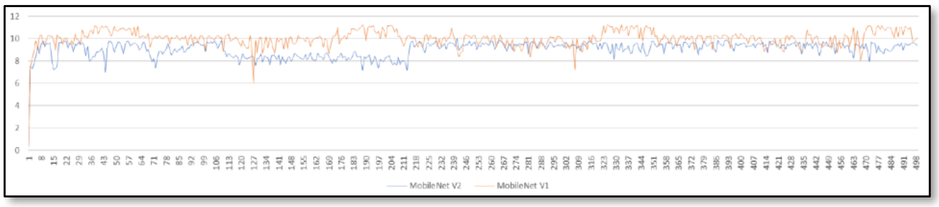


Figure 13 - Base-Line Speed Results

c. Base-Line Speed

As above, a base-line speed was taken on both V1 and V2 using the windows machine they were trained on. The baseline results, shown in Figure 13, show that both

V1 and V2 both models achieved an average speed of between nine and ten frames per second.

The results of the speed test, again, show no significant difference between the models. Therefore, both models were deployed on the Raspberry Pi and tested again to determine if deployment on the Pi adversely affects the speed of either model more than the other and to determine if there is any difference in speeds when using the PiCam compared to using the web camera. Furthermore, a speed of around ten frames per second would be adequate to be used for the intelligent retrieval of recyclables.

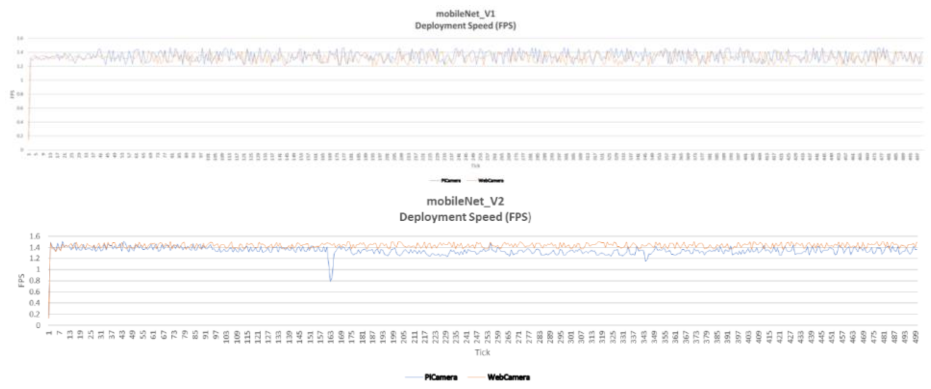


Figure 14 - Speed Results for MobileNet V1 and V2 9top and bottom panels, respectively)

d. <u>Deployment Speed</u>

The deployments speed results for V1 and V2, shown in Figure 14, both show an average speed of around 1.4 frames per second, using both the PiCam and web camera. Thus showing, again, no difference in speed between the two models and, surprisingly, no difference in speed when using either the Picam and web camera.

The speed comparison results between the cameras was expected to show a significant difference in terms of speed. The difference was expected due to the difference in the resolution of the cameras. With the PiCam having a resolution of 5MP and the web camera having a resolution of 12MP, it was expected that the web camera images would take significantly longer to process.

However, the deployment speed of the models on the Raspberry pi shows a significant drop in speed from 10 frames per second to around 1.4 FPS. This, again, is not believed to be due to the models themselves but due to the processing power of the Raspberry Pi.

The drop in speed will, however, make the system virtually unusable for the intelligent retrieval of recyclable objects.

e. <u>Cost</u>

The other consideration of the system was its cost as the lower the cost of the system the higher the likelihood of the system being put to use is significantly higher. The cost breakdown and total is shown in Table 1. With a total cost of the system being £ 67.81 (≈ $ 85), the system is undoubtedly cheap and would, therefore, would increase the likelihood of the system being adopted for use [8-10].

Table 1. Cost Breakdown and Total.

Hardware	Cost (£)	Software	Cost (£)
Raspberry Pi 3 Model B+	37.85	TensorFlow/API	Free and Open Source
PiCam Module	11.99	VNC Viewer	Free and Open Source
Web Camera	6.99	LabelImg	Free and Open Source
RPi Case and Fan	7.99		
RPi Heat Sinks	2.99	**TOTAL COST**	**67.81**

The accuracy and cost of the system is perfectly adequate to be used for both single capture and real-time capture. However, more investigation into optical sensors must be carried out with a focus on increasing the quality of the captured images to increase the inference accuracy.

The speed of the system, however, is an issue. Therefore, the system would not be suitable for the intelligent retrieval of recyclable objects. Therefore, more research must be carried out, and more powerful platforms should be investigated, as to attempt to increase the speed of the system.

4 Conclusion

The world in which we live is increasingly becoming more and more polluted by man made waste. There have been great strides made recently in the retrieval of this waste, however, this will increase the pressure on the already inefficient material recovery facilities. Therefore, more efforts must be made to increase the efficiency and lower the cost of the MRFs.

The research into CV, more specifically embedded CV could be the answer to this and many other related problems. Especially now, due to the advances in technologies and software making it more acsesible, easier to use and more robust.

The main aims of this project was to build a cheap universal embedded computer vision system that could be used in both the MRF for sorting and in the field for the intelligent retrieval of waste. As a whole the aims have been achieved, however, the speed of the system would make it virtually impossible to use for intelligent retrieval of waste. Therefore, more research with a focus on the embedded platform must be undertaken.

ACKNOWLEDGMENTS

This work was presented in dissertation form in fulfilment of the requirements for the MSc in Robotics Engineering for the student Karl Myers under the supervision of E.L. Secco from the Robotics Laboratory, School of Mathematics, Computer Science and Engineering, Liverpool Hope University

References

1. Phadnis , R., Mishra, J. & Bendale , S., 2018. Objects Talk-Object Detection and Pattern Tracking. Coimbatore, India , IEEE.
2. Guennouni, S., Ahaitouf, A. & Mansouri, A., 2014. Multiple Object Detection Using OpenCV on an Embedded Platform. Tetouan, Morocco, IEEE.
3. Swain , M., Dhariwal , S. & Kumar, G., 2018. A Python (openCV) Based Automatic Tool for Parasitemia Calculation in Peripheral Blood Smear. International Conference on Inteligent Circuits and Systems, pp. 445-448.
4. Howard , A. G. et al., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications , s.l.: arXiv.
5. Boyko, N., Basystiuk, O. & Shakhovska, N., 2018. Performance Evaluation and Comparison of Softaware for Face Recognition, Based on Dlib and OpenCV Library. Lviv, IEEE.
6. Pan, S. J. & Yang, Q., 2010. A Survey on Transfer Learning. IEEE Transactions on Knowledge and Data Engineering , 22(10), pp. 1345-1359.
7. Malisiewicz, T., 2017. Nuts and Bolts of Building Deep Learning Applications: Ng @ NIPS2016. [Online] Available at: https://www.datasciencecentral.com/profiles/blogs/nuts-and-bolts-of-building-deep-learning-applications-ng-nips2016 [Accessed 4 October 2019].
8. D. McHugh, N. Buckley, E.L. Secco, A low cost visual sensor for gesture recognition via AI CNNS, Intelligent Systems Conference (IntelliSys) 2020, Amsterdam, The Netherlands, *accepted*
9. E.L. Secco, R. Abdulrahman, I. Felmeri, A.K. Nagar, Development of Cost-effective Endurance Test Rig with Integrated Algorithm for Safety, Soft Computing for Problem Solving, Advances in Intelligent Systems and Computing, 1139, chapter 14, Springer, DOI: 10.1007/978-981-15-3287-0_14
10. E.L. Secco, C. Moutschen, A Soft Anthropomorphic & Tactile Fingertip for Low-Cost Prosthetic & Robotic Applications, EAI Transactions on Pervasive Health and Technology, 4, 14, 2018