uc3m | Universidad Carlos III de Madrid

Master Degree in Big Data Analytics

2022-2023

Master Thesis

"Garbage Detection with YOLOv5 for City Streets"

Emilie Naples

Carlos Ruiz Mora

Nerea Luis Mingueza

Madrid, 2023

SUMMARY

In this research paper, the design and implementation of a garbage detector as both a user interface (UI) and system software for installation on edge devices is presented in detail. The system is constructed using the YOLOv5, a pre-trained, state-of-the-art, single-stage neural network used for real-time object detection. Developed by UltraAnalytics, and with only a single pass through the neural network, the network's algorithm is able to efficiently classify and detect objects it is trained to identify. The project leverages this pre-trained YOLOv5 Nano version of the model from the UltraAnalytics GitHub Page and adapts it to a custom dataset specific to garbage for garbage detection.

A series of multiple custom datasets are highlighted in this paper to address the convergence problem that was encountered during research. This problem was accounted for after model construction and during the training process when the model metrics did not reach stability at the beginning of this study. The convergence issue was related to inconsistent data (i.e. what kinds of images are considered trash, and which are not). Each dataset served as a small piece of the trial and error process to see which "trash types" would be consistently recognized by the constructed model, thus allowing for convergence and stabilized metrics during training for final model deployment.

Data gathering, data processing, model training, and model testing phases of the garbage detection software were facilitated by an MLOps pipeline developed previously within a company setting where the student participated as an intern. The MLOps pipeline optimizes the model for deployment that, when integrated with cameras, will capture image metadata from video streams and transmit this data back to a centralized database for efficient and better-optimized trash pick-up. The project combines AI-powered object detection and video image stream location metadata to promote environmental sustainability and trash collection.

Keywords: garbage detection, object detection, machine learning, computer vision, waste management, environmental sustainability

DEDICATION

I would like to thank my tutor and director of the Master in Big Data Analytics at the Universidad Carlos III de Madrid, **Carlos Ruiz Mora**, my tutor at Sngular, **Nerea Luis Mingueza**, and my master degree coordinator, **María Galán Luengo**. Two years ago, at the beginning of the process in the decision to study this master, Carlos was the first to tell me that I was capable of such an accomplishment, even with no background in programming. Had he warned me about not having the proper background, I would not be where I am today defending this thesis after a long journey. He believed in me and encouraged me despite the fact. I owe the same thanks to María for her patience, understanding, and the incredible amount of help she provided along the way as I faced several complex situations throughout the 2-year journey. I will never forget her kindness, willingness, and enthusiasm in helping me no matter what. Lastly, I owe many thanks to Nerea Luis Mingueza, my supervisor and tutor during my internship at Sngular where this master thesis was developed. She will always be the person that gave me my very first job opportunity that began my career in Data Science. She gave me a chance that led me to where I am today in the defense of this thesis with many career prospects ahead of me. Thank you to the three of you.

CONTENTS

1. INTRODUCTION

1.1. Background and Motivation

Computer Vision techniques such as object detection are not a recent advancement in the field of Artificial Intelligence. However, the refinement of such techniques including increased accuracy and growing variety of applications certainly *are* a recent breakthrough [1] [2]. Such development can be attributed to the emergence of Deep Learning and Convolutional Neural Networks (CNNs). Machine Learning Frameworks such as TensorFlow, PyTorch, and architectures like YOLO, SSD, and Faster R-CNN have played an equally important role in such progress, which has in turn caused the development of custom object detection models [3] with applications in natural and urban environments to become more feasible than ever. It is not an understatement that the environment as a theme itself has also become a growing concern among the general public in the last decade [4] as people, organizations, and companies develop more of an awareness and desire to care for planet Earth and their neighborhood surroundings.

An increase in popularity for "smart cities" and the reliance on automation are also large contributing factors to the desire and the necessity to identify and effectively manage waste [5]. Computer Vision and advancements in algorithms such as neural networks are a possible solution to waste management concerns [6]. Neural networks like YOLO (You Only Look Once), are the algorithms that facilitate real-time object detection [7]. What if it were also possible to develop such software in a way that trash location metadata on city streets around the world could be sent back in real-time to a centralized database in order to optimize trash detection, clean-up crew deployment, and beyond?

1.2. Problem Statement

Despite the advancements in object detection algorithms, there is still a lack of efficient systems designed for garbage detection [6]. Most applications for these algorithms are geared toward facial recognition, pedestrian detection, and more general object detection tasks [8]. This research paper proposes a potential solution to the lack of existing garbage detection systems by constructing a machine learning model trained to identify garbage. The model will be downloaded as software onto cameras and edge-devices for efficient trash-pick up on city streets.

Developed within the company where the student conducted this research project, the real-time garbage detection problem involves a pipeline that uses the YOLOv5 neural network architecture to iterate through various custom datasets of between 1,000 and 1,600 garbage images on city streets for model training. Compared to the image quantity used for model training in other professional settings, these numbers are very low, but

used anyway due to tutor instruction. A large problem encountered along the way was one of convergence.

For a good portion of the research period in which the project was carried out, the model failed to converge for some time. The issue was the result of many inconsistencies in the images used to train the garbage detection model, which neglected to include images according to only one definition of trash. The machine learning model trained to detect trash uses the YOLOv5 architecture, as previously mentioned, and was trained on 17 different trial and error datasets, each consisting of 1,000 - 1,600 labeled images of trash on city streets. Each dataset attempted to solve the model convergence problem and improve performance metrics. The goal of the project was to construct an effective garbage detection machine learning model to be used on edge-devices and as a web application user interface for clients of Sngular.

The challenges associated with garbage detection are unique:

1. **Variety in Appearance;** Garbage items exhibit a range of appearances including shapes, colors, sizes and even stages of decay. Take a banana peel as an example; its appearance can vary depending on its decomposition stage.

2. **Obstacles** In real life situations garbage is usually blocked or hidden, which can make its detection more challenging.

3. **Background** Garbage tends to be surrounded by other garbage AND non-garbage items, especially in urban environments like city streets. This adds complexity to the task of detecting garbage.

4. **Limited Data Availability;** This posed a challenge for the project. Machine learning models usually require large datasets for training. While there are well-labeled datasets for other object detection tasks, finding labeled datasets specifically for garbage detection can be difficult. For instance, YOLOv5 was trained on the COCO Dataset (containing 80 object classes) which contains over 200,000 images, but does not include trash images. Although there are other datasets that do contain trash, they are limited.

5. **Incentive:** Garbage detection is important, especially because of current-day environmental concerns. However, it still might not attract as much investment or attention as other applications that have greater potential for profitability.

1.3. Objectives of the Study

The primary objectives of this study are:

- To use version 5 of the YOLO algorithm and adapt it to efficient garbage detection in real-time.
- To solve the model convergence problem during model training.

- To construct a user interface so that clients and other users can interact with the model as a fine-tuned garbage detector.
- To explore the implementation of the constructed garbage detection system for deployment on edge-devices and cameras to be used for optimized trash-pick up and location on city streets.
- To compare the proposed solution with existing systems in terms of its overall objective for deployment on cameras and edge-devices.

2. LITERATURE REVIEW

2.1. Object Detection in Computer Vision

Object Detection is a branch of Computer Vision that involves the identification of objects in an image or video [9]. When an**object detection** program locates an object, it creates a bounding box around it and then assigns a label to classify it. However, **image classification** differs from object detection as image classification involves labeling the image itself, whereas in object detection, the software learns to locate and categorize objects *within* an image based on pre-existing labels [9].

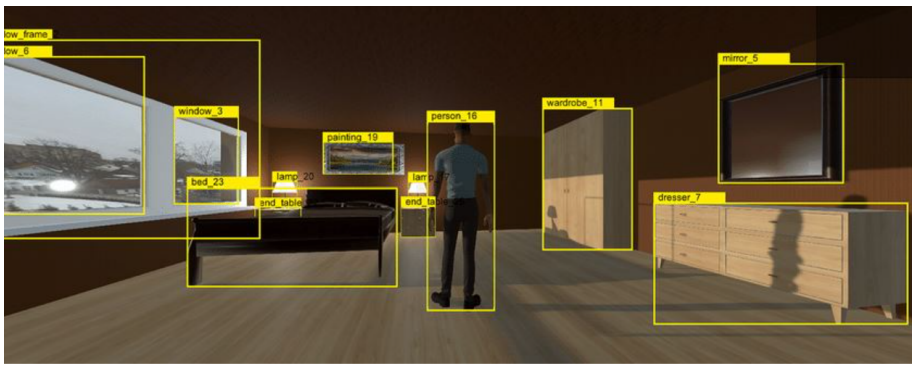


Fig. 2.1. Bounding boxes. Image taken from [10].

For instance, an image or video stream includes trash bags. In this case, an object detection model would identify the positions within the image where the trash bags are located, and it would then proceed to "draw" a box around the trash bag object, accurately categorizing it as a trash bag.

Other Computer Vision techniques include **semantic segmentation** which is "the task of clustering parts of images together which belong to the same object class [11]." It produces "...inference by predicting labels for every pixel in an input image [12]," and is critical for a more in-depth understanding at the pixel level in an image or video. It is used in medical imaging and autonomous driving where identifying specific boundaries can mean the difference between a life-threatening false positive or false negative. **Instance segmentation** also categorizes each pixel in an image like semantic segmentation, and goes one step further in recognizing separate instances of objects that belong to the same class [12], making it useful not only for identifying what categories are found within an image, but also for answering *how many* and *which* are present. **Object tracking**, **facial recognition**, **image generation**, and **activity recognition** are also important and more advanced Computer Vision topics related to **object detection** as a whole.

Within the field of Computer Vision, object detection tasks have a large number of real-world applications like surveillance and security. In retail, object detection can make

product identification easier on shelves and even track customer movement. Self-driving cars and pedestrian identification are also two useful applications within the automotive industry, and within healthcare, the same object detection tasks can be tailored to the identification of tumors in radiology or the detection of abnormal cells in other medical imaging software.

Object detection is the foundation for tasks in Computer Vision including object tracking, activity recognition and instance segmentation [13]. It has applications in gaming and augmented reality [14] as well as interactive systems like robots and drones that rely on object detection for their operations [13]. Additionally, it plays a role in image based search engines by enabling understanding of an image content beyond just image location metadata [15].

It is worth noting that object detection and other Computer Vision techniques such as image classification, semantic segmentation, instance segmentation, object tracking, facial recognition, image generation, activity recognition and motion analysis are not recent breakthroughs in the field of Artificial Intelligence. Rather, the recent breakthrough *really* lies in (1), the accuracy of these algorithms achieved through deep learning and (2), the wide array of applications made possible by these advancements [2].

2.2. Brief Overview of Neural Networks

In this section, a brief overview of Neural Networks will be discussed. Neural networks are the algorithms that have enabled recent advancements in Deep Learning, including object detection and image recognition. In an excellent visual demonstration by Grant Sanderson, a known author who specializes in demonstrations, highlights an interesting point about how the number 8 can appear differently or similarly in different images [16]. The same idea applies to images of cats. Sanderson also mentions that our brains occipital lobe, the visual cortex (V1) plays a crucial role in how we perceive and differentiate various objects [17].

Fig. 2.2. Handwritten digits can often appear differently with different pixel representations.

The process of seeing and the ability to identify between objects involves a series of complex processes across the Primary Visual Cortex (V1), The Visual Association Areas (V3, V3, V4, V5/MT) handle color processing, the recognizing of shapes, and motion

Fig. 2.3. Even the same digit can have a different a representation at the pixel level. Taken from [16].

detection [17]. The Dorsal Stream (for spatial awareness and object location), the Ventral Stream, (in charge of object and face recognition), the Fusiform Face Area or FFA [18], and the Parahippocampal Place Area (PPA) for recognizing scene and environmental stimuli [19] are all part of the brain's network that contribute to sight and object identification. This process combines a vast amount of visual and other sensory information, providing for an extraordinarily powerful identification ability.

According to a study titled, *"Comparing Object Recognition in Humans and Deep Convolutional Neural Networks,"* human vision and machine vision can be compared to one another. However, due to the "difference in scale with eye tracking heatmaps in humans, the use of saliency maps from DCNNs introduces a challenge in comparing the two [20]." Interestingly, despite these differences, the study found that vNet's performance in object recognition was significantly correlated with human performance. The study also showed that both humans and DCNNs were more accurate in recognizing inanimate objects compared to animate ones [20]. Although there are similarities between human vision and machine vision, the study does not explicitly claim superiority of one over the other. Nonetheless, humans are still inspired by the brain's ability to differentiate between objects and continue modeling processes modeled after the way the brain operates. These neural network algorithms are the main component in object detection programs.

Such programs are made to take in a grid or image of a certain number of pixels (in this example, 28 by 28 pixels), and be able to output a single label identifying the object found in the image [16].

Digit recognition for numbers 0 to 9 is a classic example used to explain how neural networks identify objects in images, but the example can be extended to any object.

2.2.1. Structure of a Neural Network

In neural networks, every individual pixel is treated as a neuron and holds a value ranging from 0 to 1 based on its color. A black pixel is represented by a value of 0.00 while a white pixel has a value of 1.00. So in an image with dimensions of 28 by 28 pixels, each of the 784 pixels (or neurons) will have a value that corresponds to the color intensity of that pixel [16].These pixels make up the input layer of the network, and on the hand, the output layer consists of neurons that represent object categories for detection within an image [16].

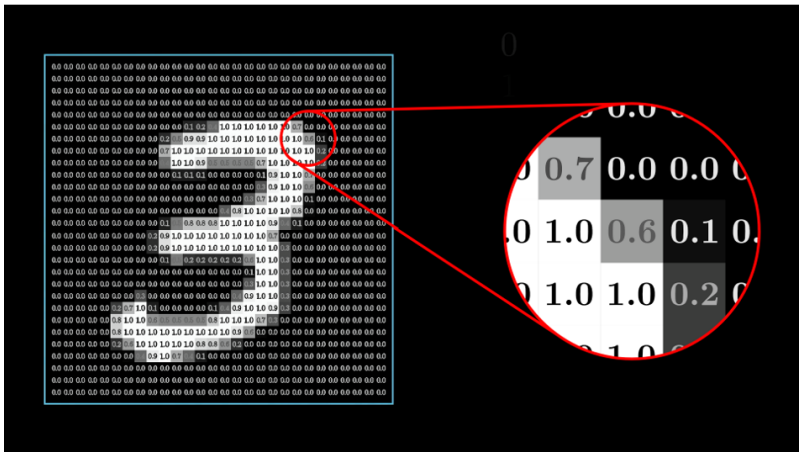


Fig. 2.4. A 28 x 28 image with all 784 pixels and their corresponding values. Each pixel is represented as a neuron with that value in the first layer of the network. Taken from [16].

To explain this using digits as an example, say a layer in a network consisting of 784 neurons corresponds to matches the number of pixels (28 by 28) in an image. Consequently, the final layer would consist of 10 neurons representing each digit from zero to nine. The layers situated between these two are commonly referred to as "hidden layers" and they play a crucial role in recognizing digits and their sub-parts through complex algorithms highly related to linear algebra [16].

Quoting Grant Sanderson's explanation, "the 'hidden layers are activated based on activations within one layer which subsequently determine activations in the next layer [16]." Mathematically speaking, the object classes or categories can be represented by using a vector ($a^{[L]}$) in the output layer. When the network predicts a digit, it essentially identifies the index that corresponds to the value in the output vector

$$\text{Prediction} = \arg\max_i a_i^{[L]}$$

Here i ranges from 1 to the number of possible digit categories (which is 10 for digits ranging from 0 to 9).

Figure 2.6 shows how activation values $(a_1, \ldots, a_n)$ in one layer are multiplied by the weights $(w_1, \ldots, w_n)$ that are associated with each link between a neuron in the second layer and the 784 neurons in the previous layer. When each activation is multiplied by its corresponding weight [16], the equation looks like: $S = w_1 \cdot a_1 + w_2 \cdot a_2 + \cdots + w_n \cdot a_n$. Pixels having negative values are identified using their neuron weights, as shown in Figure 2.5. According to [16], pixels with activation values closer to 1.00 will seem brighter, while those with activation values closer to 0.00 would appear darker.

Starting from the input layer, a network already trained to recognize digits will see an image that will activate all 784 input neurons based on the corresponding pixel values as mentioned previously [16]. According to the brightness of the pixel in each image [16], the specific pattern of activations (or value inputs) will cause other activations in the next layer. This process continues through the next layer until the end of the network [16].

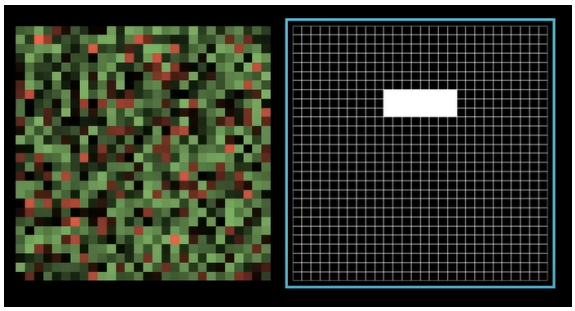


Fig. 2.5. Image depicting green pixels and red pixels. Adapted from reference [16]. In an image, the red pixels indicate pixels with negative weights while the green pixels represent pixels with positive weights in a weighted sum equation: $S = w_1 \cdot a_1 + w_2 \cdot a_2 + \cdots + w_n \cdot a_n$.

After the activations finally reach the output layer, the neuron with the highest activation value is the network prediction.

2.2.2. Patterns to Digits

Each neuron plays a specific role in the task of object detection on a smaller level in terms of sub-tasks. For example, some neurons could be responsible for detecting whether or not an image has an edge in a certain area, which is determined in part by the weights assigned to each one of the connections between a neuron and all other neurons in the previous layer [16]. Then, computing the weighted sum for all activations from each neuron in the previous layer, a pixel representation of an image with negative and positive weights would look like

$$a_1 w_1 + a_2 w_2 + \ldots + a_i w_i$$

[16]

Green pixels represent pixels with positive weights while red pixels represent pixels with negative weights [16]. When computing a weighted sum, the sigmoid function (one of many types of activation functions) allows normalization of the activation values so that they fall between zero and one for mathematical convenience.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

[16]

After applying the sigmoid function, the equation would become

$$a_o^1 = \sigma\left(w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \cdots + w_{0,n} a_n^{(0)}\right) \tag{2.1}$$

[16]

which is, according to Sanderson in his demonstration, an indication of how positive or how negative the value of the weighted sum will be [16]. It is also important to introduce a bias for when the weighted sum of activations for when a certain neuron is above or

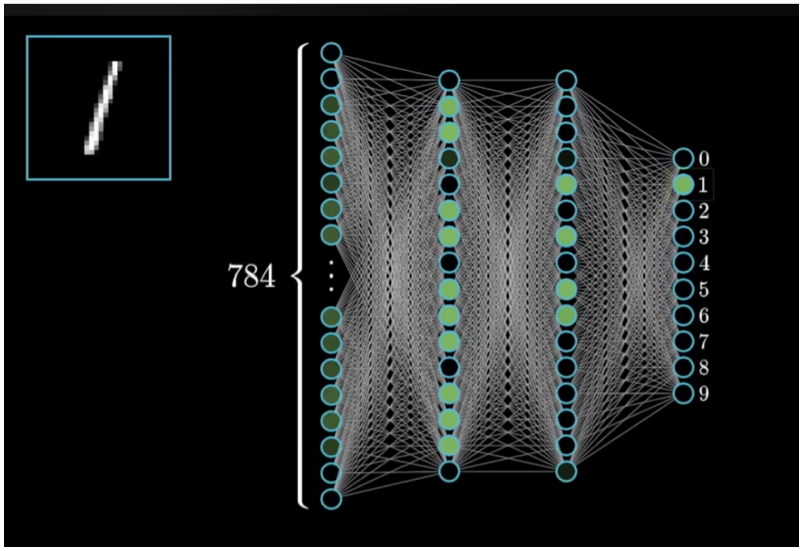


Fig. 2.6. Taken from [16]. A neural network with 4 layers. The first layer (input layer) contains 784 neurons, each representative of the 784 pixels (28x28) in the image of the number "3." The next two layers are known as "hidden layer 1" and "hidden layer 2." The network's output shows that the neuron for the number "1" has the highest activation, representing its correct "guess" for the image it has just seen. Indeed, the digit the network believes it has "seen" is the number "1."

below a certain value [16]. This is done by subtracting ten or any other number at the end of the weighted sum before the sigmoid function is applied [16]:

$$\sigma\left(a_1 w_1 + \ldots + a_n w_n - 10\right)$$

The weights indicate the pattern detected by the neurons in the second layer, and a bias specifies how high the weighted sum should be before a neuron reaches a significant value [16]. The same process holds for each hidden layer between the input layer and the output layer.

It should be stated that linear algebra plays a very important role in the mathematics behind neural network algorithms [16]. The role of weighted sums and biases is explored more in detail. These elements can be represented as products between matrices.

The matrix representing the weights, $\mathbf{W}$, can be defined as

$$\mathbf{W} = \begin{bmatrix} w_{0,0} & w_{0,1} & \ldots & w_{0,n} & b \end{bmatrix} \tag{2.2}$$

[16]

The matrix vector representing the activations $\mathbf{A}^{(0)}$, can be defined as

$$\mathbf{A}^{(0)} = \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \\ 1 \end{bmatrix} \tag{2.3}$$

[16]

And after this, combining the matrices for the weights, bias, and activations would look as follows without expansion:

$$a_o^1 = \sigma\left(\mathbf{W}\mathbf{A}^{(0)}\right) \tag{2.4}$$

[16]

and expanding the terms:

$$a_o^1 = \sigma\left(w_{0,0}a_0^{(0)} + w_{0,1}a_1^{(0)} + \cdots + w_{0,n}a_n^{(0)} + b\right) \tag{2.5}$$

[16]

In the calculations above, it is assumed there are k neurons in the current layer and n neurons in the previous layer so that the weight matrix would have dimensions $k \times (n+1)$. The activation vector would then have dimensions $(n+1)$, and the dimension of the weight vector would be $k \times 1$ while the bias vector would have a dimension of $k \times 1$.

In more detailed matrix form, $\mathbf{W}$ would be defined as

$$\mathbf{W} = \begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \tag{2.6}$$

The activations matrix $\mathbf{A}^{(0)}$ would be

$$\mathbf{A}^{(0)} = \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} \tag{2.7}$$

[16]

The vector $\mathbf{B}$ containing the biases for each weighted sum between each neuron and each of the neurons in a previous layer would be

$$\mathbf{B} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_k \end{bmatrix} \tag{2.8}$$

[16]

Then when the weights $\mathbf{W}$ are multiplied by the activations $\mathbf{A}^{(0)}$ and the biases $\mathbf{B}$ are added in, the next activations vector $\mathbf{A}^{(1)}$ in the next layer would be

$$\mathbf{A}^{(1)} = \sigma\left(\mathbf{W}\mathbf{A}^{(0)} + \mathbf{B}\right) \tag{2.9}$$

where:

- the terms will look like $\sigma(w_{i,j} * a_j^{(i)})$ where i ranges from 0 to k and j ranges from 0, to n.
- with the first term being $\sigma(w_{0,0}a_0^{(0)} + w_{0,1}a_1^{(0)} + \cdots + w_{0,n}a_n^{(0)} + b_0)$
- and the second term as: $\sigma(w_{1,0}a_0^{(0)} + w_{1,1}a_1^{(0)} + \cdots + w_{1,n}a_n^{(0)} + b_1)$

and so on, until reaching the kth neuron [16]. The sigmoid function is applied to each weight vector inside, and the code for such operation looks something like:

```
class Network(object):
  def __init__(self, *args, **kwargs):
    # yada yada, initialize weights and biases...

  def feedforward(self, a):
    """Return the output of the network for an input vector a"""
    for b,w in zip(self.biases, self.weights):
      a = sigmoid(np.dot(w,a) + b)
    return a
```

Fig. 2.7. Sample python code showing how to represent the activation from one layer to the next in a neural network. Specifically, Equation 2.10. Taken from [16].

2.2.3. Explanation for a Layered Structure

Neural networks are structured in the form of layers because there is usually a specific function for each layer [16]. In the digit example and as can be seen in Figure 2.8, a "9" contains a stick and a loop, an "8" contains two loops crossed together, and a "4" involves a stick on the right side with a smaller "L" shape meeting that stick in the middle. To recognize a number as a whole, the network must be able to recognize the combination of each subcomponent, which also involves being able to identify the subcomponents themselves [16]. As an example, the first hidden layer in the network could correspond to the detection of subcomponents like edges, activating all of the neurons in the current layer that are associated with specific edges. These neurons would then activate the neurons in the second hidden layer associated with an upper loop, for example, such as in the digit "8" or the digit "9"). Then, the neurons associated with a long vertical line (such as in the number "9") would also become activated, and in the final layer of the network, the neuron representing the number "9" will have the highest activation value on a scale of 0.00 to 1.00 [16]. This would then indicate the network's prediction.

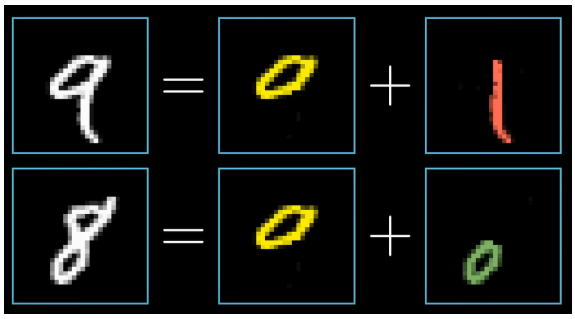


Fig. 2.8. Parts of an image are represented by how each layer in a neural network identifies them. Taken from [16].

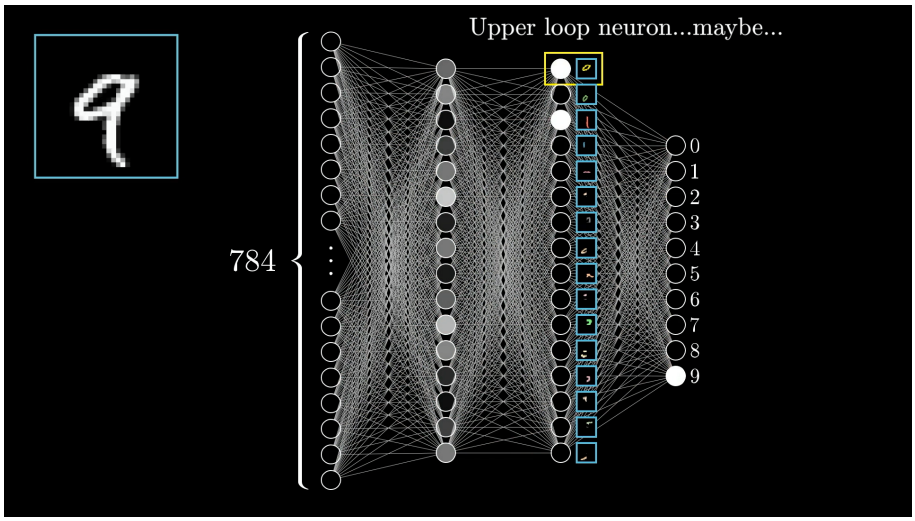


Fig. 2.9. Hidden layers in a neural network and their expected functions in object detection and classification (not always known). The function for hidden layer 1 could be in determining the edges of a digit. The function for hidden layer 2 could be in detecting other areas or features of a digit. The neuron highlighted in the figure is thought to be activated by presence of an upper loop (specific to the digits 8 and 9). Taken from [16].

Figures 2.8 and 2.9 explain how different layers in a network and different neurons within each layer can play a role in identifying specific features of an image. Recognizing a loop is also a sub-problem in itself, which could be the function of each neuron in a specific network layer. However, neural networks are complex models which sometimes offer much less explanation in terms of how exactly they are working internally [16]. "Learning" in machine learning refers to finding and tuning the weights and biases in such a way that the network is able to perform a given task [16].

To find out more about the learning strategy of a neural network, read about Gradient Descent and Back-propagation.

2.3. YOLO (You Only Look Once) Algorithm

YOLO is a single-stage object detection system and the neural network architecture of choice used in training a garbage detection machine learning model for this project. YOLO is trained on several datasets throughout its various versions including the PASCAL VOC (Visual Object Classes dataset), the COCO dataset, and the ImageNet dataset. It is unique in that it treats object detection as a regression problem, as opposed to using classifiers adapted for object detection [21]. It is single-staged, meaning it performs both detection (for identifying bounding boxes) and classification (for class probabilities) in a single pass through the network.

Traditional object detection systems are usually two-stage detectors such as R-CNN, Faster R-CNN like, and DPM. During the first stage, these detection systems generate potential bounding boxes, and in the second stage they perform classification of each proposed region [21]. These two-staged detectors tend to be slower due to the two-staged process [21].

YOLO is known for its speed and real-time processing capabilties. It is capable of handling 45 frames per second, which is very significant for real-world applications like video streaming [21]. The even faster variant, Fast YOLO, is able to process up to 155 frames per second [21].

Despite its advantages, YOLO still comes with some challenges. Errors in localization is one disadvantage. This means it can sometimes misplace objects or struggle with detecting smaller objects [21]. This happened to be one of the problems encountered by the student during research when training the garbage detector for this project.

2.3.1. Historical Context and Evolution of YOLO

The YOLO model has progressed significantly through the various iterations since its development in 2015 from the first YOLOv1 to the most recent YOLOv8 [8], [23].

YOLOv1
The first version of YOLO (You Only Look Once) presented in 2015 was revolutionary because it accomplished both object localization and classification tasks in one pass. However, while fast, YOLOv1 had lower accuracy compared to state-of-the-art models at that time, such as Faster R-CNN [8].

YOLOv2 (YOLO9000)
YOLOv2 came with the nickname YOLO9000 because of its ability to detect over 9000 object classes. It uses the neural network DarkNet-19, a 19-layer architecture. YOLOv2 detected smaller objects and was faster than its competitors while also staying quite accurate [8].

YOLOv3

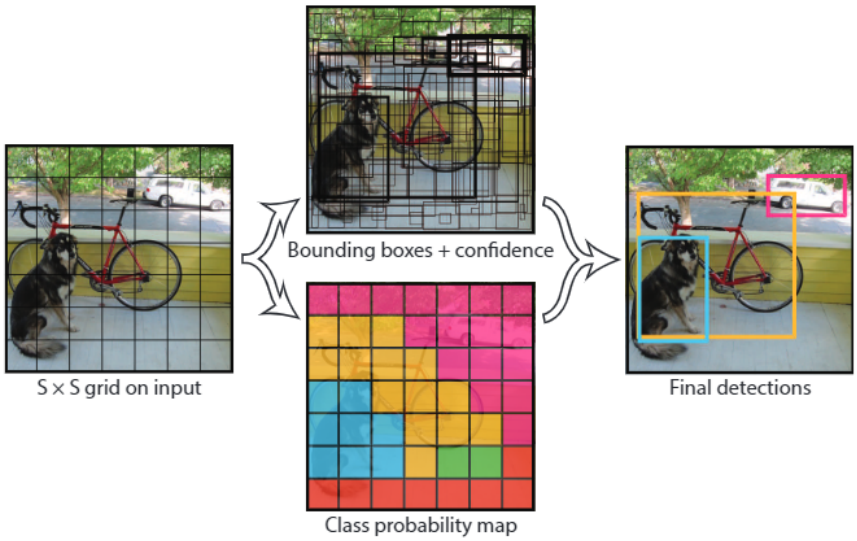


Fig. 2.10. Reprinted from Redmon, Divvala, Girshick, *et al.* [21], Figure 2. YOLO divides an image into an $S \times S$ grid. For each grid cell, it predicts B bounding boxes and confidence scores for the corresponding boxes, and C class probabilities. The result is an $S \times S \times (B * 5 + C)$ tensor [22].

YOLOv3 uses the neural network DarkNet-53's architecture, which led to better feature extraction. It also employed logistic regression for objectness prediction and sum-squared error loss for bounding box predictions. This made it significantly better at detecting smaller objects. However, it was slightly slower than YOLOv2 due to its deeper architecture [8].

YOLOv4

YOLOv4 was not created by Joseph Redmon (original creator of YOLO), but it did introduce several new techniques to continue improving speed and accuracy. It also uses the Darknet53 as its backbone architecture and integrated the PANet and SAM block for better integration of features and better object detection [21]. It also implemented the CIOU loss function which considers shape and orientation for more accurate bounding box prediction. These changes made YOLOv4 more efficient and accurate than YOLOv3 [8].

YOLOv5

YOLOv5, despite its name, is not the YOLO variant that followed YOLOv4. It was developed separately by a company called UltraAnalytics and was geared toward a more user-friendly deployment capability [8]. It is the architecture variant of choice for this project and is discussed more in detail in the *Methodology: YOLOv5 Neural Network* section of this paper.

YOLOv6

YOLOv6 had an anchor-free approach and revised backbone and neck as well as a two-loss function. These are the changes that distinguished it from other YOLO versions [23].

YOLOv7, YOLOv8

YOLOV7 focused on improving accuracy and maintaining speed, and YOLOv8 (like YOLOv5) aims at accomplishing the difficult task of high-speed inference also on edge-devices [23].

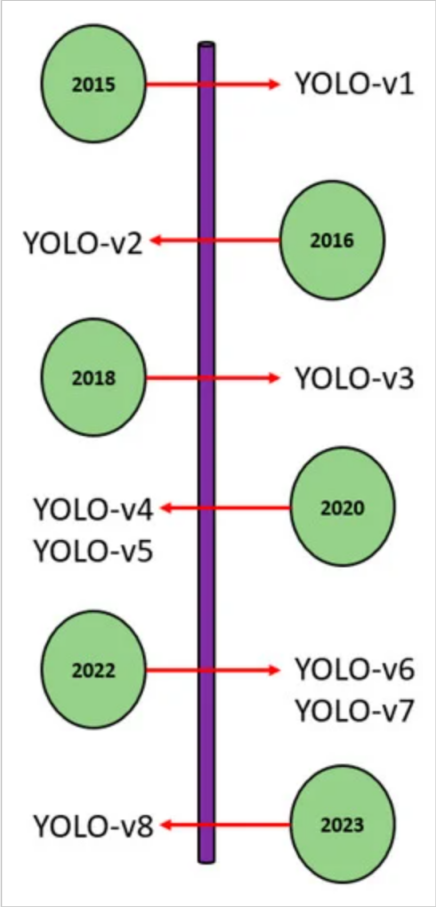


Fig. 2.11. Taken from Hussain [23], Figure 2: YOLO evolution timeline.

2.3.2. Full Architecture Summary

The architecture of YOLO is different depending on its version. For instance, the original YOLO uses a single convolutional network that predicts both class probabilities and bounding box coordinates. With an input image of size 448×448, the network outputs a tensor with the shape $7 \times 7 \times 30$ (for YOLO v1). In the size $7 \times 7 \times 30$, 30 is the number of channels which come from 2 bounding box predictions [23]. Each box has 5 predictions: $x, y, w, h,$ confidence) and 20 class probabilities [23] (for the VOC dataset with 20 classes).

Training: The network's layers are pretrained on the ImageNet classification dataset and the PASCUAL VOC dataset. The network uses a loss function that takes object presence into account as well as bounding box accuracy and classification accuracy. Dropout and

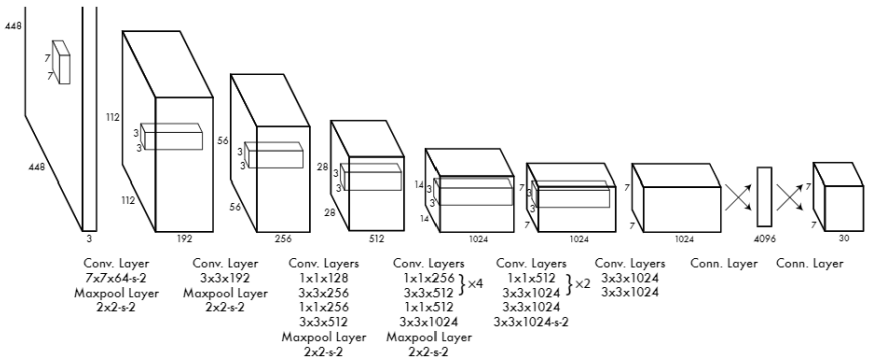


Fig. 2.12. **YOLO Architecture.** Reprinted from Redmon, Divvala, Girshick, *et al.* [21], Figure 3. *"The YOLO detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating* 1×1 *convolutional layers reduce the features space from preceding layers. The convolutional layers are pretrained on the ImageNet classification task at half the resolution (*224×224 *input image) and then double the resolution for detection."*

data augmentation (scaling, translations, and color adjustments in the HSV color space) are used to avoid overfitting and create variation in the training data [23] [21].

The final output YOLO gives for the PASCAL VOC dataset is a $7 \times 7 \times 30$ tensor, which is caused by the grid size and the number of predictions per grid [21]. For **activation and optimization**, the rectified linear activation is used for non-linearity. The sum-squared error is used for optimization, but is used with different weights for different kinds of prediction errors. Doing this would look like:

$$E_w = \sum_i w_i (y_i - \hat{y}_i)^2$$

where w_i is the weight for the i-th prediction error [21].

Bounding Box Predictions

Each cell predicts:

- (x, y): Center of the box relative to the grid cell, normalized between 0 and 1.
- w and h: Width and height predictions, normalized with respect to the image.
- Confidence score: Reflecting both the presence of an object and the accuracy of the bounding box.

Loss Function

The loss function used in YOLO as described by Redmon, Divvala, Girshick, et al. Redmon, Divvala, Girshick, *et al.* [21] is given by:

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{obj}}^{ij} \left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right) \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{obj}}^{ij} \left((\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right) \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{obj}}^{ij} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{\text{noobj}}^{ij} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} 1_{\text{obj}}^{i} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

where Line 1 represents the **Bounding Box Coordinate Loss**, Line 2 represents the **Bounding Box Dimension Loss**, Line 3 represents the **Objectness Loss**, Line 4 represents the **No Objectness Loss**, and Line 5 represents the **Class Probability Loss**

and

- λ_{coord} and λ_{noobj} are the weights for the coordinate and no objectness losses.
- S^2 is the number of grid cells in an image.
- B is the number of bounding box predictors.
- 1_{obj} and 1_{noobj} are opposites and tell whether there an object is or is not found in a grid cell, and
- i and j indices iterate over grid cells and bounding box predictors respectively.

Credit: Redmon, Divvala, Girshick, et al. [21]

Non-max Suppression

Post prediction, non-max suppression is applied to filter out duplicate detections. For overlapping boxes, if two boxes predict the same class but have different confidence scores, the one with the lower score is discarded [21].

2.3.3. YOLOv5 Nano

Below is an overview of the YOLOv5 Nano variant's architecture in detail taken from the yolov5n.yaml file [24] from the UltraAnalytics YOLOv5 Github Page.

```
1  # YOLOv5 by Ultralytics, AGPL-3.0 license
2
```

```yaml
# Parameters
nc: 80  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.25  # layer channel multiple
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],  # 5-P4/16
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],  # 7-P5/32
   [-1, 3, C3, [1024]],
   [-1, 1, SPPF, [1024, 5]],  # 9
  ]

# YOLOv5 v6.0 head
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],  # cat backbone P4
   [-1, 3, C3, [512, False]],  # 13

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],  # cat backbone P3
   [-1, 3, C3, [256, False]],  # 17 (P3/8-small)

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],  # cat head P4
   [-1, 3, C3, [512, False]],  # 20 (P4/16-medium)

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],  # cat head P5
   [-1, 3, C3, [1024, False]],  # 23 (P5/32-large)

   [[17, 20, 23], 1, Detect, [nc, anchors]],  # Detect(P3, P4, P5)
  ]
```

Parameters include:

- **nc:** The number of classes, which is 80 in this configuration.
- **depth_multiple** and **width_multiple:** Multipliers to adjust the depth and width of the model. This makes the model smaller or larger without completely redesigning the architecture.
- **anchors:** Pre-defined anchor box sizes for three different scales (P3/8, P4/16, P5/32).
- **Backbone:** Component in charge of feature extraction for the entire architecture.

Architectural Components:

Conv: is a convolutional layer. The arguments are [number of filters, kernel size, stride, padding].

C3: is a CSPNet bottleneck layer which is a design to improve the gradient flow in deep networks. The number shows how many times this bottleneck is repeated.

SPPF: is the spatial pyramid pooling feature which allows the network to interpret features at different scales (zoomed in or zoomed out, for example) and put them into a single tensor.

Head: is where the actual bounding box predictions and class predictions are made.

nn.Upsample: is an upsampling layer which increases the spatial dimensions of the feature maps.

Concat: concatenates features from different layers which is useful in architectures where multi-scale features are combined, especially in object detection.

Detect: is the detection layer which outputs the predicted bounding boxes, objectness scores, and class predictions.

The model begins with a layers that reduce the input size by half each time. Then there are some layers called C3 layers that help extract features at scales. To combine these features there is a layer that merges them into one tensor. In the head part of the model Upsample layers are used to make the feature maps larger allowing for predictions, at resolutions. The Concat layers bring together features from both the backbone and the head sections. Finally the Detect layer takes these processed tensors. Produces predictions for bounding boxes, objectness and class.

This architecture of YOLOv5 allows it to effectively detect objects of sizes. The three scales (P3/8, P4/16 P5/32) correspond, to large, medium, and small objects in that order.

2.3.4. Inference with YOLO:

Number of Predictions: On the PASCAL VOC dataset, the network generates predictions for 98 bounding boxes for each image and class probabilities for every box. These predictions are very important for YOLO's speed during inference [21].

Grid Detection: The division of images into grids allows diversity in bounding box predictions. Usually it is obvious which grid cell an object corresponds to, so the network mostly produces one box for each object [21].

Single Evaluation: For making predictions on a test image, YOLO needs only one evaluation of its neural network. This is very efficient compared to classifier-based methods which need multiple evaluations [21].

Fig. 2.13. Inference with YOLOv5 nano conducted by the student on a trash image from the validation set.

Overlapping Predictions: In situations where bigger objects take up multiple grid cells or when objects are close to grid borders, more than one grid cell might predict bounding boxes for the same object [21].

Performance Boost: Using non-maximal suppression isn't as important for YOLO as it is for other methods like R-CNN or DPM. Incorporating non-maximal suppresion improves the mean Average Precision (mAP) by 2-3% [21].

2.4. Existing Garbage Detection Systems

Waste management is both a global and local challenge in many cities. Garbage detection methods that exist today use artificial intelligence, robotics, and the Internet of Things (IoT). These are the processes involved in some of the garbage detection systems being used currently to detect and manage trash pick-up, sorting, and more. Drones and wearable cameras to handle ground trash pick-up are just some of the technologies available for building garbage detection systems.

Image-based Garbage Detection using Deep Learning is the first way a garbage detection system might be built. It is the method adopted in this project to train an effective garbage detection system and involves CNNs trained on labeled datasets that contain images of garbage to recognize and classify different types of trash, such as plastics, metals, organic waste, and more. This method can be used in drones or CCTV cameras for aerial surveillance of large areas like beaches or parks [25].

The use of IoT-based smart bins by integrating sensors into garbage bins is another way to implement such a system. The sensors detect when garbage bins are full and can send notifications for collection. Some bins are even equipped with cameras and AI algorithms to automatically classify and dispose of waste correctly [26].

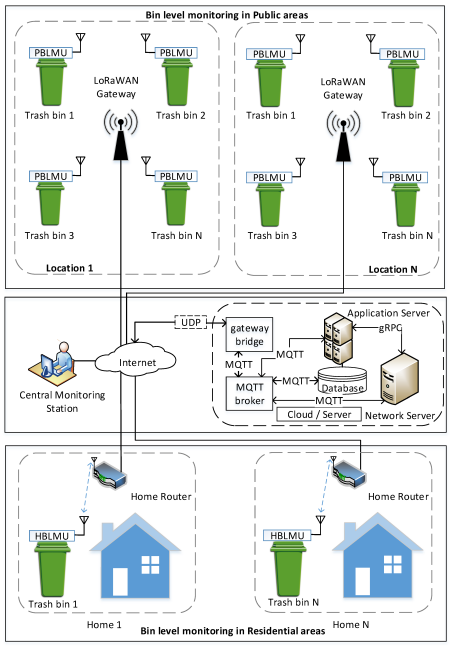


Fig. 2.14. Taken from Vishnu, Ramson, Senith, *et al.* [26], Figure 1. The network architecture of the developed IoT-based smart bin system.

Robotics can also be used to build and deploy robotic waste sorters. These can also be used with cameras and AI algorithms which can sort waste on conveyor belts in garbage treatment plants. For example, they can be trained to recognize and separate recyclables from non-recyclables in classification tasks. [27]

Drones can also be attached with cameras and used for garbage detection to easily view large areas of land [28]. They can see garbage or litter using AI models and even identify trash patterns if there are any [28].

Garbage Detection is more complicated in under water than on land, so specialized systems are developed to detect and collect garbage in oceans, rivers, and lakes. The "Ocean Cleanup" project is an example. It aims to get rid of plastic from oceans using detection and collection algorithms like net and floating systems, satellite imaging, underwater drones, AI and machine learning [29].

Even mobile apps exist to help classify and identify garbage. Applications including Litterati allow users to take pictures of garbage and upload them to an already existing dataset [30]. The TACO (Trash Annotations in Context) dataset is an example of this and consists of images that have been uploaded by users to add to trash data.

This concludes the section on existing garbage detection systems.

3. METHODOLOGY

3.1. The YOLOv5 Neural Network

The YOLOv5 neural network was trained on the COCO dataset, consisting of more than 200,000 labeled images and 1.5 million instances of objects from 80 categories. It has a speedy and lightweight architecture which makes it a great choice for deployment on edge-devices and the choice for neural network architecture for the garbage detector in this project. single neural network is able to process an entire image in a single pass [23].

It consists of three parts: a backbone in charge of feature extraction, a neck for feature aggregation, and a head in charge of generating bounding boxes, classes, and confidence scores [23]. Its backbone is the DarkNet53, the same pre-trained network from YOLOv3 and YOLOv4, used to extract features and increase the spatial resolution of images [23]. Its neck is used to extract features, [23] and lastly, the model head is used to perform the final stage which involves generating anchor boxes on feature maps as well as the final result which includes: classes, objectivity scores, and bounding boxes [23].

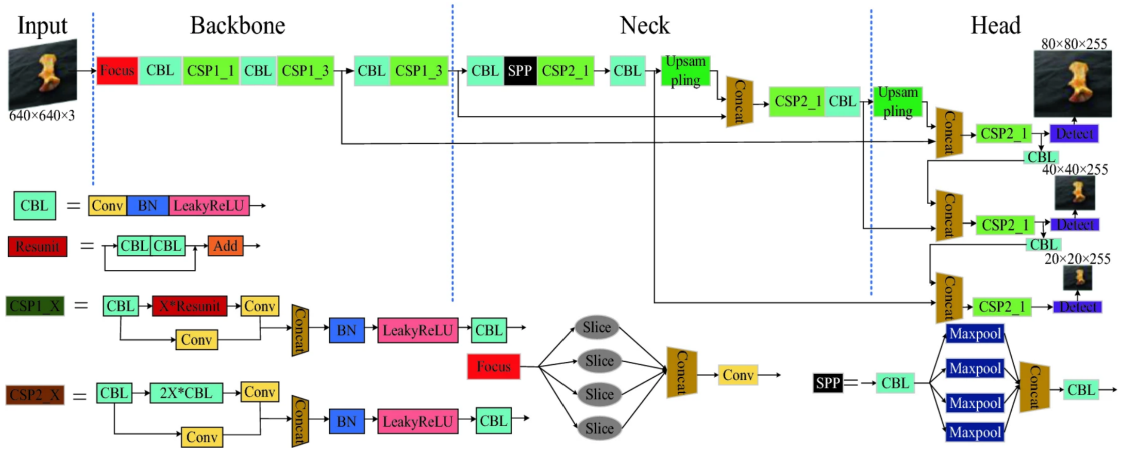


Fig. 3.1. Reprinted from Jiang, Hu, Qin, *et al.* [31], Figure 1. YOLOv5 architecture.

YOLOv5 was the first in the YOLO familiy to be written in PyTorch instead of Darknet [23]. YOLOv5 also automated the anchor box mechanism using k-means clustering on the COCO dataset which allowed the network to learn the best-fit anchor boxes for any custom dataset [23].

YOLOv5 comes in several variants with different parameters. For example, YOLO-v5n has a weight file size of 4 MB, compared to 89 MB for the larger YOLO-v5l [23].

The Nano variant was the model of choice in this project. It is smaller in size and more convenient for environments with limited resources, such as is on edge devices like smartphones, watches, sensors, IoT devices, and more. YOLOv5 Nano aims to provide a

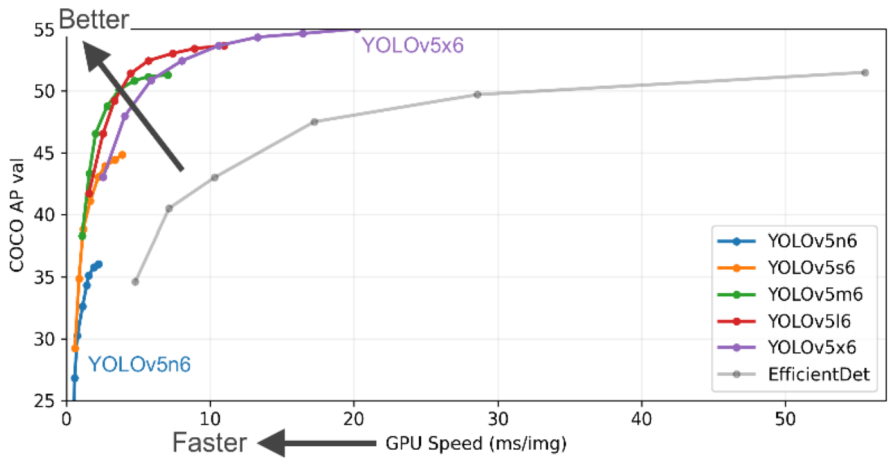


Fig. 3.2. Taken from Gillani, Munawar, Talha, *et al.* [32], Figure 2. YOLOv5 variant comparison vs. EfficientNet

balance between model size, computational demands, and detection performance.

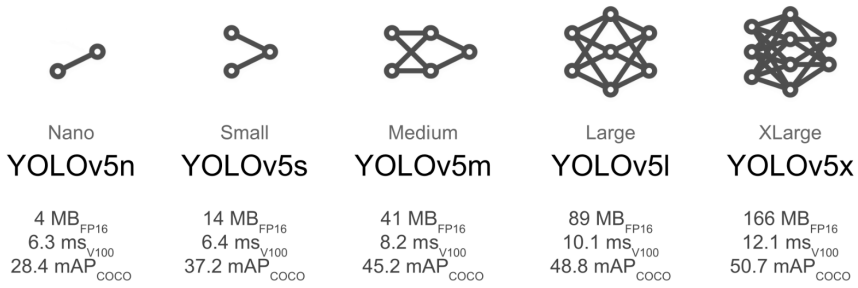


Fig. 3.3. Taken from Ultralytics [24]. YOLOv5 variant sizes.

YOLOv5 Nano Features

- **Size:** YOLOv5 Nano is designed with 1.9 million parameters, which's actually fewer compared to other YOLO variants. This model achieves a size by incorporating convolutional layers and reducing the channel sizes making it ideal, for edge devices where space is limited.
- **Layers**: One advantage of YOLOv5 Nano is its parameter count and fewer convolutional layers. This allows for detections making it perfect for real time object detection on edge devices.
- **Detection**: Additionally YOLOv5 Nano performs detections with Floating Point Operations Per Second (FLOPs) requirement making it more convenient for real time object detection on edge devices.
- **Feature Extraction**: Another aspect that sets YOLOv5 Nano apart is its utilization

of feature extractors (backbones) that can capture image features, with computational cost.

Model Name	Params (Million)	Accuracy (mAP 0.5)	CPU Time (ms)	GPU Time (ms)
YOLOv5n	1.9	45.7	45	6.3
YOLOv5s	7.2	56.8	98	6.4
YOLOv5m	21.2	64.1	224	8.2
YOLOv5l	46.5	67.3	430	10.1
YOLOv5x	86.7	68.9	766	12.1

Fig. 3.4. Reprinted from Ultralytics [24], YOLOv5 model overview

3.1.1. Implications for Edge Deployment

Some implications for edge deployment are:

1. **Memory**; YOLOv5 Nano's smaller model size makes it suitable for edge devices with memory.
2. **Speed**; Given the resource constraints on edge devices, faster inference time enables real time object detection within limits.
3. **Power**: The computational complexity allows for reduced power consumption, which is ideal for battery powered edge devices.
4. **Easy Deployment**: The smaller and faster model simplifies the process of deploying and remotely updating it across a network of edge devices when necessary.

Figure 3.4 provides an overview of all YOLOv5 models, including their inference speed on CPU and GPU as the number of parameters required for images with size 640x640 pixels. These factors strongly support the selection of the YOLOv5 Nano variant as the model, for this project.

4. CUSTOM DATASET PREPARATION AND USAGE

During the process of developing a machine learning model that can accurately identify and classify types of trash we encountered a significant challenge; the quality and specificity of the data. Initially using a dataset, with images labeled as 'trash' was not enough. The lack of images containing one specific type of trash in the training data caused the model to struggle in distinguishing between types of trash, such as glass, garbage cans, bags, sticks, plastics and so on. This highlighted the importance of having a constructed dataset for machine learning model convergence. This section describes how the student prepared and utilized custom datasets to improve the models accuracy and redefine its understanding of what constitutes 'trash.'

To begin, the YOLOv5 Nano model trained on the COCO dataset was used for transfer learning. A list of checkpoints can be found on their Github. Below is an overview of each custom dataset that was then specifically created to train the garbage detector. The goal was to enable it to identify "trash" while disregarding the 80 object classes present in the pre trained model.

"Your model is only as good as your data." - Unknown.

The following datasets were constructed by the student with images from Roboflow, a website used for preparation, augmentation, labeling, and deployment of datasets for machine learning and computer vision projects. The problem with generic, non-specific trash "types" in the data images resulted in the model's inability to converge, meaning that metrics such as precision, mAP (mean Average Precision), and recall remained extremely low and unstable throughout many of the training runs.

4.1. Model Convergence:

A model is considered to have converged under the circumstances:

- The performance metrics such, as mean Average Precision (mAP) precision and accuracy have reached a point of stability indicating that the model has absorbed all the knowledge it can from the training data and is no longer making improvements.
- A given metric displays consistency across runs suggesting that the model consistently produces results when presented with new data.
- There is no evidence of overfitting; If the metrics obtained from training and validation data show values (i.e. there isn't a difference, between training metrics and validation metrics), it suggests that the model has reached convergence.

References; [33] [34]

4.2. Performance Metrics for Determining Model Convergence:

Performance metrics were recorded from each training session and stored in MLflow. Used to define both convergence and a successful model, their automatic upload to MLflow was included as part of the MLOps pipeline used for training and testing.

- **Precision**: Ratio of correctly predicted positive observations to the total predicted positives. It gives you an idea of when your model predicts "Yes, there's an object", i.e. how often it is correct [35] and is defined by:

$$p = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.1}$$

- **Average Precision (AP)**: AP is found by taking the average precision values at different recall levels. Recall is the ratio of correctly predicted positive observations to all the observations in the actual class [36]. For object detection, precision is usually plotted against recall for different threshold levels, and the AP is the area under the curve [36]. To calculate the Average Precision (AP):

  1. Arrange the detections, in descending order based on their confidence.
  2. Compute the precision. Recall for each detection as you go down the list.
  3. Represent the computed values on a graph with recall on the x axis and precision on the y axis.
  4. Determine the area under this curve by integrating or summing it.

$$\text{AP} = \sum_i (R_i - R_{i-1}) \times P_i \tag{4.2}$$

  where:

  - R is recall
  - P is precision
  - i indexes over the sorted list of detections

  [36]

- **Mean Average Precision (mAP)**: Average of the AP over all classes [36].

- **Mean Average Precision (mAP) at 0.5**; This metric represents the mAP value when the Intersection over Union (IoU) threshold is set to 0.5. In order for a bounding box to be classified as a "detection or a true positive it must have an overlap of least 50% with the ground truth bounding box. If the IoU is below 0.5 the prediction is considered a false positive [36]. To compute Average Precision $mAP_{0.5}$:

1. For each class:
   (a) Get the precision, and recall at different thresholds.
   (b) Calculate the Average Precision (AP) for that category using an IoU threshold of 0.5.
2. Get the average of all AP values, across all classes to obtain $mAP_{0.5}$.

- **IoU (Intersection over Union)**: A measure of the overlap between two bounding boxes (the ground truth and the prediction). It's a way to understand how close the predicted bounding box is to the ground truth [36]. It's given by:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{4.3}$$

- **Box loss**: In object detection, "box loss" is the loss associated with how accurate the bounding box predicted by the model is compared to the actual (ground-truth) bounding box of the object in the image. This loss makes sure that the predicted bounding box is "drawn" tightly and accurately around the object [36]. The IoU loss is:

$$\text{IoU Loss} = 1 - \text{IoU} \tag{4.4}$$

4.3. Challenges in Dataset Construction for the Garbage Detector:

The process of developing a garbage detector that can accurately identify and classify trash is continued in this section with the model's inability to differentiate between different forms of trash. This problem is tackled in depth by performing training runs with over 17 different hand-constructed datasets from Roboflow. Sample images are provided for significant training runs.

It is important to point out that the largest problem encountered in the project besides the lack of specific trash types in the dataset images was the use of precise labels in the training images. Labels appear like bounding boxes, but instead of these boxes being produced as output by the model as predictions, they are actually boxes used as labels on the input images. They tell the model where the trash is located in each training image. Their purpose is crucial to the way the model learns to identify trash to produce its own labels (predictions), and sometimes the process of creating or editing these labels can be very complex. In this project, making sure image labels for training data accurately defined the trash locations in each image was a challenge for the following reasons:

Ambiguity: The concept of "trash" is quite broad and can refer to materials such as glass, plastic, garbage bags, bins, cardboard and more. During the training process the biggest challenge faced was determining what exactly falls under the category of trash. It was often unclear whether certain items should be labeled as trash or not. As a result, this ambiguity in the data required training the model with datasets until an accurate outcome was achieved.

Precision: Finding the balance of detail when annotating trash objects in images also caused difficulties. It was necessary to make sure that bounding boxes encompassed the trash object without being overly restrictive. If a bounding box missed parts or if it included background details by being too large, it could affect the model's performance.

Consistency: Consistency among bounding box sizes for objects across different images was also important. This consistency allowed the model to learn the features. Inconsistencies in bounding box sizes and shapes could introduce biases or irregularities during training.

Blockage: In real life scenarios, trash objects often block each other. Drawing bounding boxes in these situations proved to be challenging.

Scale: some objects appeared closer to or farther away from the camera within images. Occasionally, bigger pieces of trash seemed to be much more distant and therefore smaller, while smaller trash items appeared to be closer. This scenario also made it difficult to remain consistent in data labeling.

4.4. Dataset Experiments

The Roboflow repository containing the individual datasets from this subsection can be found here. Some of the access links to the individual dataset within the repository are also included.

Attempt 1 - DATASET garbageV1

Fig. 4.1. Some images from the first dataset - DATASET garbageV1.

- **Description:**
  - About 6000 images.
  - Dataset built with roboflow, annotations included.

  - Bounding boxes too large.
  - Too much noise (garbage heaps).
  - Garbage heaps occupy the entire image in many images and very little of the image at other times.
  - Images are too granular (glass, mounds, entire objects, garbage bins, rubble, bags, etc.).

Attempt 2 - DATASET garbageV2

- **Description:**
  - 1000 images.
  - Dataset built with roboflow, annotations included.
  - Smaller bounding boxes with individual, more defined garbage heaps.
- **Preprocessing:**
  - Auto-Orient: Applied.
  - Static Crop: 25-75% Horizontal Region, 25-75% Vertical Region.
  - Resize: Stretch to 320x320.
  - Modify Classes: 7 remapped, 0 dropped.

Attempt 3 - DATASET garbageV2.1

- **Description:**
  - Same dataset from attempt 2 with the augmentations excluded.
  - 1000 images.
  - No augmentations applied (Note: Since YOLOv5 already applies augmentations).
- **Preprocessing:**
  - Auto-Orient: Applied.
  - Static Crop: 25-75% Horizontal Region, 25-75% Vertical Region.
  - Resize: Stretch to 320x320.
  - Modify Classes: 7 remapped, 0 dropped.

Attempt 4 - DATASET garbageV3

- **Description:**
  - 1000 images of individual trash objects rather than of trash piles and pictures taken from the streets.

Fig. 4.2. Images from the garbage version 3 (garbageV3 dataset.

Attempt 5 - DATASET garbageV3.1

- **Description:**
  - Add 810 more images to the data for a total of 1810 images.
  - Resize to 320x320 (originally 640x640).
  - Delete images with bounding boxes occupying more than 80% of the image.

Attempt 6 - DATASET garbageV3.1.1

- **Description:**
  - Same dataset as garbageV3.1 except without resizing images.
  - Let YOLO resize the images.

Attempt 7 - DATASET garbage V4

- **Description:**
  - 1000 garbage images with ONE type of garbage: trash bags next to garbage bins, full garbage bins, etc.

Fig. 4.3. Sample images from Attempt 7 - garbageV4 dataset.

Attempt 8 - DATASET garbageV4.2

- **Description:**
  - Same garbageV4 dataset in Attempt 7.
  - Edited bounding boxes.
  - Added blurring, color augmentations, and contrast to increase variability of the training data.

Attempt 9 - DATASET garbageV5

- **Description**
  - Link to Dataset on Roboflow.
  - Created bounding boxes for all 1000 images by hand.

Fig. 4.4. Images from the garbageV5 dataset.

Attempt 10 - DATASET garbageV5.1

- **Description**
  - Link to Dataset on Roboflow.
  - Same dataset as garbageV5 with the following changes:

  - Increased number of images to around 1600.

- **Augmentations**
  - Grayscale: Apply to 25% of images.
  - Brightness: Between -25% and +25%.

Attempt 11 - DATASET garbageV5.2

- **Description**
  - Same dataset as V5.
  - More specific bounding boxes.

- **Augmentations**
  - Grayscale: Apply to 25% of images.
  - Brightness: Between -25% and +25%.

Attempt 12 - DATASET garbageV5.3

- **Description**
  - Same dataset as V5.
  - Eliminate trash containers.
  - Merge trash classes and try only 2 separate trash classes (cardboard, trash) instead of one.

Attempt 13 - DATASET garbageV6

- **Description**
  - Link to Dataset on Roboflow.
  - Edit bounding boxes more specifically for Garbage V5.3.
  - Combine garbage bag, trash, 0, trash-plastic classes.
  - 2 classes: cardboard and trash.

Attempt 14 - DATASET garbageV5_XL

- **Description**
  - Link to Dataset on Roboflow.

  - Add 700 more images to garbageV5 by adding augmentations. The goal was to DUPLICATE data.

- **Augmentations Added:**
  - Outputs per training example: 2.
  - Brightness: Between -25% and +25%.
  - Blur: Up to 2.5px.
  - Noise: Up to 5% of pixels.
  - Bounding Box: Brightness: Between -25% and +25%.
  - Bounding Box: Blur: Up to 2.5px.
  - Bounding Box: Noise: Up to 5% of pixels.

Attempt 15 - DATASET garbageV5_final

- **Description**
  - Link to Dataset on Roboflow.
  - Same dataset as garbageV5 except:
  - Instead of duplicating the data by applying augmentations, simply add 500 more images. Edit annotations by hand.

Attempt 16 - DATASET garbageV10

- **Description:**
  - Link to Dataset on Roboflow.
  - Removed all images that are only tipped-over garbage cans with trash!

Fig. 4.5. Sample images from Attempt 16.

Attempt 17 - DATASET garbageV10

- **Description:**
  - same garbageV10 dataset as in Attempt 16.
  - Epochs increased to 150 from only 100.

[37]

Fig. 4.6. Bounding box annotations for some images.

The process of creating datasets for training demonstrates the complexity involved in handling real world data. Preprocessing and augmenting the data play a role in achieving exceptional model performance.

Now that the data has been improved, treated, and cleaned, it is important to understand how it will later affect the rest of the project (training and evaluation). The next section will provide an overview of the project structure explaining how each project component in the outline contributes to addressing all aspects of a machine learning model. Additionally it will outline the role that each document plays in the MLOps pipeline described in Section 5.1.

5. PROJECT STRUCTURE

The project's directory and file organization is structured as follows for the task of garbage detection using YOLO. The project structure and the files within the repository assisted in basic information setup, code and model file organization, documentation, data management, experimentation, testing and quality, visualization, versioning, and even CI/CD. Credit: Sngular Data & AI Team.

- **Makefile**: Common commands like make requirements to streamline development processes.
- **README.md**: First document developers should refer to - Overview of the project and its setup.
- **data**: Holds all datasets, both raw and processed, necessary for the model.
- **docs**: Includes documentation such as the Sphinx project and a model card template.
- **models**: Contains trained and serialized versions of the models.
- **notebooks**: Folder for Jupyter notebooks, especially the pipeline used for training, testing, and model version registration in mlflow.
- **references**: A place for data dictionaries, manuals, and other explanatory materials.
- **reports**: Generated analyses including graphical visualizations.
- **scripts**: Set of utility scripts from environment testing to actual inference.
- **requirements.txt**: Python dependencies required to run the project.
- **setup.py**: Makes the project pip-installable, facilites imports.
- **src**: Core source code directory which also includes a clone of the Ultralytics YOLOv5 repository (necessary for the object detection task).
- **tox.ini**: Settings for running tox, allows testing across multiple environments.

5.0.1. Outline - Gitlab Repository

Below is an outline of the Gitlab repository for the project structure described above. Its actual content remains private:

```
Object Detection with YOLO
|
+-- Makefile           <- Makefile with commands like ‘make requirements‘
+-- README.md          <- The top-level README for developers.
+-- data
|   +-- processed      <- The final, dataset for modeling.
|   |   |-- data.zip <- must be in a format accepted by YOLO
|   +-- raw            <- The original, immutable data dump.
|       |-- raw_data.zip
|
+-- docs               <- A default Sphinx project;
|   |-- MODEL_CARD_TEMPLATE.pdf
|
+-- models             <- Trained and serialized models
|
+-- notebooks          <- Jupyter notebooks.
|
+-- references         <- Data dictionaries, manuals, etc
|
+-- reports            <- Generated analysis as HTML, PDF, LaTeX, etc.
|   |-- figures        <- Generated graphics and figures
|
+-- scripts            <- Useful scripts.
|   |-- test_environment.py       <- Script to test environment.
|   |-- yolov5_train.py           <- script that trains new model.
|   |-- makefil_utils.py          <- Auxiliary Makefile python files
|   |-- pt_inference.py           <- Rrunning inference yolo.pt
|   |-- tflite_inference.py       <- Running inference yolo.tflite
|   |-- drawing.py                <- Script for drawing detection’s bboxes.
|
+-- requirements.txt   <- The requirements file
|
+-- setup.py           <- makes project pip installable
+-- src                <- Source code for use in this project.
|   |-- __init__.py               <- Makes src a Python module
|
|   +-- yolov5         <- Ultralytics YOLOv5 cloned repository.
|
+-- tox.ini            <- tox file with settings for running tox
```

Credit: Sngular Data & AI Team

5.1. MLOps Pipeline Integration

In addition to the project structure and repository, one of the most crucial parts of this project was an MLOps pipeline developed within Sngular, prior to the student's arrival as an intern. The pipeline is a Google Colab notebook that can be used for any object detection task within the company. It includes the following steps outlined in Figure 5.1 from architecture selection to model version registration and uploads to MLflow for managing the machine learning lifecycle.

The last step in the pipeline for Web Integration and the creation of a User Interface was completed separately using a different code repository. Docker was also used in construction of the UI to deploy the model as a web application, as well as Streamlit, a Python library that helps turn data scripts into web applications without requiring a background in web development.

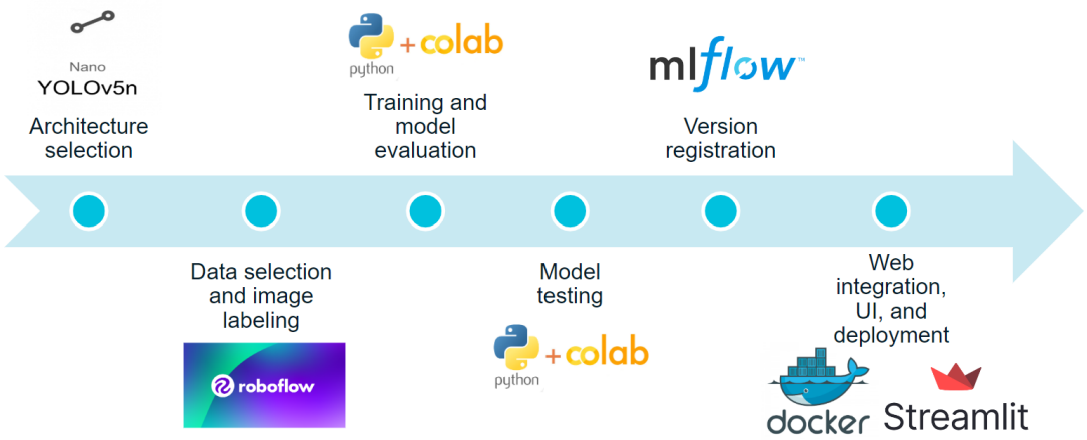


Fig. 5.1. MLOps pipeline used for dataloading, model training and testing, and model versioning

5.1.1. Pipeline Steps

To train a garbage detection model with YOLOv5 Nano and deploy it as a user interface, the following steps were taken in the pipeline:

0. Connect the Colab notebook to Gitlab Repository
1. Clone Ultralitics YOLO repository and install requirements
2. Install useful libraries
3. Set MLflow and Google Cloud environment variables
4. Assemble Data For Training (Pull Images from GoogleCloud after data gathering and image labeling
5. Train and Test Custom YOLOv5 model
6. Export trained model to other formats.

- Upload .tflite to MLflow
- Upload classes.txt to MLflow

7. Download Training Artifacts from MLFlow
8. Model Registry with MLFlow
9. Web integration, UI, and app deployment

Credit: Sngular Data & AI Team

6. EXPERIMENTS AND RESULTS

6.1. Detailed Analysis of YOLOv5 Nano Training Script for Garbage Detection

Training Setup

- Model Configuration: YOLOv5 Nano (`yolov5n.pt`).
- Dataset: Defined in `data.yaml`.
- Default hyperparameters on the UltraAnalytics Github Page in the following scripts
  `hyp.scratch-low.yaml` here,
  `hyp.scratch-med.yaml` here,
  `hyp.scratch-high.yaml` here
- Epochs: 100.
- Image Size: 320x320 pixels.
- Batch Size: 32 images.
- Classes: 1 class - trash.
- Optimizer: Stochastic Gradient Descent (SGD).
  Note: SGD is one traditional optimization method used in deep learning. It is effective for the most part, but newer methods like Adam, RMSprop, and more have become more popular recently for better and more adaptive learning rates and other features that can lead to faster convergence in some cases. Despite this fact, SGD, especially with momentum, still works well and can achieve high performance given the right hyperparameters and enough training time.

Training Command

```
# Train YOLOv5 nano
!python3 src/yolov5/yolov5_train.py --img 320 \
                                    --batch 32 \
                                    --epochs 100 \
                                    --data 'data.yaml' \
                                    --weights 'yolov5n.pt' \
                                    --hyp 'hyp.scratch-low.yaml' \
                                    --experiment_id '13' \
                                    --tags '{"dataset": "V6", "project": \
```

```
"GarbageDetector"}' \
--parameters '{"inputSize": "320", \
"model": "yolov5n.pt", \
"batch_size": "32", \
"no_epochs": "100", \
"hyp_location": "hyp.scratch-low.yaml"}' \
--run_name 'Run25'
```

Fig. 6.1. More sample bounding box annotations.

Hyperparameters

YOLOv5 Hyperparameters for High-Augmentation COCO Training
Ultralytics, AGPL-3.0 license [24]

`hyp.scratch-low.yaml` [24]

Parameter	Value
initial learning rate (`lr0`)	0.01
final OneCycleLR learning rate (`lrf`)	0.01
SGD momentum/Adam beta1 (`momentum`)	0.937
optimizer weight decay (`weight_decay`)	0.0005
warmup epochs (`warmup_epochs`)	3.0
warmup initial momentum (`warmup_momentum`)	0.8
warmup initial bias lr (`warmup_bias_lr`)	0.1

box loss gain (box)	0.05
cls loss gain (cls)	0.5
cls BCELoss positive weight (cls_pw)	1.0
obj loss gain (obj)	1.0
obj BCELoss positive weight (obj_pw)	1.0
IoU training threshold (iou_t)	0.20
anchor-multiple threshold (anchor_t)	4.0
focal loss gamma (fl_gamma)	0.0
HSV-Hue augmentation (hsv_h)	0.015
HSV-Saturation augmentation (hsv_s)	0.7
HSV-Value augmentation (hsv_v)	0.4
image rotation (degrees)	0.0
image translation (translate)	0.1
image scale (scale)	0.5
image shear (shear)	0.0
image perspective (perspective)	0.0
image flip up-down (flipud)	0.0
image flip left-right (fliplr)	0.5
image mosaic (mosaic)	1.0
image mixup (mixup)	0.0
segment copy-paste (copy_paste)	0.0

hyp.scratch-med.yaml [24]

Parameter	**Value**
initial learning rate (lr0)	0.01
final OneCycleLR learning rate (lrf)	0.1
SGD momentum/Adam beta1 (momentum)	0.937
optimizer weight decay (weight_decay)	0.0005
warmup epochs (warmup_epochs)	3.0
warmup initial momentum (warmup_momentum)	0.8
warmup initial bias lr (warmup_bias_lr)	0.1
box loss gain (box)	0.05
cls loss gain (cls)	0.3
cls BCELoss positive weight (cls_pw)	1.0
obj loss gain (obj)	0.7
obj BCELoss positive weight (obj_pw)	1.0
IoU training threshold (iou_t)	0.20
anchor-multiple threshold (anchor_t)	4.0
focal loss gamma (fl_gamma)	0.0
HSV-Hue augmentation (hsv_h)	0.015

HSV-Saturation augmentation (hsv_s)	0.7
HSV-Value augmentation (hsv_v)	0.4
image rotation (degrees)	0.0
image translation (translate)	0.1
image scale (scale)	0.9
image shear (shear)	0.0
image perspective (perspective)	0.0
image flip up-down (flipud)	0.0
image flip left-right (fliplr)	0.5
image mosaic (mosaic)	1.0
image mixup (mixup)	0.1
segment copy-paste (copy_paste)	0.0

hyp.scratch-high.yaml [24]

Parameter	**Value**
initial learning rate (lr0)	0.01
final OneCycleLR learning rate (lrf)	0.1
SGD momentum/Adam beta1 (momentum)	0.937
optimizer weight decay (weight_decay)	0.0005
warmup epochs (warmup_epochs)	3.0
warmup initial momentum (warmup_momentum)	0.8
warmup initial bias lr (warmup_bias_lr)	0.1
box loss gain (box)	0.05
cls loss gain (cls)	0.3
cls BCELoss positive weight (cls_pw)	1.0
obj loss gain (obj)	0.7
obj BCELoss positive weight (obj_pw)	1.0
IoU training threshold (iou_t)	0.20
anchor-multiple threshold (anchor_t)	4.0
focal loss gamma (fl_gamma)	0.0
HSV-Hue augmentation (hsv_h)	0.015
HSV-Saturation augmentation (hsv_s)	0.7
HSV-Value augmentation (hsv_v)	0.4
image rotation (degrees)	0.0
image translation (translate)	0.1
image scale (scale)	0.9
image shear (shear)	0.0
image perspective (perspective)	0.0
image flip up-down (flipud)	0.0
image flip left-right (fliplr)	0.5

image mosaic (`mosaic`)	1.0
image mixup (`mixup`)	0.1
segment copy-paste (`copy_paste`)	0.1

Model Architecture

- Number of layers: 270.
- Number of parameters: 1,766,623.
- Computational requirement: 4.2 GFLOPs for one image forward pass.
- Pre-trained weights from `yolov5n.pt` are used with 343 out of 349 items transferred.

Dataset Information

- Training set = 700 images (60 with empty annotations)
- Validation set = 100 images (2 with empty annotations)
- Test set = 200 images (4 with empty annotations)
- Single class - trash

Loss Curve - IMPORTANT

Ideally a loss curve showing the training and validation losses over the training epochs could have been useful to include here in order to understand when and how the models were converging for the training runs. However, this information was logged to MLflow and remains private to Sngular. It should be made clear that the student intern completed reserach on the project in June 2023 and at the time of the defense and writing of this project, access to the information is no longer available.

Hardware and Training Time:

- **Hardware:** Training was done on a GPU with CUDA, specifically a Tesla T4 with 15102MiB of RAM.
- **Training Time:** Each of the model runs were trained for just 100 epochs which took 0.557 hours, or 33.4 minutes.

Pretrained Checkpoints

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6

Fig. 6.2. Pretrained checkpoints for the YOLOv5 Nano and v5n6 release used for transfer learning. Taken from [24] at yolov5/README. Table Notes: All checkpoints are trained to **300 epochs** with default settings. Nano and Small models use hyp.scratch-low.yaml hyps, all others use hyp.scratch-high.yaml. mAPval values are for single-model single-scale on COCO val2017 dataset. Reproduce by `python val.py -data coco.yaml -img 640 -conf 0.001 -iou 0.65`. Speed averaged over COCO val images using a AWS p3.2xlarge instance. NMS times (1 ms/img) not included. Reproduce by `python val.py -data coco.yaml -img 640 -task speed -batch 1`. TTA Test Time Augmentation includes reflection and scale augmentations. Reproduce by `python val.py -data coco.yaml -img 1536 -iou 0.7 -augment`.

Model Size and Inference Time:

- **Model Size:** After the training, the optimizer was stripped from the model weights, which reduced the model size to 3.8MB. This was necessary for future deployment on edge-devices.
- **Inference Time:** Inference was performed at a rate of about 2 batches every 3.38 seconds, which is considered quite fast. The student researcher did not test the speed of each individual image.

6.2. Training Results

Results from each training run and the corresponding dataset used are presented below for all 17 attempts at training an accurate garbage detection model. Each training initiation was run for 100 epochs. Precision and mean Average Precision with an Intersection Over Union (IoU) threshold of 0.5 are the two metrics outlined in the results. Other metrics are not included here and have been kept private as information confidential to the Data & AI Team at Sngular.

The highest precision metric was obtained with the garbage V10 and garbageV5 datasets at 70.3 percent and 66.4 percent respectively. Mentors and tutors within Sngular to the student intern quoted this object detection task as the "most complicated yet," and that the suboptimal performance metrics still served as a PoC (Proof of Concept) sufficient enough to demonstrate to clients.

Attempt	Dataset	$mAP_{0.5}$	Precision
1	garbageV1	0.057	0.376
2	garbageV2	0.068	0.157
3	garbageV2.1	0.057	0.438
4	garbageV3	0.05	0.566
5	garbageV3.1	0.053	0.479
6	garbageV3.1.1	0.053	0.46
7	garbageV4	0.062	0.464
8	garbageV4.2	0.063	0.406
9	garbageV5	0.253	0.664
10	garbageV5.1	0.053	0.519
11	garbageV5 - 320x320	0.256	0.58
12	garbageV5.3	0.244	0.625
13	garbageV6	0.228	0.536
14	garbageV5 XL	0.128	0.464
15	garbageV5 final	0.28	0.639
16	garbageV10	0.236	0.703
17	garbageV10 + 600 images	0.225	0.635

TABLE 6.4. LIST OF DATASETS USED IN VARIOUS ATTEMPTS WITH THEIR CORRESPONDING $MAP_{0.5}$ AND PRECISION.

Potential Improvements:

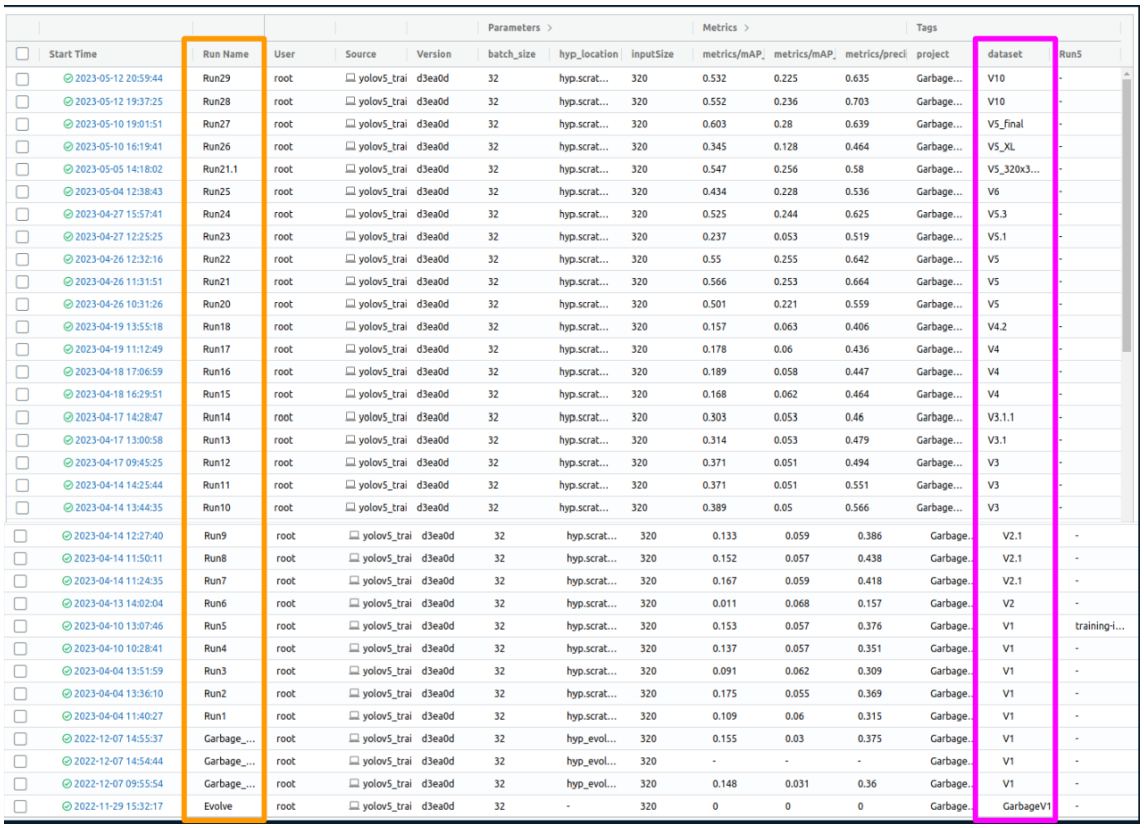
Increasing training data and number of epochs would probably improve model performance, and of course using a larger YOLOv5 model variant (small, medium, large, or extra-large) might also improve accuracy, but would not be the best option for use on edge-devices.

More training runs along with their hyperparameters, model versions, and other metrics such as box loss, and mAP with an IoU threshold of 0.95 can be found in MLflow. Figure 6.3 shows a brief snapshot of the interface in MLFlow that the student had access to while conducting the research project.

6.3. Evaluation

6.3.1. Testing and Validation Results

Out of 1000 images in the garbageV10 dataset, visual inferences were run on a validation set of 100 images and a test set of 200 images. Sample images and the model's inference for Attempt 17 with the garbageV10 dataset can be seen below in Figures 6.2 and 6.4 respectively. There is much improvement still to be made as the performance metric of interest (precision) is still suboptimal.



					Parameters			Metrics			Tags		
Start Time	Run Name	User	Source	Version	batch_size	hyp_location	inputSize	metrics/mAP_	metrics/mAP_	metrics/preci	project	dataset	Run5
2023-05-12 20:59:44	Run29	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.532	0.225	0.635	Garbage...	V10	-
2023-05-12 19:37:25	Run28	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.552	0.236	0.703	Garbage...	V10	-
2023-05-10 19:01:51	Run27	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.603	0.28	0.639	Garbage...	V5_final	-
2023-05-10 16:19:41	Run26	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.345	0.128	0.464	Garbage...	V5_XL	-
2023-05-05 14:18:02	Run21.1	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.547	0.256	0.58	Garbage...	V5_320x3...	-
2023-05-04 12:38:43	Run25	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.434	0.228	0.536	Garbage...	V6	-
2023-04-27 15:57:41	Run24	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.525	0.244	0.625	Garbage...	V5.3	-
2023-04-27 12:25:25	Run23	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.237	0.053	0.519	Garbage...	V5.1	-
2023-04-26 12:32:16	Run22	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.55	0.255	0.642	Garbage...	V5	-
2023-04-26 11:31:51	Run21	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.566	0.253	0.664	Garbage...	V5	-
2023-04-26 10:31:26	Run20	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.501	0.221	0.559	Garbage...	V5	-
2023-04-19 13:55:18	Run18	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.157	0.063	0.406	Garbage...	V4.2	-
2023-04-19 11:12:49	Run17	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.178	0.06	0.436	Garbage...	V4	-
2023-04-18 17:06:59	Run16	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.189	0.058	0.447	Garbage...	V4	-
2023-04-18 16:29:51	Run15	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.168	0.062	0.464	Garbage...	V4	-
2023-04-17 14:28:47	Run14	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.303	0.053	0.46	Garbage...	V3.1.1	-
2023-04-17 13:00:58	Run13	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.314	0.053	0.479	Garbage...	V3.1	-
2023-04-17 09:45:25	Run12	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.371	0.051	0.494	Garbage...	V3	-
2023-04-14 14:25:44	Run11	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.371	0.051	0.551	Garbage...	V3	-
2023-04-14 13:44:35	Run10	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.389	0.05	0.566	Garbage...	V3	-
2023-04-14 12:27:40	Run9	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.133	0.059	0.386	Garbage...	V2.1	-
2023-04-14 11:50:11	Run8	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.152	0.057	0.438	Garbage...	V2.1	-
2023-04-14 11:24:35	Run7	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.167	0.059	0.418	Garbage...	V2.1	-
2023-04-13 14:02:04	Run6	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.011	0.068	0.157	Garbage...	V2	-
2023-04-10 13:07:46	Run5	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.153	0.057	0.376	Garbage...	V1	training-i...
2023-04-10 10:28:41	Run4	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.137	0.057	0.351	Garbage...	V1	-
2023-04-04 13:51:59	Run3	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.091	0.062	0.309	Garbage...	V1	-
2023-04-04 13:36:10	Run2	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.175	0.055	0.369	Garbage...	V1	-
2023-04-04 11:40:27	Run1	root	yolov5_trai	d3ea0d	32	hyp.scrat...	320	0.109	0.06	0.315	Garbage...	V1	-
2022-12-07 14:55:37	Garbage_...	root	yolov5_trai	d3ea0d	32	hyp_evol...	320	0.155	0.03	0.375	Garbage...	V1	-
2022-12-07 14:54:44	Garbage_...	root	yolov5_trai	d3ea0d	32	hyp_evol...	320	-	-	-	Garbage...	V1	-
2022-12-07 09:55:54	Garbage_...	root	yolov5_trai	d3ea0d	32	hyp_evol...	320	0.148	0.031	0.36	Garbage...	V1	-
2022-11-29 15:32:17	Evolve	root	yolov5_trai	d3ea0d	32	-	320	0	0	0	Garbage...	GarbageV1	-

Fig. 6.3. All 29 training runs recorded in MLflow over the 17 different datasets.

!! IMPORTANT !!

It should be stated that performance metrics on the test set are not highlighted in this section and will be kept private as information confidential to the Data & AI Team at Sngular. **The primary focus of the project in the company after achieving the 70 percent precision threshold was to develop the User Interface outlined in the next section**. The user interface would make it possible for clients of the company to view and interact with the product. The model software was tested and further improved after the end of the duration of the student's internship.

6.3.2. Exploring the Model's Output: Running Inference

The YOLOv5 inference script, detect.py, was used for running inference on the constructed garbage detection model. Uploaded to MLflow upon training completion and re-downloaded in the .pt (PyTorch) format, this was done on the final garbage detection model by running:

```
#Inference:
!python src/yolov5/detect.py --weights models/
artifact_downloads/weights/best.pt \
--source data/processed/valid/images/c--339 _jpg.rf.8
    efe195a481896570e3893496337ee78.jpg \
```

Fig. 6.4. Sample images from the garbageV10 dataset which achieved the highest precision metric (70.3 percent) out of all training runs performed as part of the project.

Fig. 6.5. Sample images from attempt 9 with the garbageV5 dataset which achieved the second highest precision metric (66.4 percent) out of all training runs performed as part of the project.

```
5  --save-txt --save-conf --conf-thres 0.8 --max-det 10 --iou-thres
       0.45
```

The pretrained model weights file and the image file on which to run the inference are specified and the bounding boxes and their confidence scores are saved as text files. A 0.8 confidence threshold, a maximum number of detections per image (10), and the IoU threshold for non-maximum suppression (0.45) is set. For video inference, the process is identical and produces a .mp4 downloadable file with the model inference bounding boxes for trash.

Inference on the Test Set

Inference on the validation set of 200 images out of the total 1000 is performed by running the script:

```
1  from scripts.pt_inference import run_inference
2  from scripts.drawing import draw_boxes, explain_detections
3  import os
4  import cv2
5  from PIL import Image
6  import numpy as np
7  import yaml
```

Fig. 6.6. Sample inference on an image in the validation set of 100 images out of 1000 for the final ML model. Serves as a PoC to clients.

```
8
9   # Load classes from data.yaml into a list
10  CLASSES_PATH = "/content/garbage-detection/src/yolov5/data/data.yaml
        "
11  data_yaml = open(CLASSES_PATH)
12  parsed_data_yaml = yaml.load(data_yaml, Loader=yaml.FullLoader)
13  class_names = parsed_data_yaml["names"]
14
15  pt_path = '/content/garbage-detection/models/artifact_downloads/
        weights/best.pt'
16  valid_image_dir = '/content/garbage-detection/data/processed/test/
        images'
17  valid_label_dir = '/content/garbage-detection/data/processed/test/
        labels'
18
19  # Get a list of the image files in the validation directory
20  valid_image_files = os.listdir(valid_image_dir)
21
22  for image_file in valid_image_files:
23      # Load the image
24      image_path = os.path.join(valid_image_dir, image_file)
25      image = cv2.imread(image_path)
26
27      # Load the label
28      label_path = os.path.join(valid_label_dir, image_file.replace('.
            jpg', '.txt'))
29      with open(label_path, 'r') as f:
30          label = f.read()
31
32      boxes, scores, classes = run_inference(image, pt_path)
33
```

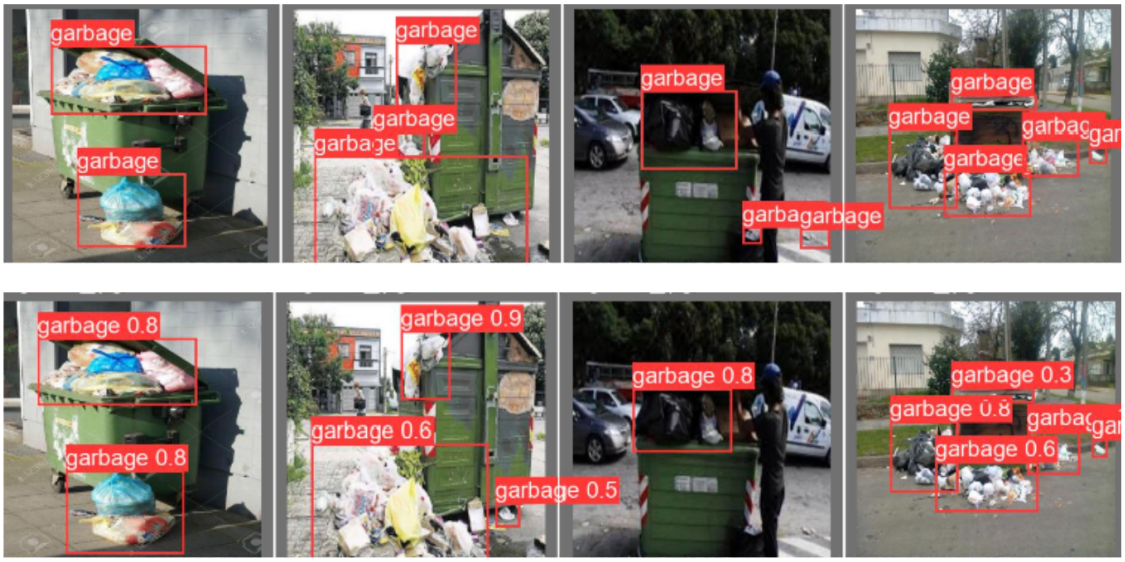


Fig. 6.7. Real image labels on 4 images from the test set (top) compared to the model's bounding box predictions on the same 4 images (bottom).

```
    input_images = Image.open(image_path)
    img_array = np.asarray(input_images)

    frame = draw_boxes(
        img_array,
        boxes,
        scores,
        classes,
        class_names,
    )
    det_list = explain_detections(classes, class_names)

    img = Image.fromarray(frame, 'RGB')
    display(img)

    print("Analyzing image:", image_file)
```

where `scripts.pt_inference` is the script from the YOLOv5 Github used to run inference of an image on a PyTorch YOLO model.

`scripts.pt_inference` is provided below for context:

```

## Module to run inference of an image on a -pytorch yolo model


#Import libraries
import numpy as np
import torch

```

```
"""
Pytorch (.pt) Detector Module
==============================
Module containing the Detector class used for running inference on
Tensorflow Lite modules.
"""

def run_inference(img_path: str, pt_model_path ):
    """Run model inference

    Args:
        img_path (str): path to image that we want to run prediction
            on

    Returns:
        boxes: bounding boxes
        scores: confidence level of detected objects
        classes: classes detected
    """
    model = torch.hub.load('src/yolov5','custom', path=pt_model_path
        ,force_reload=True,source='local')
    results = model(img_path)

    #Results
    results_xyxy = results.xyxy[0][:,:4].cpu().numpy().astype(int)
    results.pandas().xyxy[0]  # img predictions (pandas)

    boxes = np.transpose(np.array([results_xyxy[:,1],results_xyxy
        [:,0],results_xyxy[:,3],results_xyxy[:,2]]))
    scores = results.xyxy[0][:,4].cpu().numpy()
    classes = results.xyxy[0][:,5].cpu().numpy().astype(int)

    return boxes, scores, classes
```

The primary objectives of this study that have been completed so far are:

- *To use version 5 of the YOLO algorithm and adapt it to efficient garbage detection in real-time.* This has been done effectively as can be seen in the following video: See the live video inference on city streets.

- *To solve the model convergence issue encountered during training.* This has been achieved by training the model over multiple datasets with achieved convergence and performance metric stability.

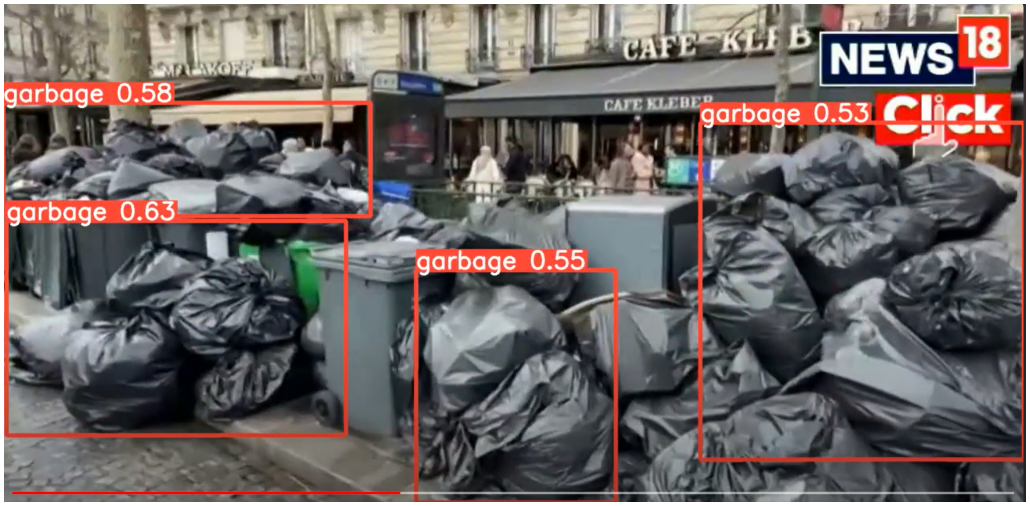


Fig. 6.8. Final garbage detection model inference over video stream on Paris city streets. Credit: YouTube: Emilie Naples

Fig. 6.9. Final garbage detection model inference over video stream on Paris city streets. Frame 2. Credit: YouTube: Emilie Naples

7. UI SYSTEM IMPLEMENTATION

7.1. UI Design and Development

Once a final garbage detection model was developed, the next goal was to allow users and potential clients of Sngular to interact with the model via web application. A user interface (UI) for the final garbage detection model developed by the student was made possible by cloning a previously existing repository belonging to the DATA & AI TEAM at Sngular. The steps below are an overview of how the project structure for the Steamlit app was strutured.

7.1.1. Project Structure for the Streamlit App

Credit: Sngular Data & AI Team

```
streamlit/
|
|-- .gitkeep
|-- README.md                    <- readme file
|-- requirements.txt             <- version requirements
|
|-- .streamlit/                  <- folder with password
|
|-- assets/
|   |-- sample images/           <- provided to the user to try
|   |-- image1.png               <- Sngular logo
|   |-- image2.png
|   |   ... [other .png files]   <- other images for UI
|
|-- app/
|   |-- models/                  <- garbage detection model stored here
|   |   |-- .gitkeep
|   |
|   |-- utils.py
|   |-- style.css                <- style the interface
|   |-- main.py                  <- structure for the app
|   |-- constants.py             <- image and directory paths
|   |-- body.py                  <- implementation
|   |-- __init__.py
```

7.1.2. Steps to Creating the UI

Credit: Sngular Data & AI Team

Outlined in this section is a template and steps followed by the student researcher to create the Streamlit app for deployment of the custom YOLOv5 garbage detection model. It has been used in deploying the web app for the garbage detector:

1. The **model must be in .tflite format** and stored in the `models` folder.
2. **Select an image for the body of the app** and store in the folder `streamlit>assets>` and save its path inside the `BODY_IMG_PATH` variable in `streamlit>app>constants.py`.
3. **Select 7 sample images** and save them to `streamlit>assets>images>`. The images chosen show the capacity of the model and works very well as an inference over them. A descriptive name was then given. For example, since the model detects garbage, some sample images for the demo were:
   - garbage bins
   - sample trash
   - sidewalk trash
   - dirty streets
   - last trash sample
4. Edit: `SAMPLE_IMG_PATHS` and `SAMPLE_CLASSES` variables in `streamlit>app>constants.py` add the sample images' paths to `SAMPLE_IMG_PATHS`. The `SAMPLE_CLASSES` variable in the same file should be a list of names describing the images.
5. Check the paths in `streamlit>app>constants.py` are well defined.
6. Customize the demo.

7.1.3. User Experience: Use Cases

Credit: Sngular Data & AI Team

1. The user reads about the solution and can choose to use it with some sample images in the "Test our results" section or with an image from their computer in the "Try it yourself!" section (See Figures 7.2 and 7.3).
2. **Test our results** Section:
   (a) *"Select image with the expander:"* when the user opens the expander, they will see a list of names describing an image. By clicking into that name, the user will select the image to process.

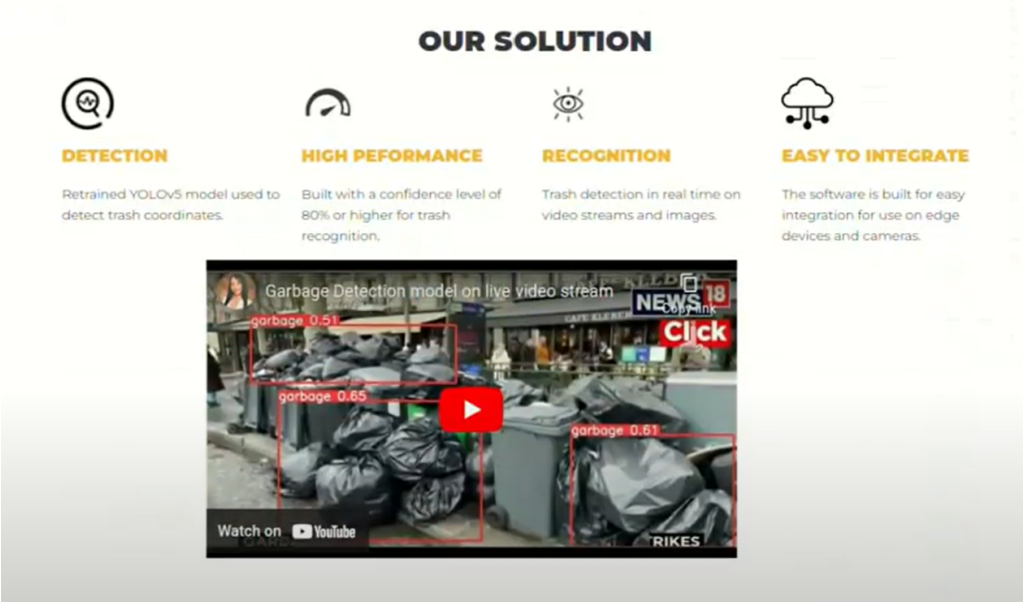


Fig. 7.1. Opening page on the web app for the garbage detector that shows model inference over a video stream. The video can be found here.

(b) *"Advanced settings" button:* Before clicking on "Try random image", the user can select some advanced settings by pressing the "Advanced settings" button. This will unfold an expander and show:

i. "Select min detection confidence" slider: the objects will be detected with, at least, the confidence value selected.

ii. "Select min confidence" slider: bounding box overlap allowed.

(c) *"Detect whatever" button:* When clicked, the selected will be processed and the objects will be detected. The results will be shown as follows:

i. On the left, the selected image will be shown, along with the bounding boxes, labels, and confidence of the detections.

ii. On the right, a summary of the detections will be displayed. In addition, an email is provided for reporting bugs.

3. **Try it yourself!** Section:

(a) The user will browse an image (.png, .jpg, or .jpeg) on their computer and by clicking on the "Detect whatever" button and selecting the desired advanced settings, the same actions as in the previous section will be performed.

The following objective has been completed:

- To construct a user interface so that clients and other users can interact with the model as a fine-tuned garbage detector.

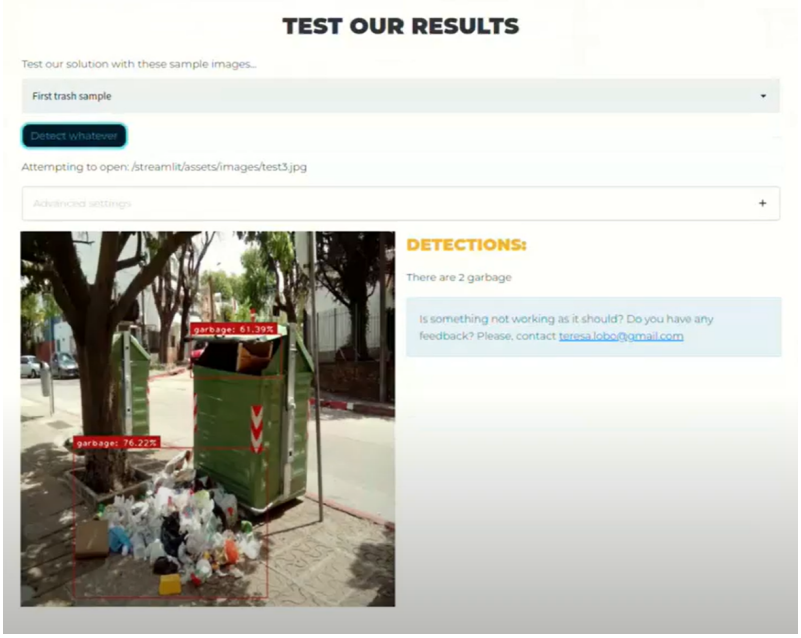


Fig. 7.2. Part of the web app where the user can select from a list of photos in the sample images folder to try out the garbage detection model.

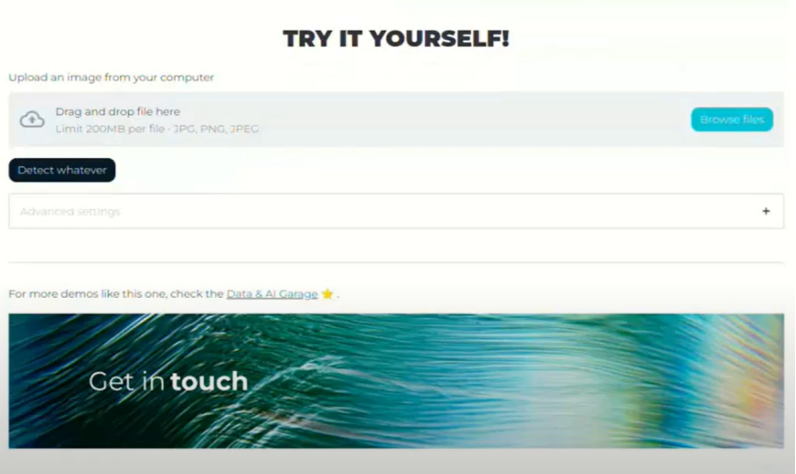


Fig. 7.3. The user can interact with the garbage detector by uploading his own photos to the interface as well.

8. INTEGRATION ON EDGE DEVICES AND CAMERAS

This section was part of this project's investigation, and although not fully implemented on actual cameras and edge devices, it should be made clear that this is the end goal of the research project.

8.1. Camera Setup and Calibration

Setup

For a YOLOv5 model used for garbage detection, the camera would be the eyes of a system that detects garbage and sends location metadata back to a centralized database for efficient trash pick-up. Detection quality, camera setup, and calibration would be important steps. Installation of the necessary software packages and dependencies on the camera or edge device would allow the camera to be compatible with the garbage detection model. The device should also be connected to the camera's data stream so that it receives, processes, and sends data correctly.

8.2. Camera and Edge Device Integration

The model should be capable of real-time analysis of video streams from the camera or cameras. A demo can be found in the YouTube video link found in the previous chapter and Section 6.2.1. The edge device should be able to send detection results and image metadata back to a centralized database.

Model Format

The model developed by the student is exported in PyTorch (`.pt`) and TensorFlow formats automatically, but other formats might be necessary for edge computing such as ONNX (`.onnx`) and TensorFlow Lite (`.tflite`) depending on camera or edge-device hardware.

8.3. Image Metadata Collection and Storage

The data collected by the model (for example, the amount of waste in different locations) should be sent in real-time to a central database or cloud service.

Metadata Extraction could also be implemented by a method like timestamps and geolocation from the video stream. A demo developed by the student using the Python library `'folium'`, which builds on the JavaScript library `Leaflet.js` was used to create the interactive map in this screen-recorded video. A GeoJSON file with the geographical coordinates (longitude and latitude) contain the location where each image was taken. The GeoJSON features are looped through to add markers and an HTML string was created

that includes an image whose source is specified in the GeoJSON file. The map is then saved to an HTML where, when opened, displays the interactive map centered on Madrid showing all of the trash locations. The map is intended as a demonstration for the end goal of image metadata collection and usage for efficient garbage detection software deployed on cameras.

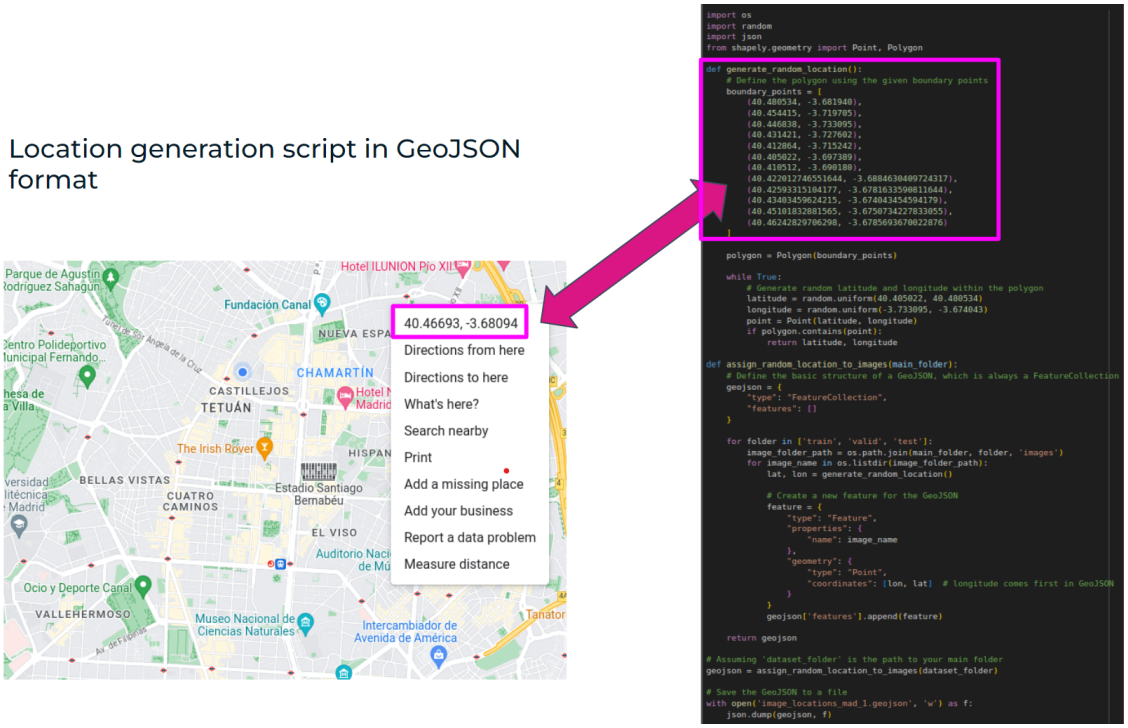


Fig. 8.1. Location generation script in GeoJSON format as a demonstration for image metadata collection and usage for an efficient garbage detection model integrated to cameras.

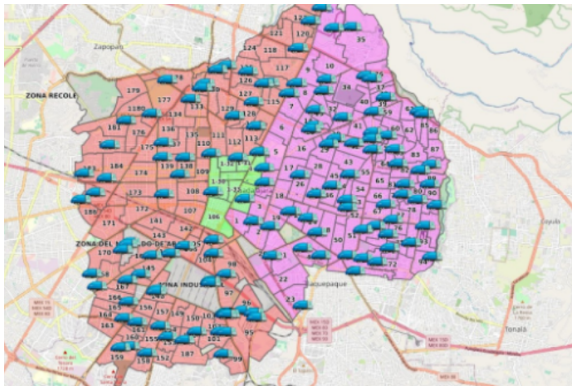


Fig. 8.2. Garbage pick-up deployment sample scenario for the usage of the garbage detector on cameras and edge devices.

A dynamic map that is updated in real time based on incoming data can then be used to monitor waste levels, identify hotspots, and deploy resources more effectively.

Data Pipeline

All in all, a data pipeline for running the entire process would be the most ideal. The

pipeline would capture video streams from the camera, perform inference using the YOLOv5 model on the camera, and send results back to a database.

In this section, the following objective for the project has been completed:

- To explore the implementation of the constructed garbage detection system for deployment on edge-devices and cameras to be used for optimized trash-pick up and location on city streets.

9. DISCUSSION

9.1. Interpretation of Results

Out of all the datasets used to train the garbage detection model, the 1000 images in the garbageV10 dataset with default hyperparameters hyp.scratch-med.yaml produced the best results with $mAP_{0.5}$ and precision at 0.236 and 0.703 respectively. Of course, these metrics should still be significantly improved, but were still sufficient to serve as a PoC (Proof of Concept) and for the construction of the Streamlit UI.

9.2. Brief Comparison with Existing Systems

As outlined previously, some of the garbage detection methods that exist today already include image-based garbage detection for drones. These drones are already identifying garbage with image detection algorithms, and once the garbage is detected, the device has the ability to use the same GPS coordinates to re-monitor the area multiple times [38]. A Real-Time Garbage Truck Supervision Based on Object Detection system has also been implemented directly with sensors mounted on a garbage truck to recognize garbage collection duration [39].

In conclusion, the proposed system in this paper is well on its way to becoming a fully-functional system for garbage detection on edge-devices and cameras for use on city streets.

The following objective has been completed:

- To compare the proposed solution with existing systems in terms of the overall application.

9.3. Future Limitations

A series of challenges still present themselves nonetheless when deploying the newly developed custom YOLOv5 trash detection model on cameras and edge-devices. Hardware constraints such as memory limitations, power consumption, and limited computational resources are yet a few. Changing backgrounds that come with the movement of a garbage truck, changing lighting conditions, and other vehicles, pedestrians, and more can also obstruct the view a camera might have of any trash in passing. Software compatibility, real-time processing, data storage and transmission without consuming too much bandwidth or storage are also some challenges and limitations. To combat such challenges, a combination of robust hardware, advanced software optimizations, careful model testing

under many weather conditions, and a reliable update strategy would be necessary.

9.4. Project Timeline

The garbage detection project was carried out from start date on March 27, 2023 to the project end date of completion on June 27, 2023. The entire development time from the very beginning phases of understanding necessary concepts to model deployment and the creation of a user interface took place over a 3 month period with exact dates and project phases outlined below.

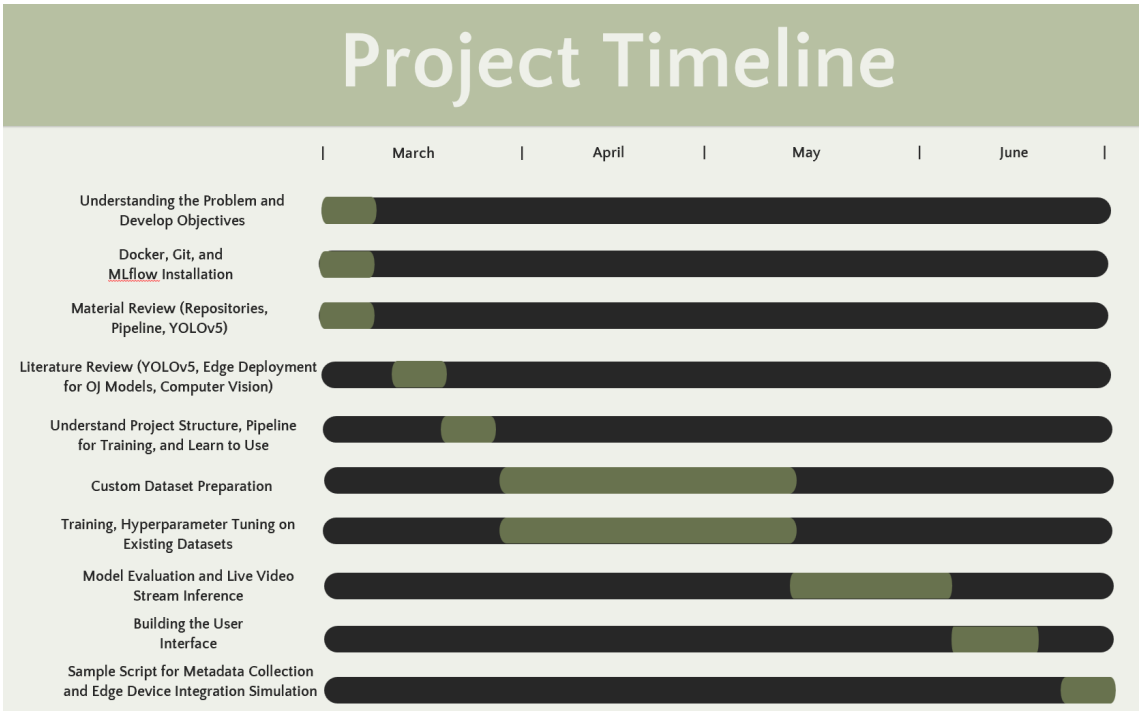


Fig. 9.1. Gantt chart representing the period of time during which the research project was carried out.

Project Stages

Project Stage	Timeline - Start	Timeline - End
Understanding the Problem and Study Objectives	2023-03-27	2023-03-31
Linux, Docker, Git, and MLflow Installation	2023-03-27	2023-03-31
Material Review (Repositories, Pipeline, YOLOv5)	2023-03-27	2023-03-31
Literature Review (YOLOv5, Edge Deployment for OJ Models, Computer Vision	2023-04-03	2023-04-07
Understand Project Structure, Pipeline for Training, and Learn to Use	2023-04-10	2023-04-14
Custom Dataset Preparation and Usage	2023-04-17	2023-05-17
Training, Hyperparameter Tuning on Existign Datasets	2023-04-17	2023-05-17
Model Evaluation and Live Video Stream Inference	2023-05-18	2023-06-06
Building the User Interface	2023-06-06	2023-06-17
Sample Script for Metadata Collection and Edge Device Integration Simulation	2023-06-20	2023-06-27
	2023-03-27	2023-06-27

Fig. 9.2. Exact dates for each project stage and when the time taken for development.

10. CONCLUSION AND FUTURE WORK

The primary objectives of the study have been completed:

- Used version 5 of the YOLO algorithm and adapt it to efficient garbage detection in real-time.
- Solved the model convergence issue after a trial version of the model software is constructed.
- Constructed a user interface so that clients and other users can interact with the model as a fine-tuned garbage detector.
- Explored the implementation of the constructed garbage detection system for deployment on edge-devices and cameras to be used for optimized trash-pick up and location on city streets.
- Compared the proposed solution with existing systems in terms of application and objective.

10.1. Summary of Contributions

An enormous thanks to the entire Sngular Data & AI Team, and specifically, the Computer Vision Chapter for providing the project structure for the garbage detection system, repository to be cloned for use in the Streamlit App UI, and MLOps pipeline.

Nerea Luis Mingueza: Conceived the original idea for the project, supervised all phases, and was responsible for administrative decisions, including hiring the student intern. Many thanks to her.

Teresa Lobo Alonso: Assisted in providing necessary resources and advised on methodology. Provided original YOLOv5, garbage detection, and Streamlit app code repositories for cloning and project use. Reviewed and edited the code and assisted the student intern throughout the learning process. Developed the pipeline used for training and testing the model and automatic model versioning to MLflow. Many thanks to her.

Emilie Naples (Author): Performed all data collection and preprocessing, model training and testing, version logging to MLFlow, and the construction and finalization of the Streamlit app user-interface with the provided resources.

10.2. Future Work

As the most difficult object detection task currently confronted by the Data & AI Team at Sngular, the garbage detection model can definitely be improved. The improvement of performance metrics, especially precision, should be further explored. Additionally, the chapter on *Integration on Edge Devices and Cameras* including the sections on Camera Setup and Calibration, Camera and Edge Device Integration, and Image Metadata Collection and Storage all make for future work to be implemented for improving and applying the garbage detection machine learning model developed in this project.

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, vol. 25, 2012, pp. 1097–1105.

[2] R. Singh. "Recent advances in modern computer vision." Accessed: 2023-05-08. (2019), [Online]. Available: https://towardsdatascience.com/recent-advances-in-modern-computer-vision-56801edab980.

[3] S. Majchrowska *et al.*, "Deep learning-based waste detection in natural and urban environments," *Waste Management*, vol. 138, pp. 274–284, 2022. DOI: https://doi.org/10.1016/j.wasman.2021.12.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0956053X21006474.

[4] C. Funk, A. Tyson, B. Kennedy, and C. Johnson. "Concern over climate and the environment predominates among these publics." Accessed: 2023-08-08. (2020), [Online]. Available: https://www.pewresearch.org/science/2020/09/29/concern-over-climate-and-the-environment-predominates-among-these-publics/.

[5] Y. K. Dwivedi *et al.*, "Climate change and cop26: Are digital technologies and information management part of the problem or the solution? an editorial reflection and call to action," *International Journal of Information Management*, vol. 63, p. 102 456, 2022. DOI: https://doi.org/10.1016/j.ijinfomgt.2021.102456. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401221001493.

[6] B. Fang *et al.*, "Artificial intelligence for waste management in smart cities: A review," *Environmental Chemistry Letters*, pp. 1–31, 2023, PMID: 37362015. DOI: 10.1007/s10311-023-01604-3.

[7] M. Bie, Y. Liu, G. Li, J. Hong, and J. Li, "Real-time vehicle detection algorithm based on a lightweight you-only-look-once (yolov5n-l) approach," *Expert Systems with Applications*, vol. 213, p. 119 108, 2023. DOI: https://doi.org/10.1016/j.eswa.2022.119108. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422021261.

[8] R. Kaur and S. Singh, "A comprehensive review of object detection with deep learning," *Digital Signal Processing*, vol. 132, p. 103 812, 2023. DOI: https://doi.org/10.1016/j.dsp.2022.103812. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200422004298.

[9] J. Solawetz, *Object detection: A comprehensive guide*, https://blog.roboflow.com/object-detection/, 2021. [Online]. Available: https://blog.roboflow.com/object-detection.

[10] "Bounding boxes in computer vision: Uses, best practices for labeling, and more." Accessed: 2023-05-09. (2023), [Online]. Available: https://www.ayadata.ai/blog-posts/bounding-boxes-in-computer-vision-uses-best-practices-for-labeling-and-more.

[11] M. Thoma, "A survey of semantic segmentation," *arXiv preprint arXiv:1602.06541v2*, 2016, Computer Vision and Pattern Recognition (cs.CV). [Online]. Available: https://doi.org/10.48550/arXiv.1602.06541.

[12] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: State of the art," *International Journal of Multimedia Information Retrieval*, vol. 9, no. 3, pp. 171–189, 2020. DOI: 10.1007/s13735-020-00195-x. [Online]. Available: https://doi.org/10.1007/s13735-020-00195-x.

[13] G. Boesch. "Computer vision applications." (2023), [Online]. Available: https://viso.ai/applications/computer-vision-applications/ (visited on 09/05/2023).

[14] "8 real-world applications of computer vision: A comprehensive guide." Accessed: 2023-09-05. (2023), [Online]. Available: https://www.augmentedstartups.com/blog/8-real-world-applications-of-computer-vision-a-comprehensive-guide.

[15] V. Flovik. "Deep learning based reverse image search for industrial applications." (2020), [Online]. Available: https://towardsdatascience.com/deep-learning-based-reverse-image-search-for-industrial-applications-33ba4b0d32c4 (visited on 09/05/2023).

[16] 3Blue1Brown. "But what is a neural network? | deep learning, chapter 1." Accessed: [May 20, 2023]. (2017), [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk.

[17] The University of Texas Health Science Center at Houston (UTHealth). "Chapter 15: The autonomic nervous system." Accessed: 2023-09-06. (2020), [Online]. Available: https://nba.uth.tmc.edu/neuroscience/m/s2/chapter15.html.

[18] B. Rossion, "Face perception," in *Brain Mapping*, A. W. Toga, Ed., Waltham: Academic Press, 2015, pp. 515–522. DOI: https://doi.org/10.1016/B978-0-12-397025-1.00037-3. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780123970251000373.

[19] R. Epstein, A. Harris, D. Stanley, and N. Kanwisher, "The parahippocampal place area: Recognition, navigation, or encoding?" *Neuron*, vol. 23, no. 1, pp. 115–125, 1999. DOI: https://doi.org/10.1016/S0896-6273(00)80758-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0896627300807588.

[20] L. E. van Dyck, R. Kwitt, S. J. Denzler, and W. R. Gruber, "Comparing object recognition in humans and deep convolutional neural networks-an eye tracking study," *Frontiers in Neuroscience*, vol. 15, 2021. DOI: 10.3389/fnins.2021.750639. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8526843/.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788. [Online]. Available: https://arxiv.org/abs/1506.02640.

[22] TensorFlow, *Tensorflow guide: Tensor*, Accessed: 2023-08-05, 2023. [Online]. Available: https://www.tensorflow.org/guide/tensor.

[23] M. Hussain, "Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection," *Machines*, vol. 11, no. 7, p. 677, 2023, Received: 30 May 2023 / Revised: 15 June 2023 / Accepted: 21 June 2023 / Published: 23 June 2023. DOI: 10.3390/machines11070677. [Online]. Available: https://doi.org/10.3390/machines11070677.

[24] Ultralytics, *Yolov5*, https://github.com/ultralytics/yolov5, GitHub repository, 2021.

[25] A. MITRA, "Detection of waste materials using deep learning and image processing," Accessed: 2023-08-20, M.S. thesis, California State University, San Marcos, California, 2020. [Online]. Available: https://scholarworks.calstate.edu/concern/theses/hq37vs75c.

[26] S. Vishnu *et al.*, "Iot-enabled solid waste management in smart cities," *Smart Cities*, vol. 4, no. 3, pp. 1004–1017, 2021, Received: 25 May 2021 / Revised: 1 July 2021 / Accepted: 5 July 2021 / Published: 14 July 2021. DOI: 10.3390/smartcities4030053. [Online]. Available: https://www.mdpi.com/2624-6511/4/3/53.

[27] L. Bunger, I. Chan, N. Cotto, and M. Sun, "Robotic waste sorting," Degree of Bachelor of Science in Mechanical Engineering, Robotics Engineering and Computer Science, Major Qualifying Project, Worcester Polytechnic Institute, 2021. [Online]. Available: https://digital.wpi.edu/concern/student_works/cj82k986z?locale=en.

[28] P. Achaliya, G. Bidgar, H. Bhosale, P. Dhole, and K. Gholap, "Drone based smart garbage monitoring system using computer vision," *International Journal of Creative Research Thoughts (IJCRT)*, vol. 8, 4 Apr. 2020. [Online]. Available: https://www.researchgate.net/publication/353971820_DRONE_BASED_SMART_GARBAGE_MONITORING_SYSTEM_USING_COMPUTER_VISION.

[29] H. Panwar *et al.*, "Aquavision: Automating the detection of waste in water bodies using deep transfer learning," *Case Studies in Chemical and Environmental Engineering*, vol. 2, p. 100 026, 2020. DOI: https://doi.org/10.1016/j.cscee.2020.100026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666016420300244.

[30] H. I. K. Fathurrahman, A. Ma'arif, and L.-Y. Chin, "The development of real-time mobile garbage detection using deep learning," vol. 7, pp. 472–478, Dec. 2021. DOI: 10.26555/jiteki.v7i3.22295. [Online]. Available: https://doi.org/10.26555/jiteki.v7i3.22295.

[31] X. Jiang, H. Hu, Y. Qin, *et al.*, "A real-time rural domestic garbage detection algorithm with an improved yolov5s network model," *Scientific Reports*, vol. 12, p. 16 802, 2022, Received: 19 April 2022 / Accepted: 21 September 2022 / Published: 07 October 2022. DOI: 10.1038/s41598-022-20983-1. [Online]. Available: https://doi.org/10.1038/s41598-022-20983-1.

[32] I. Gillani *et al.*, "Yolov5, yolo-x, yolo-r, yolov7 performance comparison: A survey," Sep. 2022, pp. 17–28. DOI: 10.5121/csit.2022.121602. [Online]. Available: https://doi.org/10.5121/csit.2022.121602.

[33] A. Bose. "What is mean average precision (map) in object detection?" Accessed: Aug 21, 2023. (2021), [Online]. Available: https://towardsdatascience.com/what-is-mean-average-precision-map-in-object-detection-8f893b48afd3.

[34] *Ai wiki - convergence*, Accessed: 07-09-2023. [Online]. Available: https://machine-learning.paperspace.com/wiki/convergence.

[35] *Precision and recall*, Wikipedia, The Free Encyclopedia; Accessed: 07-09-2023, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall.

[36] D. Shah, *Mean average precision*, Accessed: 07-09-2023, 2022. [Online]. Available: https://www.v7labs.com/blog/mean-average-precision.

[37] E. Naples. "Garbage detection project repository." Roboflow. (2023), [Online]. Available: https://app.roboflow.com/universidad-carlos-iii-de-madrid-i1yfs.

[38] *Trash-eating drone tackles water pollution*, Accessed: 07-09-2023, 2021. [Online]. Available: https://h2oglobalnews.com/trash-eating-drone-tackles-water-pollution/.

[39] H. Cheng, X. Zhang, Y. Gao, G. Xiao, B. Feng, and W. Chen, "A real-time garbage truck supervision and data statistics method based on object detection," *Wireless Communications and Mobile Computing*, vol. 2020, p. 8 827 310, 2020. DOI: 10.1155/2020/8827310. [Online]. Available: https://doi.org/10.1155/2020/8827310.