The role of object recognition in inventory management

Investigating the process of developing a Neural Network inventory management system for packaging

Tudor Ventuneac

University of Limerick, 23390131@studentmail.ul.ie

Ivan Novik

University of Limerick,

This paper presents a novel approach to the detection and classification of bagged items using Edge Impulse, an embedded machine learning platform. The proposed system utilizes a custom-trained neural network model to scan and identify the contents of bags in real-time. By integrating advanced signal processing techniques with machine learning algorithms, the system achieves robust performance even in resource-constrained edge devices. The methodology involves data collection, preprocessing, feature extraction, model training, and deployment onto microcontrollers. Experimental results demonstrate the efficacy of the developed system in accurately detecting the presence of items within bags and classifying them into predefined categories. The evaluation showcases high accuracy rates across various bag types and contents, highlighting the system's adaptability and effectiveness in real-world scenarios. This research contributes to the advancement of AI-enabled edge computing applications, offering promising implications for automated inventory management, security screening, and retail optimization.

CCS CONCEPTS • Machine learning • Supervised learning • Object detection

Additional Keywords and Phrases: Deep-learning, FOMO (Faster Objects, More Objects), Edge Impulse, Arduino Tiny ML

ACM Reference Format:
First Author's Name, Initials, and Last Name, Second Author's Name, Initials, and Last Name, and Third Author's Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

1 INTRODUCTION

This project aims to determine how machine learning algorithms can be used in real-life practical applications to detect packaging status for small stationery like biros, pencils, crayons, etc. While starting with a complex model we encountered performance issues and a low detection rate, we ultimately used a gradual training and learning model moving from simple to complex model with significant detection rate improvement. As part of this project, we realized there are real issues [ref] with automated detection and classification of small objects, and several improvements were made to compensate for such known issues.

As a result, we set out to develop and implement an interactive system capable of detecting if small stationary items are bagged or not, and if the detected bagged ones contain the correct inventory. The premises and the full steps employed for training the model for small stationary items detection is described in the next sections.

2 Development

a) Overview of the system we built

Our object detection system was implemented with the help of Edge Impulse, an embedded machine learning platform [ref edge]. With the help of Edge Impulse, we were able to create and design an object detection system capable of recognizing bagged and unbagged inventory, as well as multiple pieces of inventory within each bag.

We originally started with a complex model (model A) with multiple labels, but because of a lower detection rate we had to switch to a simpler model (model A) and gradually train in it and scale it up to accommodate multiple labels. Feeding the system our test data, which was taken in the picture format by hand, we were able to achieve a significantly better accurate object detection rate for model B compared with model A. Model B also performed well in live classification. Model B was then deployed to the Arduino BLE 33 Sense board and successfully utilized for live model testing through the OV7670 camera.

b) Choice of algorithm

In our process of choosing the best-suited algorithm for our system, we encountered many errors and setbacks which are seen through the development of our system. Initially, we used the FOMO algorithm for its ability to detect objects with centroids, which we believed would be better suited for the small size of our objects. As well as this, the performance of the FOMO algorithm was much better suited for the Arduino Nano 33 BLE Sense due to its much better RAM management when compared to the MobileNetV2 SSD FPN algorithm[ref]. Finally, FOMO was much better suited for our project due to all our objects being similar in size and shape.

However, what we failed to realize was that FOMO was not suited for recognizing objects that were too close. This is because the centroids would overlap and not be recognized as multiple objects if they were too close together, which presented itself as a problem to us in some of our test data.

c) Training process, training data

For our system's training, we decided to use a phone camera to take JPG pictures of the stationery [Fig. 1] and non-stationary items, both in a bagged [Fig. 2] and unbagged state [Fig. 3]. Summary of initial conditions for the model is provided below

- each object was individually photographed – as a precondition to build our starting simple object detection model
- All the objects were photographed on a neutral cardboard background
- All bagged objects used plastic packaging, of three different colors and four different sizes.
- Additionally, for the bagged items, each bag had a white piece of paper as additional bag background.

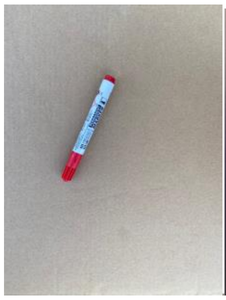


Fig. 1

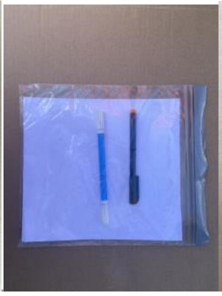


Fig. 2

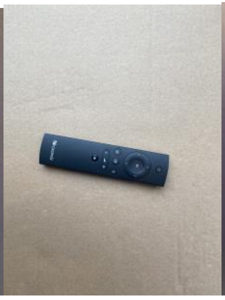


Fig. 3

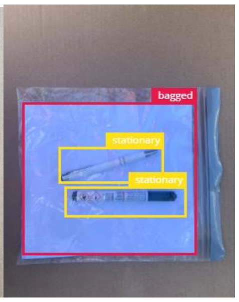


Fig. 4

The following considerations were considered when selecting the initial conditions for the model

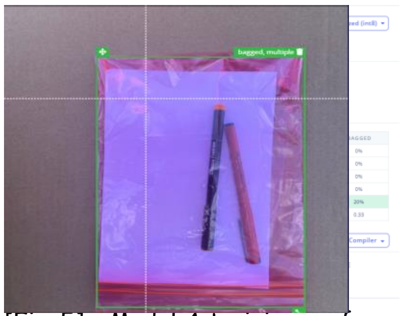
- Photographing individual items allowed us to build and label an inventory of items, very useful for later detection and classification
- The neutral cardboard background was expected to help with detection of bagged and unbagged objects.
- The white paper in the plastic bags enhanced the detection process of the bagged items. This acted as the backdrop for the bagged items, reducing the chance of our system confusing the cardboard background, because of the see-through / transparent bag.

<u>Initial data set used for training and testing and steps for model A (complex one)</u>

1. We took around 20 individual pictures of the unbagged stationery items (biros, crayons, pencils, markers, etc.) and non-stationary items (sunglasses, coins, phone etc.). [Fig. 1] [Fig. 3]
2. The labeling was done for individual items for both unbagged and bagged items. For multiple labeling, an example is shown in [Fig. 4]
3. These were then uploaded to Edge Impulse and resized to a dimension 96x96 (due to FOMO's square resolution restriction and to lower the ram usage of our Nano 33 BLE Sense Lite) [ref].
4. We then initially decided to use grayscale as our color depth for the training data.
5. With these processed images, we then trained our initial system for 100 training cycles and at a learning rate of 0.01 using FOMO.

<u>Initial results & issues for model A (complex one)</u>

Screenshots for low 50% accuracy results [fig 5]



[Fig. 5] : Model A training performance

[Fig. 6] : Incorrect implementation of multiple object labelling

For model A, we realized some of the earlier mistakes done as follows
- The model was too complex to start with, and we expected accurate results too fast/too early
- Overlapping object labels
- Incorrect implementation of multiple labelling, condensing multiple labels into a single label [Fig 6]
- Learning rate was too high for the initial model
- Usage of grayscale color depth was wrong considering the increased color variation in our object sample

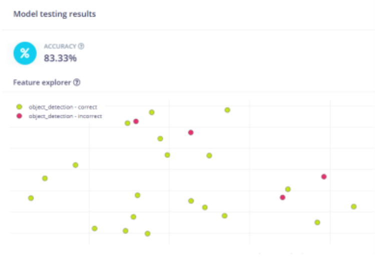
<u>Building Model B based on lessons learnt from Model A</u>

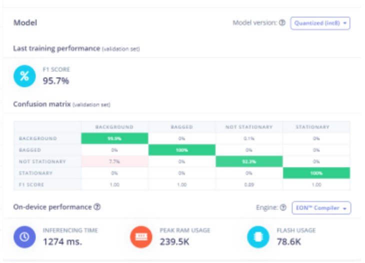
The identification of the above issues allowed us to implement a second Model B, with a better object detection rate and a higher model testing accuracy [Fig 7, fig 8]

- We started with a simpler Model B, with less labels as a start. This was because Model A contained too many labels at the start.
- Due to the close proximity of some of our objects when bagged, it led to overlapping object labels as the FOMO algorithm we chose used the object center to detect the objects instead of labelling boxes. This was something we were unable to fully resolve. However, we improved the detection of small objects by increasing the object weight of our system from 100 to 500.
- Model B was first trained for a high training performance score on just the stationary and non-stationary objects. Then we gradually added more training data into our system, adding more object labels such as bagged and unbagged, (as opposed to adding all labels variations from the start, as done in Model A)
- The learning rate of Model B was also configured as we realized that the learning rate of Model A was too fast for the initial training of the data. This resulted in a change from a 0.1 learning rate to a 0.001 learning rate. [fig]
- Our object detection of small objects was further improved when we switched from the grayscale colour depth to the RGB colour depth. This led to a significant increase in our system's model accuracy. This was due to Grayscale having approximately 3x less features than RGB [https://product.corel.com/help/CorelDRAW/540240626/Main/EN/Doc/wwhelp/wwhimpl/common/html/wwhelp.htm?context=CorelDRAW_Help&file=CorelDRAW-Understanding-color-depth.html], resulting in a smaller image and making it harder for our system to accurately detect our objects

Results for Model B

Due to the above implementations, Model B was much more accurate and refined than Model A as seen in the results [fig 7] [fig 8].





	BACKGROUND	BAGGED	NOT STATIONARY	STATIONARY
BACKGROUND	99.9%	0%	0.1%	0%
BAGGED	0%	100%	0%	0%
NOT STATIONARY	7.7%	0%	92.3%	0%
STATIONARY	0%	0%	0%	100%
F1 SCORE	1.00	1.00	0.89	1.00

[fig 7] : Model B test data results [fig 8] : Model B training performance

Feature extraction

The following features were chosen for object detection -
- Quantity of objects
- Type of objects (stationary and non-stationary)

-State of objects (unbagged or bagged)

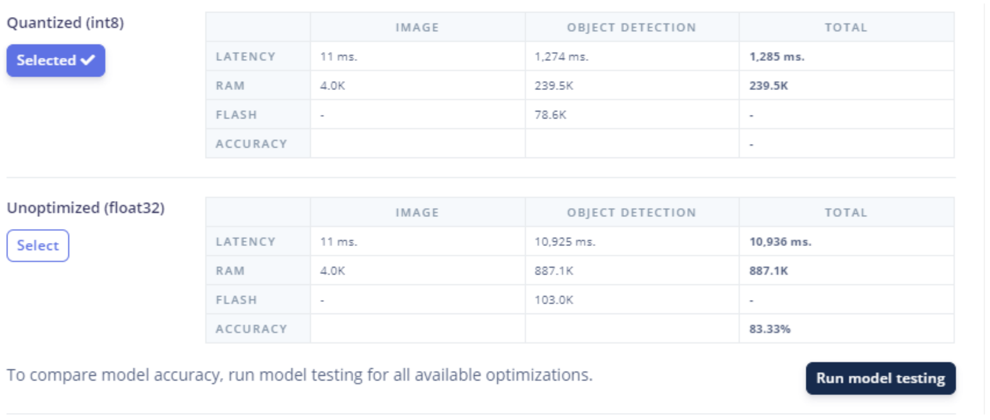
These features were chosen to support applicability to real-world problems with using machine learning for inventory management and object classification. The quantity of the objects would help us realize if there was any missing inventory, when applied to a real-world application. The type of objects would further optimize our system, when implemented into a real-world scenario, by allowing for identification of non-stationary objects. The status of the objects is useful to automatically detect if the product is ready for shipping, when applied to a real-world scenario.

Development of output

The development of our system was done using the Arduino BLE 33 Sense Lite, along with the Tiny Machine Learning Shield and the OV7675 camera module. [Fig 9]

Before we could deploy it to the Arduino library, we chose to compile our system using the int8 quantization process due to the optimized ram usage on our BLE 33 Sense Lite, when compared to the float32 unoptimized process. The total ram usage for the int8 process was around 239.5k of ram, compared to around 887.1K of ram on the float32 process. [Fig 10]

After this was done, our system was uploaded to the Arduino IDE and compiled, letting us run our system on the Arduino IDE.



Quantized (int8)

	IMAGE	OBJECT DETECTION	TOTAL
LATENCY	11 ms.	1,274 ms.	1,285 ms.
RAM	4.0K	239.5K	239.5K
FLASH	-	78.6K	-
ACCURACY			-

Unoptimized (float32)

	IMAGE	OBJECT DETECTION	TOTAL
LATENCY	11 ms.	10,925 ms.	10,936 ms.
RAM	4.0K	887.1K	887.1K
FLASH	-	103.0K	-
ACCURACY			83.33%

To compare model accuracy, run model testing for all available optimizations.

[Fig 10] : Comparison of int8 and float32 optimizations on our model

What we learnt from early models and then refined models

the neutral cardboard background to be worse for object detection when using FOMO as the algorithm worked better when it was fed a variety of different backgrounds to train on[ref] -> at end (looking back)

d) Conclusion and Future work

<u>What was learned?</u>
-The workings of the FOMO algorithm
-The importance of creating an initial simple system and scaling it
-The process of overfitting a model and how to solve it

Understanding how the FOMO algorithm was a big lesson for us throughout this project. The fact that FOMO does not output bounding boxes when compared to other object detection algorithms was immensely useful to us due to the proximity of our objects, as well as their small size. Instead, we learnt that the FOMO algorithm trains on the centroids of objects, making it much easier to count objects that are close. Another fact we learnt was the workings of the FOMO algorithm in terms of its neural network architecture and the fact that it consists of numerous convolutional layers. We learnt that each layer creates a lower-resolution image of the previous layer. However, this is where FOMO differs from the MobileNet SSD algorithm, FOMO allows for flexible square resolutions which makes it very useful even if you have larger images that need to be studied. Object weighting was another part of the FOMO algorithm that we learnt about. Object weighting is the ratio of objects to background cells in the labelled data. We tried multiple object weightings for our system to see if the model would focus even more on object detection as our objects were relatively rare within our images. This allowed us to further optimize the FOMO algorithm and tweak it based on our specific needs.

The importance of creating an initial simple system and working our way up from there was something we drastically underestimated, as seen in the drastic difference between Model A and Model B performance results. Our biggest mistake here was training our initial system on a too complicated set of training data, in terms of too many object labels. This resulted in a poorly trained initial system that performed terribly on model testing. As a result, we built Model B and first fed it only images consisting of a single object label, stationery. From there, we added more test data, adding another object label in the form of non-stationary. From here, we began to implement our third and fourth object label, bagged and unbagged. Since Model B was already trained on individual object labels, it was a lot more competent in recognizing objects after it was fully trained.

Finally, the overfitting of a model was something we saw throughout the process when retraining our system to recognize additional object labels. We were previously not too familiar with the concept of overfitting and did not initially realize our model was overfitted. We only realized that our model was overfitted when we compared the model training performance to the model accuracy on the test data. We observed a large difference between the two, prompting us to investigate the cause for this. As a result, we reduced the number of training cycles, reduced the learning rate of our model and enabled data augmentation for our model to reduce the overfitting. We decided to reduce the number of training cycles and the learning rate to counter the model not being able to generalize and fitting too closely to the training dataset. We also enabled data augmentation for the retraining of our model as it would increase the variations of our already existing data, helping the model generalize better.

As a result of all these learning processes and new-found knowledge, we feel a lot more confident and comfortable attempting to build an object detection model as we're now more familiar with the customization of object detection algorithms to fit our specific needs.

What worked and didn't work?

We initially set out to build a system capable of recognizing inventory in a particular state, in our case bagged and unbagged, as well as recognizing multiple objects within our inventory dataset, stationary and non-stationary. This was accomplished with Model B, leading to a high-accuracy model that correctly recognized the state of our inventory.

However, there were also a few things that didn't work as well as we expected them to.
The detection of closely placed objects was one of these things. Even though our model achieved relatively high object detection accuracy, it still sometimes mislabeled objects and sometimes did not even label them at all if they were too close. Initially, we did not realize that editing the ratio of the input image to the heatmap of where our objects were would give us a significant improvement in the detection of small objects. Looking back, we should have set this to 1:1, giving us pixel-level segmentation and the ability to count more small objects. The overlapping of the object centroids also contributed to this issue. Reflecting on these issues, the implementation of a higher resolution heat map would have allowed for a more visible image that our model could scan and extract information from.

Future Work

Our model could be developed further in multiple ways in terms of extracted features and object detection accuracy. We could add another object label to detect if the packaging itself is breached or not, training the model on a dataset consisting of breached packaging. This would improve the real-world applicability of our system, allowing for a safer and more secure system in terms of keeping the inventory sealed and away from any contaminants. Furthermore, our model could incorporate a larger variety of objects to be detected and classified such as books and magazines due to their bigger size for online shops. A larger data set could also be implemented into our system, allowing for greater object detection accuracy and variety. We could also train our system on more real-time object detection algorithms such as YOLO and Fast R-CNN. This would have the added benefit of covering differently sized objects and allowing for spacial detection when compared to FOMO.