Aalto University
School of Electrical Engineering
Master of Communications Engineering and Data Science (CODAS)

Walid Lachhab

Deep Learning for Efficient Retail Shelf Stock Monitoring and Analysis

Master's Thesis
September, 2023

Thesis Supervisor:

Joni Pajarinen

Thesis Advisors:

Andrzej Duda
Guillaume Decor

Aalto University

Abstract

This thesis explores the automation of stock management in retail stores, with a specific focus on stores specializing in the sale of fruits and vegetables. Traditionally, these stores have relied on manual stock management methods, involving periodic inspections to maintain product availability. In response, this study proposes the application of Deep Learning techniques, particularly object counting models, to automate stock management.

The automation process comprises two key steps. Initially, a camera positioned above a shelf of fruits and vegetables captures an image, which is processed to identify boxes containing fruits and vegetables, along with their respective categories. Afterward, a Deep Learning counting model is employed to provide an estimation of the number of objects present within each box. These estimations can then be continuously monitored or subjected to analysis to optimize store operations.

The research encompasses four distinct data scenarios: supervised learning, semi-supervised learning, few-shot learning, and zero-shot learning. Within each scenario, existing object counting methods are evaluated using object detection and density estimation methodologies. The primary goals of this research are to establish an experimental setup for assessing object counting models across different learning frameworks, evaluate their performance in various scenarios, and analyze the practical strengths and limitations of these techniques in retail store environments.

Key findings from the study highlight the superior performance of YOLO models, especially YOLOv5, in supervised learning scenarios, striking a balance between speed and model size. In semi-supervised learning, the application of the Efficient-Teacher approach to YOLO models enhances performance with limited labeled data. Zero-shot learning, specifically the CLIP-Count method offering a balance between speed and acceptable error rates, is recommended for data-scarce environments with sufficient computational resources. While few-shot learning, represented by the SAFECount approach, remains as the last option due to its relatively higher error, and it is suggested for situations with limited data and computational resources.

Furthermore, our study reveals that improving the counting model's performance can be achieved through the removal of certain complex-shaped categories that present counting difficulties, such as grapes and hot peppers. Additionally, merging categories of fruits and vegetables with similar appearances emerges as a viable strategy for optimization.

Overall, this thesis offers practical insights into automating stock tracking in retail stores. It emphasizes the importance of selecting the right learning framework and model based on specific operational needs and constraints such as data availability, providing valuable guidance to improve stock management efficiency in diverse data scenarios.

Acknowledgement

This research was conducted as part of a project with COP AMACO, a French company specializing in the manufacturing of store furniture, aimed at improving store management and enhancing the overall customer experience in retail establishments.

I would like to extend my sincere gratitude to my supervisor, Joni Pajarinen, for his guidance and invaluable support throughout the course of this study. Additionally, I wish to express my gratitude to my advisors: Andrzej Duda from Grenoble-INP Ensimag and Guillaume Decor from COP AMACO, for their insightful mentorship, which contributed to the successful completion of this research.

September, 2023

Walid Lachhab

Abbreviations and Acronyms

CLIP	Contrastive Language-Image Pre-training
Detic	**Det**ector with **i**mage **c**lasses
FamNet	Feature, Affinity and Multi-dimensional assignment Network
Dino	Self-**di**stillation with **no** labels
MAC	Multifaceted Attention Counting
OWL-VIT	Open-World Localization by Vision Transformer
P2PNet	Point to Point Network
R-CNN	Region-Based Convolutional Neural Network
SAM	Segment Anything Model
SAFECount	Similarity-Aware Feature Enhancement Counting
YOLO	You Only Look Once

Table of Contents

List of Figures

List of Tables

1. Introduction

Retail stores have been around for a very long time, gradually expanding in size until we now encounter extensive chains of retail outlets. Among these, stores that specialize in selling fruits and vegetables at the retail level are emerging as a significant presence. Managing such stores presents its own set of challenges.
Traditionally, the management of these stores relies on **manual** labor, where workers periodically inspect the stock levels of each shelf. This routine is essential to ensure product availability and to prevent shortages. Hence, effective tracking of stock levels plays a pivotal role in the successful management of these types of stores.
An example illustrating a shelf stocked with fruits and vegetables is shown in *Figure 1*.

COP AMACO[1] aims to automate this process through the application of Deep Learning, as illustrated in *Figure 2*. A camera is positioned above the shelf to capture an image of its contents. In the initial stage, the focus is on identifying and extracting various boxes within the image. This can be achieved either manually by predefining specific locations, or automatically through the use of an algorithm.
Once the boxes are identified, the next step is to pass these images to a counting model, which will provide an estimate of the number of items contained within each box or image. It is important to note that the first step, involving box identification, falls outside the scope of this thesis. Instead, the focus will be on the **counting** part. The outcomes of this automated process can then be displayed for the store manager, who has the capability to receive alerts when products are nearing depletion, thereby preventing shortages. This automation offers significant advantages: it enables us to monitor stock levels automatically with minimal human intervention, resulting in faster and more cost-effective management. Moreover, it grants us the ability to gauge the rate at which specific product categories are sold, facilitating trend identification and more informed decisions when procuring supplies for the store.

In recent years, the field of Deep Learning has seen remarkable progress, giving rise to various techniques for object counting. These techniques range from object detection models **[1, 8, 17, 18]**, which identify objects coordinates within an image and subsequently tally their numbers, to density models **[2, 21, 22, 23]**, which aim to capture the probability density of objects within an image instead of individually counting them.
As is widely recognized, Deep Learning models heavily rely on data, and the acquisition of this data can be costly. Moreover, the ideal scenario of having ample labeled data for supervised learning is not always feasible. Fortunately, prior researchers have delved into scenarios where data availability varies. For instance, in zero-shot learning **[32, 36]**, they have addressed situations where no data is required except a textual description. While in few-shot learning **[29, 30]**, they have tackled cases with limited data. Additionally, in semi-supervised learning **[24, 25]**, researchers have explored situations where a reasonable amount of data is available, but only a small portion of it is labeled. Notably, both zero-shot and few-shot learning possess the advantage of accommodating a flexible number of classes without necessitating the creation of entirely new models.
Due to data constraints, we should carefully select a learning framework (supervised, semi-supervised, few-shot, or zero-shot) that is well-suited to our task.

[1] COP AMACO is a store furniture production company. *https://www.cop-amaco.fr*

However, to our knowledge, no previous study has assessed the performance of object counting models in both detection and density estimation across the four aforementioned frameworks especially given the recent release of many new approaches. Therefore, the primary objective of this thesis is to address the following **research questions**:

1. What experimental setup and evaluation criteria can be employed to assess the performance of state-of-the-art object counting models within different frameworks?
2. How do state-of-the-art counting models, operating within each framework, perform when evaluated using the established experimental setup and evaluation criteria?
3. What are the strengths and limitations of the assessed counting techniques, and how do these factors influence their applicability in retail store environments?
4. In what ways can the findings derived from this research guide the selection of the most suitable framework and counting approach to meet specific retail store needs?

In order to answer these questions, we will conduct a **benchmark** of existing counting approaches, focusing on the counting of fruits and vegetables in shelves across the four scenarios. Supervised, when we have access to an ample amount of labeled data. Semi-Supervised, where a dataset is available, but only a fraction of it is labeled. Few-shot, when facing a flexible number of categories represented by only a handful of examples. Zero-shot when no visual data is available; instead, we rely solely on textual descriptions while accommodating a flexible number of categories.

It is important to clarify that our study's **scope** is confined to the benchmarking of existing counting approaches. We do not undertake the development of new model architectures or provide detailed explanations of the inner workings of each presented model.

Consequently, the principal **contributions** of this thesis can be summarized as follows:

- Implementation of state-of-the-art object counting methods through detection and density estimation.
- Evaluation of the performance of these methods across four distinct data scenarios.
- Provision of a comparison of the strengths and limitations of these techniques

The **structure** of the remaining sections in this thesis is as follows: *Section 2* provides a background to facilitate comprehension of the methods used in this study. *Section 3* reviews relevant prior research related to our thesis. In *Section 4*, we introduce the selected approaches. The experimental setup and evaluation metrics are detailed in *Section 5*. *Section 6* presents the results, followed by their discussion in *Section 7*. Lastly, *Section 8* concludes the thesis by addressing the research questions, highlighting challenges, and suggesting potential avenues for future research.

Figure 1: Example of a fruit and vegetable retail shelf.[2]

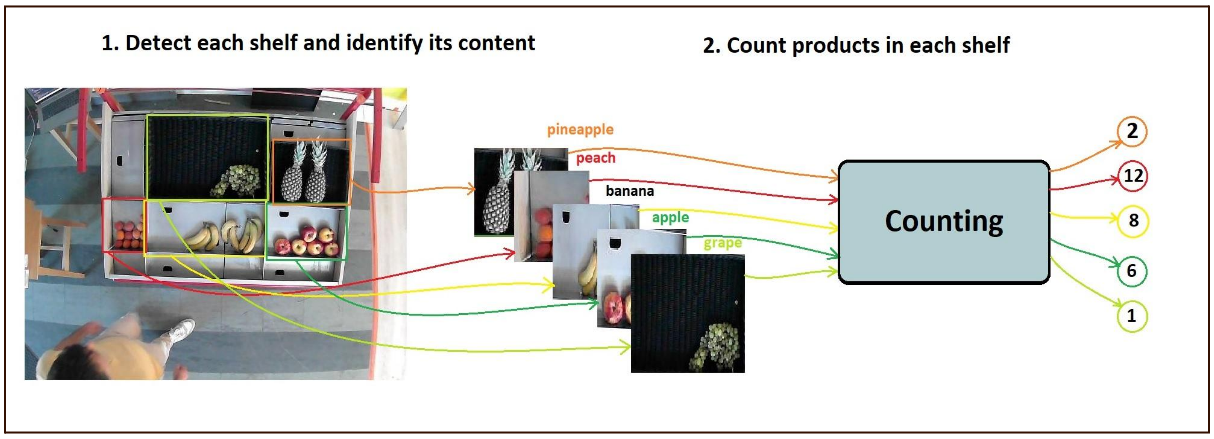

1. Detect each shelf and identify its content
2. Count products in each shelf
pineapple
peach
banana
apple
grape
Counting
2
12
8
6
1

Figure 2: Automation setup using deep learning: a camera placed above captures an image of the shelf, which is then processed to extract boxes with their corresponding categories. Afterward, a counting model is employed to predict final estimates.

[2] "Fruits and vegetables on shelves in supermarket Photos," Adobe Stock. *https://stock.adobe.com*

2. Background

In this section, we will cover fundamental concepts that will aid in understanding the methods used later. Specifically, we will explain how Deep Learning counting methods function through detection and density estimation. Subsequently, we will examine various learning paradigms tailored to different types of data.

2.1. Deep Learning and Computer Vision

2.1.1. Machine Learning

Machine learning is a process that develops algorithms capable of approximating unknown real-world mappings without explicit coding. This iterative approach relies on examples of data to enhance algorithm quality through a series of error corrections.

Machine learning broadly falls into three categories. Supervised Learning, where datasets consist of input-output pairs. The objective is to learn a function that accurately maps inputs to outputs. Unsupervised Learning, that deals with unlabeled data and aims to discover meaningful patterns or structures within it. Clustering is a common application. Reinforcement Learning, where agents interact with environments, making decisions to maximize cumulative rewards. This approach is prevalent in games and robotics.
It is noteworthy that certain tasks within unsupervised and reinforcement learning can also be framed as instances of supervised learning.

The key components of supervised machine learning include: the data, the model, and the loss function. The model represents a function mapping inputs from the input space (X) to outputs (y = f(x)) in the output space. Models have parameters, known as weights, which are not manually set but determined iteratively through Gradient Optimization. While the loss function quantifies the error between the model's predictions and actual outcomes, it guides the optimization process. This iterative process entails predicting labels for inputs, calculating prediction errors, and adjusting weights accordingly. As iterations progress, the model improves its predictions. *Figure 3* provides an illustration.

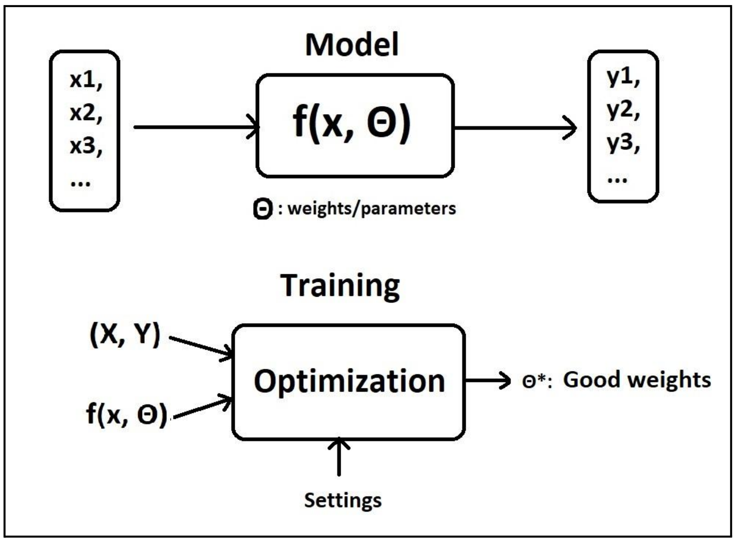


Figure 3: Illustration of supervised machine learning: modeling the input-output function, then optimizing its parameters to make good predictions through gradient optimization, involving the model, the data, and settings such as the loss function.

2.1.2. Deep Learning

Deep learning is a subfield of machine learning. It follows the same data, model, and loss function logic. What distinguishes deep learning is the use of neural networks, a type of functions that are well suited for complex tasks. It tries to simulate the working principle of brain neurons. A typical neural network, as depicted in *Figure 4*, comprises three essential components. The input layer takes in the input data, which can be represented as a vector or matrix. Hidden layers can be viewed as smaller functions within the network. Each layer receives the output from the preceding layer's neurons, applies transformations using its associated weights, and passes the result to the subsequent layer. The stacking of these layers contributes to the network's depth, making it more capable to learn complex patterns. The output layer produces the final prediction or label, serving as the network's ultimate decision-making stage.

Deep neural networks, constituting deep learning, feature multiple hidden layers, hence their name. The learning process within each layer involves the adjustment of weights, which is accomplished through gradient optimization.

During training, the network performs a forward pass to make predictions, calculates the error by comparing these predictions to the actual outputs using the chosen loss function, and subsequently updates the weights through a backward pass. This iterative weight adjustment process progressively enhances the network's ability to make accurate predictions.

2.1.3. Convolutional Neural Networks

Convolutional Neural Networks, or CNNs, represent a specialized subset of neural networks better suited for image processing tasks. They harness the concept of image filters, or kernels, to extract distinctive features from images. These kernels, such as the commonly used 3x3 matrix for edge detection, perform mathematical operations called convolutions. The result is a new image that accentuates specific aspects like edges. CNNs take this idea further by employing learnable kernels. Unlike predefined kernels, CNNs learn their kernel parameters using gradient optimization alongside network weights. This approach enables the network to adapt and extract more informative features from images, enhancing their utility compared to using raw images.

A typical CNN architecture, illustrated in *Figure 5*, encompasses three essential components. Convolutional layer with learnable kernels, that applies convolution operations, allowing the network to capture meaningful features from the input data. Activation function, that introduces non-linearity to the network, enabling it to model complex mappings effectively. Without them, the composition of linear functions would remain linear, limiting the network's capacity to learn intricate relationships. Pooling operator, which reduces the spatial dimensions of image features, promoting efficient processing and reducing the overall number of network weights. This contributes to faster computation. After traversing multiple CNN layers, the network extracts final image features. These features can be further processed using standard neurons, culminating in the prediction made at the final layer.

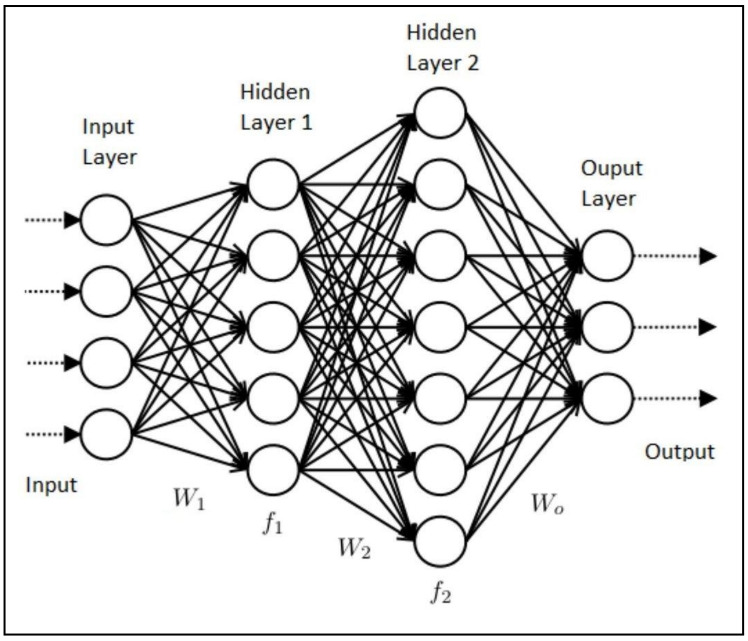


Figure 4: Illustration of the structure of a typical neural network: involving two essential processes. The forward pass, where neurons in each layer receive the output from the preceding layer and apply layer-specific weights to generate outputs for subsequent layers, culminating in the final output layer, where prediction-loss calculations occur. Subsequently, the backward pass involves weight adjustments across layers to optimize predictive performance.[3]

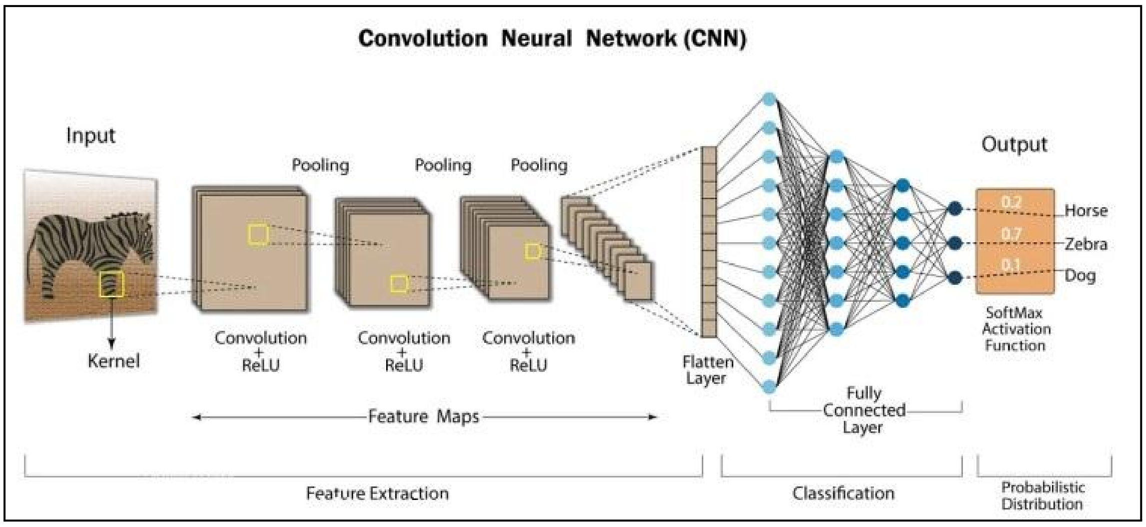


Figure 5: Application of CNN in image classification: the input image undergoes initial processing through convolutional layers, succeeded by activation and pooling layers, resulting in a reduction in size and an augmentation in the number of feature maps. These extracted features are then processed through multiple layers of neurons, enabling the generation of a final prediction providing the probability of the image belonging to each class.[4]

[3] JayeshBapuAhire, "The Artificial Neural Networks handbook: Part 1," Data Science Central, Aug. 2018, *https://www.datasciencecentral.com/*

[4] Kh. Nafizul Haque, "What is Convolutional Neural Network — CNN (Deep Learning)", Apr. 2023, *https://www.linkedin.com*

2.1.4. Counting by Object Detection

In contrast to image classification, where a single prediction pertains to the entire image, object detection is tasked with the identification of multiple objects within an image. This process involves providing coordinates for each detected object in the form of the top-left corner's (x, y) position, along with the width and height dimensions of the bounding box or rectangle encapsulating the object. Additionally, object detection furnishes valuable information such as the category id of each detected object and a confidence score, expressed as a probability ranging from 0 to 1. Consequently, an object detection model, as visualized in *Figure 6*, functions as a mapping that takes an image as input and generates output consisting of object coordinates (x, y, w, h) that define the bounding boxes, along with category ids and confidence scores for each detected object within the image. Counting objects through object detection involves the straightforward process of tallying the number of predictions or the number of distinct objects detected within the image.

2.1.5. Counting by Density Estimation

An alternative approach to object counting involves the use of density estimation models. These models function as mathematical functions that take an image as input, but instead of providing coordinates for each object, they generate a density map for the entire image. This density map takes the form of a 2D matrix, which serves as a visual representation of the probability distribution for the presence of objects. In regions where the likelihood of an object's existence is high, the values in the matrix approach 1, while areas with a lower likelihood exhibit values closer to 0.

Counting objects through density estimation can be achieved by integrating over the density map. This practical integration is performed through a summation of pixel values within the 2D matrix. Alternatively, we can employ contour detection algorithms to identify dense areas or spots and subsequently count the number of detected spots.
It is worth noting that the default counting method, both during training and inference, is the first approach involving summation. The second approach, which we have experimented with, has demonstrated its ability to yield superior results in numerous scenarios, although it may not consistently outperform the first approach. An illustration is shown in *Figure 7*.

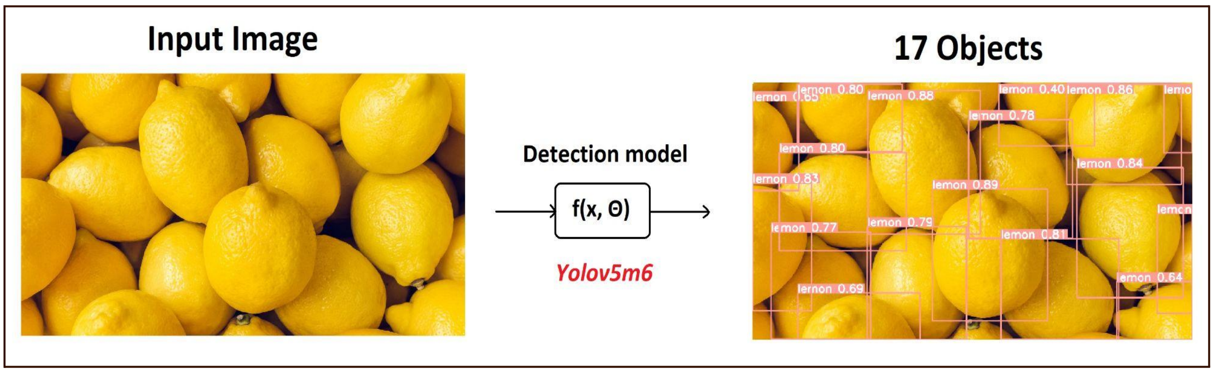


Figure 6: Object detection and counting example with YOLOv5 **[1]**: the detection model processes the input image and generates output containing the coordinates of each detected object, along with their corresponding category ids and associated confidence scores. These detected objects can subsequently be visualized or counted.

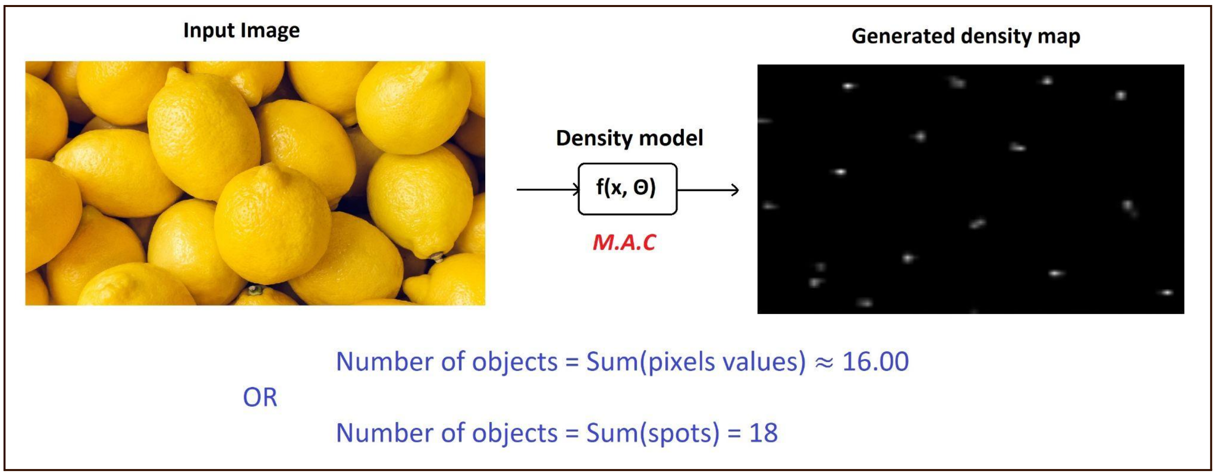

Input Image
Density model
f(x, Θ)
M.A.C
Generated density map
Number of objects = Sum(pixels values) ≈ 16.00
OR
Number of objects = Sum(spots) = 18

Figure 7: Object counting example through Density Estimation using MAC **[2]**: the density model processes the input image, resulting in a 2D probability density map that visualizes the likelihood of object presence. This density map can be employed for object counting through either summation or spots detection and counting.

2.2. Different Learning Paradigms

In the following axis, we provide insights on harnessing unlabeled data to improve the training instead of relying only on labeled data in semi supervised learning. We also explore how a model can make predictions on unseen classes in zero-shot and few-shot learning.

2.2.1. Semi-supervised Learning

Leveraging unlabeled data can be achieved through various methods, one of which is **pseudo-labeling**. This approach involves training the model using the labeled data while simultaneously making predictions on the unlabeled data and using these pseudo or simulated predictions in the training process. Although this may seem strange, it proves effective in enabling the model to generalize well to unseen data, particularly when augmentation techniques are applied. As demonstrated in **[3]**, training with pseudo-labels acts as a form of regularization that helps mitigate overfitting, ultimately leading to improved results.

2.2.2. Zero-shot & Few-shot Learning

Predicting outcomes for unseen data belonging to classes used during the training, poses a challenge in machine learning. The fundamental objective of machine learning is to equip models with the ability to generalize effectively to novel data instances. However, making accurate predictions for data belonging to entirely new, unforeseen categories represents an even greater challenge. Such scenarios frequently arise in practice, where either limited data is available for new categories or retraining the model for each new category is impractical.

One approach to illustrate the feasibility of making accurate predictions for new, previously unseen categories involves training a supervised image classification model on a predefined set of classes. Subsequently, the model's underlying architecture, often referred to as the 'backbone', is employed to extract embeddings or features from new data instances belonging to previously unencountered categories. Classification of these new instances is achieved by computing the distances between the instance's features and the features of each known category and assigning the instance to the category with the closest feature representation.

To put it into perspective, consider a model that has been trained to recognize various animal species. Now, if we encounter a new, unidentified animal species, rather than undergoing the laborious process of retraining the model from scratch, we leverage the model's existing knowledge of animal characteristics to infer the most likely category for the new species. This process is akin to saying, ‘This unfamiliar creature shares strong resemblances with known big cat species, so we classify it as a type of large feline’. In essence, this methodology underscores the potential for leveraging existing model knowledge to make informed predictions for entirely novel and unforeseen data instances.

To obtain the distinctive features of each class, several methods can be employed. One approach involves utilizing a text-based model, which translates a textual description of the class into a vector within the same feature space as that of the image features. An example of such a technique is CLIP **[4]**, which enables zero-shot classification. Alternatively, one can provide multiple images for each class. These images are then processed through the same underlying backbone architecture used for feature extraction, and their feature representations are averaged to create a prototype for each class, a method exemplified by the Prototypical Network **[5]**. This approach facilitates few-shot classification.

3. Related work

We recall that our objective is to assess state-of-the-art deep learning based counting methodologies in various data availability scenarios. This assessment aims to offer retail store managers valuable insights for enhancing store management by optimizing their stock tracking.

This section will offer an overview of relevant prior research related to our study, accompanied by pertinent discussions. Specifically, we will focus on existing deep learning techniques for object counting, which can be broadly categorized into two main approaches: object detection and density estimation. The selected methods will be presented in the subsequent section *(Section 4)*.

3.1. Existing Supervised Approaches

In recent years, deep learning-based object detection methods have attracted significant attention, categorized into three main groups. *To facilitate comprehension of the subsequent discussion, Figure 8 provides an illustrative representation.* The first, known as **two-stage** methods, was pioneered by R-CNN **[6]**, which exhibited promising results compared to non-deep learning approaches. R-CNN employed selective search to identify regions of interest in an image, followed by a CNN network for classification. However, this method suffered from slow processing speed, making it unsuitable for real-time applications. The introduction of Fast R-CNN **[7]** addressed this issue, delivering improved performance and faster predictions by replacing multiple CNN predictions with a single one. This streamlined approach utilized Region Of Interest (ROI) pooling for feature extraction, resulting in faster and more efficient object detection. Subsequently, Faster R-CNN **[8]** further improved speed and accuracy by replacing selective search with a neural network called the Region Proposal Network (RPN). This enhancement enabled multiple predictions per second, making it close to real-time usage. In contrast, **one-stage** methods, such as the YOLO series **[9]**, focus on achieving real-time object detection. Although slightly less accurate than two-stage models, YOLO introduced significant improvements with YOLOv2 **[10]**, incorporating batch normalization, anchor boxes, and multiscale training. Further advancements in YOLOv3 **[11]** and YOLOv4 **[12]**, featuring new loss functions, architectures, multiscale feature extraction, and augmentations, aimed to balance accuracy and speed. YOLOv5 **[1]** continued this trend, emphasizing ease of use and portability by adopting the PyTorch[5] environment with Python, departing from the Darknet[6] framework based on C. Subsequent iterations, YOLOv6 **[13]**, YOLOv7 **[14]**, and YOLOv8 **[15]**, focused on enhancing performance through novel architectures, loss functions, optimizers, and regularization techniques, positioning YOLO as a state-of-the-art solution for high-performance and real-time object detection. However, it is important to note that YOLO is not the sole representative of one-stage detectors. Other models like SSD **[16]** and RetinaNet **[17]** have also operated in this domain. While they may no longer be considered state-of-the-art, RetinaNet remains a notable option, appreciated for its acceptable performance when compared to other alternatives in the field. In addition to these models, **transformer**-based approaches emerged, inspired by the success of transformer networks in natural language processing. These models made their foray into computer vision tasks with Detr **[18]** and Deformable Detr **[19]**, which combine CNN networks with transformer networks for object detection. While these models performed well, they are resource-intensive and computationally demanding compared to other alternatives.

[5] PyTorch is a python deep learning framework. *https://pytorch.org*
[6] Darknet is a C based deep learning framework. *https://pjreddie.com/darknet/*

Turning our attention to density models also known as crowd counting models which garnered substantial research interest as well. These models were first introduced in **[20]**, where they transformed the task of counting individual elements within an image into an integral calculation over the probability (density map) of object presence. Subsequent efforts were dedicated to enhancing the predictive capability of these models, particularly in scenarios with highly dense images. This endeavor led to various innovations, including the utilization of multi-scale predictions in MCNN **[21]**, which aimed to reduce counting errors by employing distinct CNN filter kernels at different scales within the network. Additionally, the adoption of the U-Net architecture, renowned for image-to-image tasks like segmentation, was applied in models such as W-Net **[22]**, resulting in better quality density maps and further reductions in counting errors. Further strides in this direction introduced a purely point-based framework in P2PNet **[23]**, which contributed to cleaner density maps and lower counting errors. Furthermore, the fusion of transformers and attention mechanisms with CNN networks was leveraged to enhance the performance of crowd density models, exemplified by the approach taken in MAC **[2]** *(Figure 9)*. The latter two methods are regarded among the state-of-the-art.

3.2. Existing Semi-Supervised Approaches

Over the last few years, there has been notable interest in semi-supervised training of object detection models. The strategies employed to harness unlabeled data for enhanced generalization can be succinctly summarized as follows. Student-**Teacher** Setup, this entails the concurrent training of two models, where the student aims to replicate the predictions of the teacher. The teacher model can be obtained through various means, including methods like Exponential Moving Average (EMA) applied to the student's weights over multiple training iterations. **Consistency**, in this approach, perturbations are introduced to the data, and the models are tasked with producing similar predictions for perturbed versions of similar images. **Pseudo Labeling**, as previously described in *(Section 2.2.1)*, pseudo labeling enables the utilization of unlabeled data during training. After initial training on labeled data, the model is employed to make predictions on unlabeled data, with some of these predictions considered as new labels. Subsequently, the model is retrained on this augmented dataset.

These innovative techniques have found application in recent state-of-the-art approaches. For instance, the Consistent-Teacher method **[24]** effectively combined the teacher-student setup with pseudo labeling and data augmentation, along with a specialized loss function tailored to the RetinaNet model. This approach yielded superior results compared to the baseline model trained solely on labeled data. Similarly, the Efficient-Teacher method **[25]** *(Figure 11)*, another recent entrant achieving state-of-the-art results, adopts the student-teacher framework with EMA updates. It incorporates both strong and weak data augmentations as perturbations to facilitate robust generalization. This approach was applied to the YOLOv5 model, resulting in state-of-the-art performance. While this technique is adaptable to various YOLO model variants, YOLOv5 was the default choice in this study.

Furthermore, these methodologies have been extended to semi-supervised density models, as demonstrated in **[26]**. In this context, student and teacher networks, combined with EMA updates and perturbations, were employed to advance the performance of semi-supervised density models.

3.3. Existing Few-Shot Approaches

As previously discussed, few-shot techniques are specifically designed for scenarios where the number of categories is flexible, and new categories are represented by only a few examples. Various approaches can be employed to address this challenge. For instance, in the domain of few-shot object detection, models like DeFRCN **[27]** and Meta-DETR **[28]** utilize meta-learning and fine-tuning strategies. They incorporate the concept of a query set and support set during inference, in addition to the training dataset. This enables these models to quickly adapt and generalize to new categories through fine-tuning. However, it is worth noting that these approaches may not be practical in a retail store context, as fine-tuning typically necessitates domain knowledge, debugging skills, and a training device that store managers may not possess.
In contrast, few-shot density-based counting models offer a more practical solution, as they do not require fine-tuning. Instead, these models take two inputs during both training and inference: the query (the object to be counted) and the support images (the reference shots). They employ a principle known as "matching" *(Figure 10)*. A shared CNN neural network backbone is applied to both inputs to extract meaningful features. Subsequently, these features are projected into the same feature space, and the features of the support images are treated as kernels for CNN operator layers, which are then applied to the features of the query image. Further normalization layers and additional operations are employed to refine the localization of support images features within the query image. This process aids in generating the final density map, which is used for counting through summation.
The concept of matching is effectively employed in state-of-the-art few-shot density counting models such as FamNet **[29]** and SAFECount **[30]**.

3.4. Existing Zero-Shot Approaches

Since the past year, there has been a significant interest in class-agnostic detection and segmentation techniques. *Segmentation is akin to object detection, but instead of providing a bounding box around the object, it outlines all the individual pixels that form the object.* These methods have the capability to identify common objects within an image without necessarily categorizing each object, rendering them well-suited for zero-shot detection scenarios. The fundamental principle behind zero-shot detection can be summarized as follows. Initially, a class-agnostic segmentation or detection model is employed, which, upon receiving an input image, identifies all salient objects present. A classification filter is then applied to retain only those objects belonging to the desired categories. This classification filter leverages the concept of distance embeddings, as discussed in *(Section 2.2.2)*. Specifically, a text-image model, primarily CLIP **[4]**, transforms textual descriptions of each category into vectors. Simultaneously, objects detected in the image are transformed into vectors within the same vector space as the textual descriptions. Subsequently, the distances between the object's vector and all classes vectors are computed, and the object is assigned to the class with the shortest distance.

One notable example of such a model is SAM **[31]**, a recent and potent segmentation model capable of locating objects in an image without class identification. However, due to its substantial size, it can be challenging and slower to use on standard devices. To address this limitation, FastSAM **[32]** endeavors to create a model similar to SAM but more compact and faster. It adopts the architecture of Yolov8-seg **[15]**, a recent segmentation model, in addition to a prompting model. Despite its reduced size, FastSAM manages to achieve performance levels close to SAM, while offering improved efficiency. Several other class-agnostic detectors are available, including Detic **[33]**, Grounding-Dino **[34]**, and OWL-VIT **[35]**. These models share common traits, such as pretraining on large datasets and the utilization of transformers and attention mechanisms, which empower them to achieve state-of-the-art performance.

In contrast, zero-shot density counting models have not garnered as much attention as their detection and segmentation counterparts. However, a recent breakthrough paper titled CLIP-Count **[36]** addresses this issue *(Figure 12)*. CLIP-Count leverages pre-trained CLIP encoders with fine-tunable prompts and utilizes a patch-text contrastive loss to align visual and textual features. The model incorporates a hierarchical text-patch interaction module to generate multi-modal feature maps at different resolutions, which are then decoded using a CNN decoder. This approach enables effective fusion of visual and textual information for density counting tasks, and demonstrates promising results in the realm of zero-shot density counting.

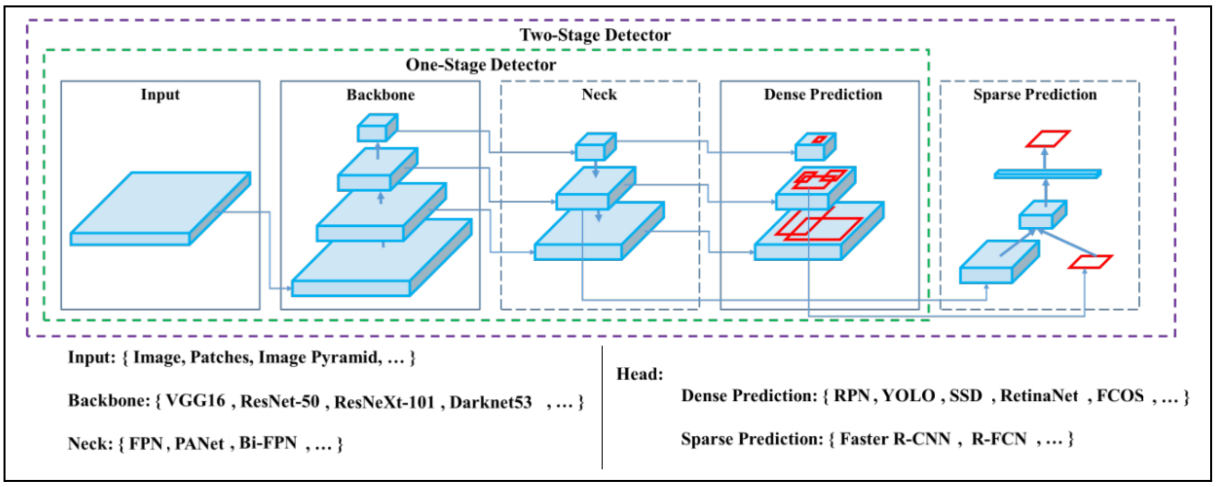

Two-Stage Detector
One-Stage Detector
Input
Backbone
Neck
Dense Prediction
Sparse Prediction
Input: { Image, Patches, Image Pyramid, … }
Backbone: { VGG16 , ResNet-50 , ResNeXt-101 , Darknet53 , … }
Neck: { FPN , PANet , Bi-FPN , … }
Head:
Dense Prediction: { RPN , YOLO , SSD , RetinaNet , FCOS , … }
Sparse Prediction: { Faster R-CNN , R-FCN , … }

Figure 8: General architecture of object detection models: the initial step is to process the input image through a backbone network, which extracts valuable feature maps. More sophisticated feature extraction techniques include multiscale feature extraction and refinement using a neck block, resulting in improved features at various resolutions. These extracted features can be employed in the head for prediction, directly through one-stage models, or by ROI pooling in two-stage detectors.[7]

[7] From YOLOv4 paper

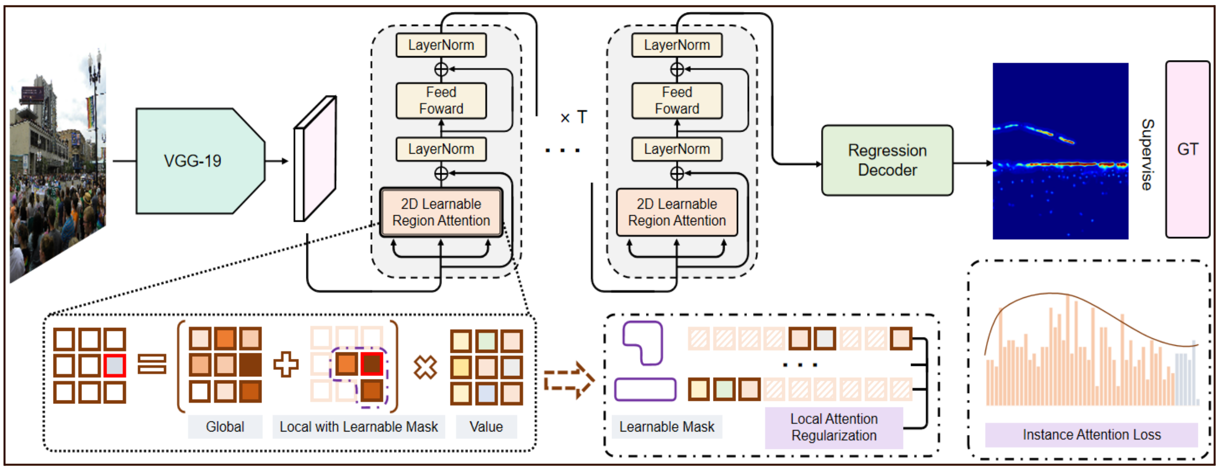


Figure 9: Multifaceted Attention Network structure: it initiates the process by extracting valuable features from the input image through the VGG19 backbone. These features are then passed through a sequence of transformers that incorporate attention mechanisms. Then, a decoder network is employed to generate the final density map, which is utilized for counting.[8]

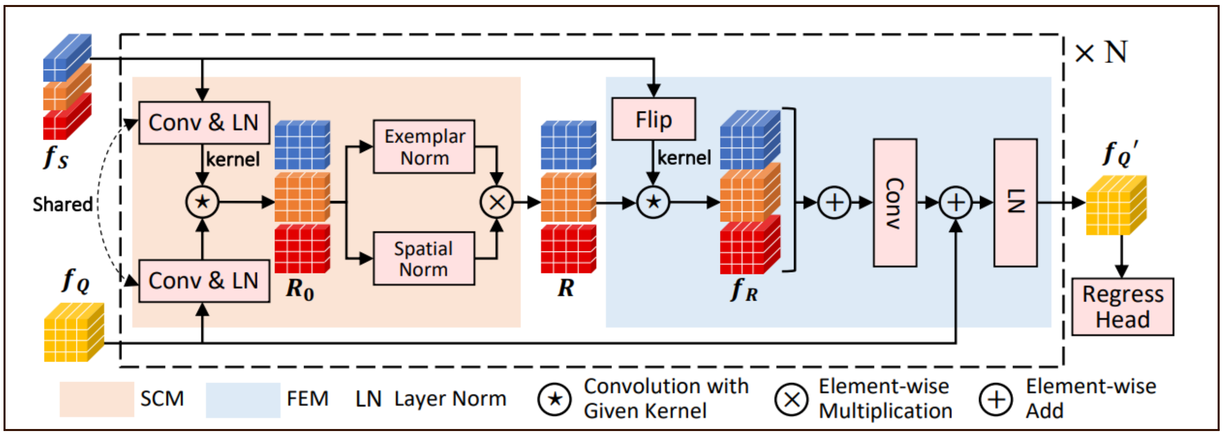


Figure 10: SAFECount model design: it commences with feature extraction from both the query image and the support images (shots). The support features are then utilized as a convolutional kernel to identify regions of interest within the query features. Following a sequence of refinement steps, a regression head is employed to produce the final density map used for counting.[9]

[8] From MAC paper
[9] From SAFECount paper

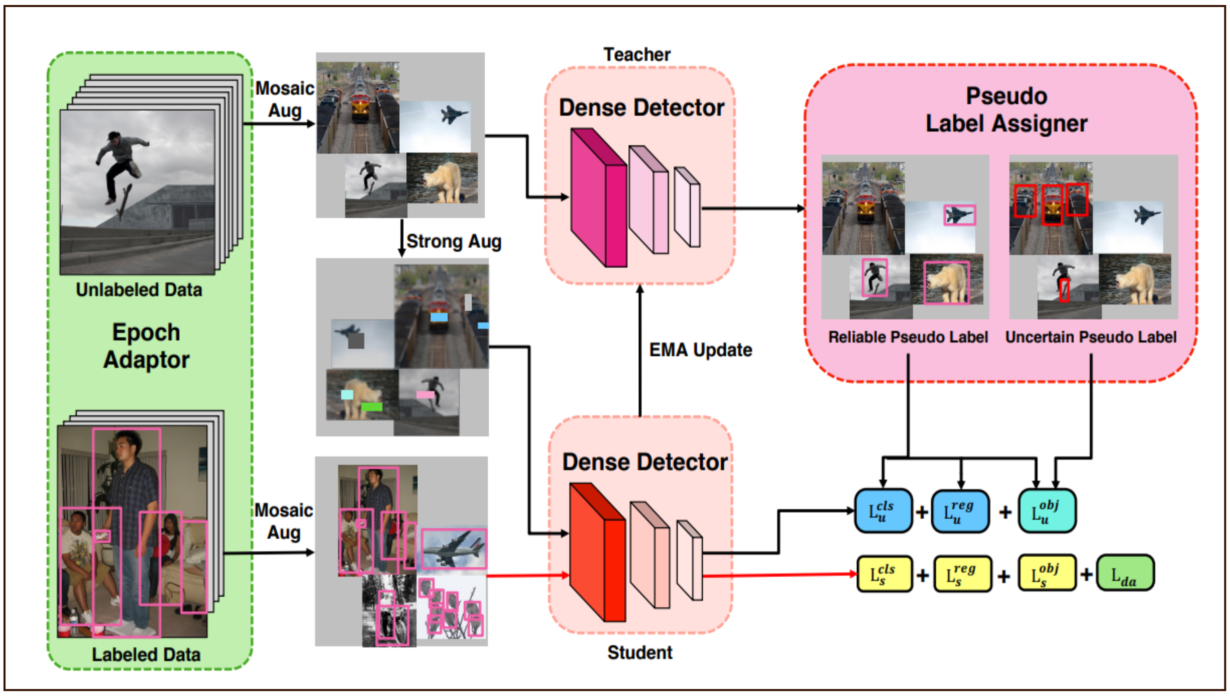


Figure 11: Efficient-Teacher training pipeline: it employs two pipelines. The first is dedicated to labeled data and incorporates mosaic augmentation before being fed into the student model for supervised training. While the second pipeline involves an additional strong augmentation to data before being used to generate two predictions: one by the student and another by the teacher model. Afterward, a loss function is computed by comparing these two predictions, constituting the unsupervised loss.[10]

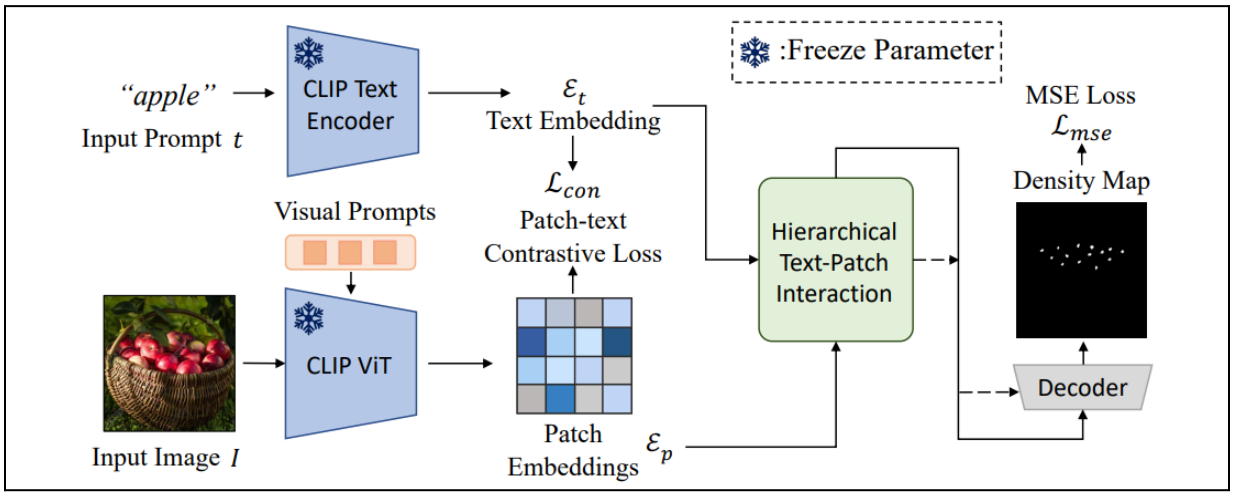


Figure 12: The training pipeline utilized for CLIP-Count: it involves the processing of an input image and an input text prompt through the frozen encoders of CLIP to extract embeddings. These are then aligned using a contrastive loss, and are passed through a multi-resolution text-patch interaction block before being forwarded to the decoder, which generates the density map.[11]

[10] From Efficient-Teacher paper
[11] From CLIP-Count paper

4. Methods

After presenting relevant prior research related to this thesis, we will now introduce the selected methods that will be implemented and evaluated in this study. Our choices were influenced by several key considerations:

Time Efficiency: Training a model involves comprehending the research paper and the provided code by the authors, followed by the actual training process, which can span up to a full day. Furthermore, obtaining final results often necessitates a significant amount of time for hyperparameter tuning and configuring the training setup. Given these time constraints, we had to be selective and prioritize models that could be implemented within a reasonable time frame.

Library Support: Some methods lacked published code, or the available code proved challenging to run due to dependency issues when trying to recreate the author's training environment. Consequently, we favored models that were supported by libraries offering a consistent training syntax. This approach allowed us to train multiple models using a familiar syntax, rather than investing additional time in learning new complex setups.

Recent Performance: In the vast landscape of methods within each area of study, we opted for recent ones that have demonstrated strong performance on publicly available datasets. This selection criterion ensured that we were working with models at the forefront of research.

Diversity: To provide a comprehensive evaluation, we made an effort to choose a diverse set of models rather than restricting ourselves to a single type. Consequently, we incorporated both density and detection models, as well as one-stage and two-stage detectors, ensuring a well-rounded exploration of the subject matter.

4.1. Selected Supervised Methods

In accordance with the categorization discussed in the preceding section, our choice of supervised counting models falls within three distinct categories:

Two-stage Methods: We have excluded R-CNN and Fast R-CNN, ultimately selecting Faster R-CNN for its superior speed and precision relative to its predecessors. Faster R-CNN deploys two backbone architectures, ResNet50 and ResNet101, both of which we have employed in our experiments.

One-stage Models: Within this category, our selection encompasses the YOLO series, specifically YOLOv5, YOLOv6, YOLOv7, and YOLOv8, chosen for their enhanced performance compared to earlier YOLO models and the convenience of consistent training syntax across them. Each version comprises multiple submodels, denoted as small (s), medium (m), large (l), extra-large (x), and a deeper ‘6’ variant (e.g., s6, m6, l6) for YOLOv5. Our chosen models include YOLOv5 (s6, m6, l6), YOLOv6 (s, m, l), YOLOv7 (l, x), and YOLOv8 (s, m, l). The sizing nomenclature corresponds to the model's complexity.
Additionally, we have selected the RetinaNet model as another one-stage detector, featuring both available backbones: ResNet50 and ResNet101, and we have utilized both for our experiments.

Density-Based Models: In this category, we have chosen the state-of-the-art P2PNet and MAC models.

This comprehensive selection comprises a total of 17 models, encompassing 15 dedicated to object detection and 2 specialized in density estimation.

4.2. Selected Semi-Supervised Methods

The semi-supervised training setup used in our study, which will be explained in the next section (*Section 5*), is equivalent to training 15 distinct models. Given the complexity and time required for such an undertaking, we made a decision to opt for a single method.
In light of this, we chose the Efficient-Teacher method for several reasons. Firstly, Efficient-Teacher implementation is based on the YOLOv5 implementation, which streamlined the replication of the training environment compared to other models. Additionally, it has proven its effectiveness among state-of-the-art techniques, making it a suitable choice for our research.
It is important to note that the Efficient-Teacher method is specifically based on YOLOv5 (l), which distinguishes it from YOLOv5 (l6) that has increased depth and more parameters. The provided code by the authors supports only the (l) version.

4.3. Selected Few-Shot Methods

In the context of few-shot counting methods, we have opted to stick with density-based approaches due to their inherent advantage of not requiring fine-tuning when introducing new classes. This characteristic makes them a better fit for retail store management. To be more precise, our chosen methods are FamNet and SAFECount.

4.4. Selected Zero-Shot Methods

In the case of zero-shot counting approaches, our decision was to explore all the previously mentioned methods. These methods are advantageous because they don't necessitate any training or fine-tuning and can be applied directly. Consequently, we utilized all six methods: SAM + CLIP, FastSAM + CLIP, Grounding Dino, OWL-VIT, Detic + CLIP, and CLIP-Count.

5. Experimental design

In this section, our objective is to provide a clear explanation of our approach to benchmarking all the selected methods. We will also outline the structure of our datasets, the training setup, and the evaluation metrics employed.

5.1. Research plan

Following the introduction of the topic, problem statement, background information, discussion of related works, and the selection of the methods for our study, we will now delve into the experimental setup used for training and evaluating these approaches.
The primary goal is to create a test dataset and assess all models from various frameworks using it. While some models, such as zero-shot and few-shot models, do not require training, others in the supervised and semi-supervised frameworks necessitate training. Consequently, we will create a training dataset to facilitate this purpose. The evaluation will encompass measuring the speed, size, and accuracy of each presented model, enabling us to compare their performance across different scenarios. To achieve this, we will proceed as follows:

1. **Data Collection and Annotation:** We will gather and annotate the necessary data to construct both our training and test datasets.
2. **Training Supervised and Semi-Supervised Models:** For models within the supervised and semi-supervised frameworks, we will undertake training to prepare them for evaluation.
3. **Performance Evaluation of All Models:** Using our test dataset, we will evaluate the performance of all models, focusing on aspects such as speed, model size, and accuracy.
4. **Discussion of Results:** Finally, we will discuss the outcomes of our evaluations and draw conclusions regarding the performance of the various models under different circumstances.

It is worth noting that this study has been conducted using Python 3, a widely adopted and popular programming language, particularly in the field of Deep Learning.

5.2. Dataset

5.2.1. Background

Object detection and density estimation models rely on training data composed of pairs such as (x, y) or (image, label) or (image, annotation). In this context, the label or annotation typically represents the information about objects within an image. For detection models, this annotation consists of (x, y), which specifies the top-left corner of the object's bounding box, along with (w, h), representing the width and height of the object's bounding box, and the category id indicating the object's category.
Density estimation models, however, require a different label format, commonly expressed as the (x, y) coordinates of the object's center. To create a dataset that accommodates both annotation requirements, we have adopted the format used for detection, specifically the YOLO format: category_id, center_x, center_y, w, h. This format allows for flexibility and enables us to write and execute a Python script to convert between different annotation formats.

The process of assigning a label to an image, achieved by manually locating and defining an object within the image, is known as labeling or annotating. Various tools are available for this purpose, which display the image and enable users to select objects using the mouse cursor. In our case, we have opted to utilize an online platform called Roboflow[12]. It offers assisting tools that alleviate the tedium of this process, although it remains time-consuming.

5.2.2. Collection and Annotation

The dataset we are working with is organized as shown in *Table 1*, comprising three distinct components:

- **Training Set**: This set consists of 10,000 images encompassing 41 categories of fruits and vegetables. In total, it contains 51,200 objects.
- **Test Set**: It comprises 606 images, covering the same 41 categories of fruits and vegetables. Within this set, there are a total of 12,500 objects.
- **Unlabeled Data**: We have also collected a dataset of 3,200 images. This dataset includes most of the 41 categories of fruits and vegetables, along with images of other categories. Importantly, this data is unlabeled and serves the purpose of semi-supervised training.

The selection of these 41 categories was made through a specific process. Initially, we gathered images from various sources on the internet, primarily from platforms like Roboflow Universe and Kaggle[13], where public datasets are available. Notably, not all of the collected images came with annotations. To address this, we compiled multiple datasets containing fruits and vegetables and merged them. Additionally, we augmented classes with fewer images by adding more data and then proceeded to annotate the images. After this meticulous process, we arrived at a final set of 41 categories for our dataset.

Dataset	**Images**	**Resolution**	**Annotations**	**Categories**
Train	10000	640x640	51200	41
Test	606	640x640	12500	41
Unlabeled	3200	640x640	None	Includes most 41 categories

Table 1: Dataset structure

5.2.3. Statistics

Figure 13 provides an overview of the categories along with the corresponding number of annotations for each class. It is evident that the classes are well-balanced, with the majority having over 1000 examples. On average, there are approximately 1252 objects per category, which provides a sufficiently large dataset for effective training while minimizing the risk of overfitting. Furthermore, the dataset exhibits diversity, as reflected in the names of the categories, which encompass a wide range of common fruits and vegetables.

[12] *https://roboflow.com*
[13] *https://www.kaggle.com*

Turning to *Figure 14*, we find statistics related to the test set. In this set, all categories are represented by more than 230 examples, with an average of 305 examples per category. Additionally, there is an average of 20.6 objects per image. It is important to note that we took care to ensure diversity within each class, covering both dense and sparse situations. This diversity makes the test set well-suited for counting tasks.

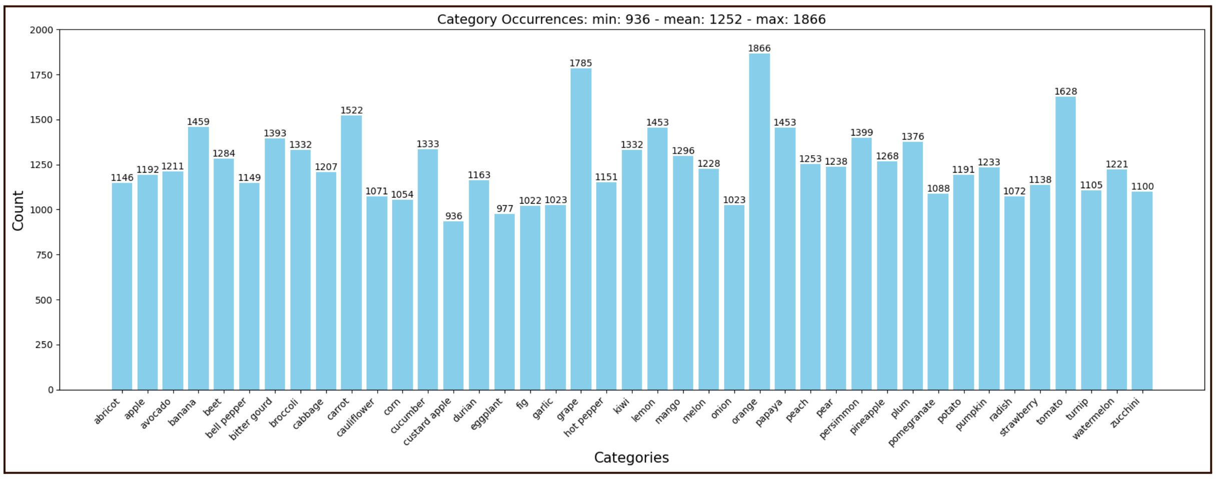


Figure 13: Statistics about training set

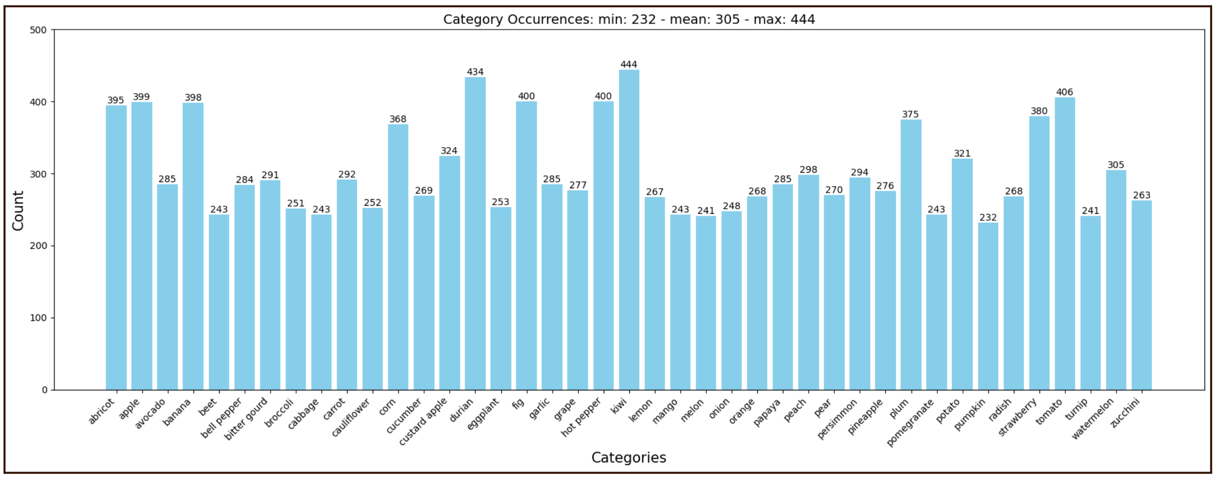


Figure 14: Statistics about test set

5.2.4. Dataset Examples

In *Figure 15*, we have displayed images from almost all categories in alphabetical order, showcasing the diversity and image quality across various categories. This arrangement allows for a comprehensive view of the dataset, highlighting the range of fruits and vegetables represented and the clarity of the images.

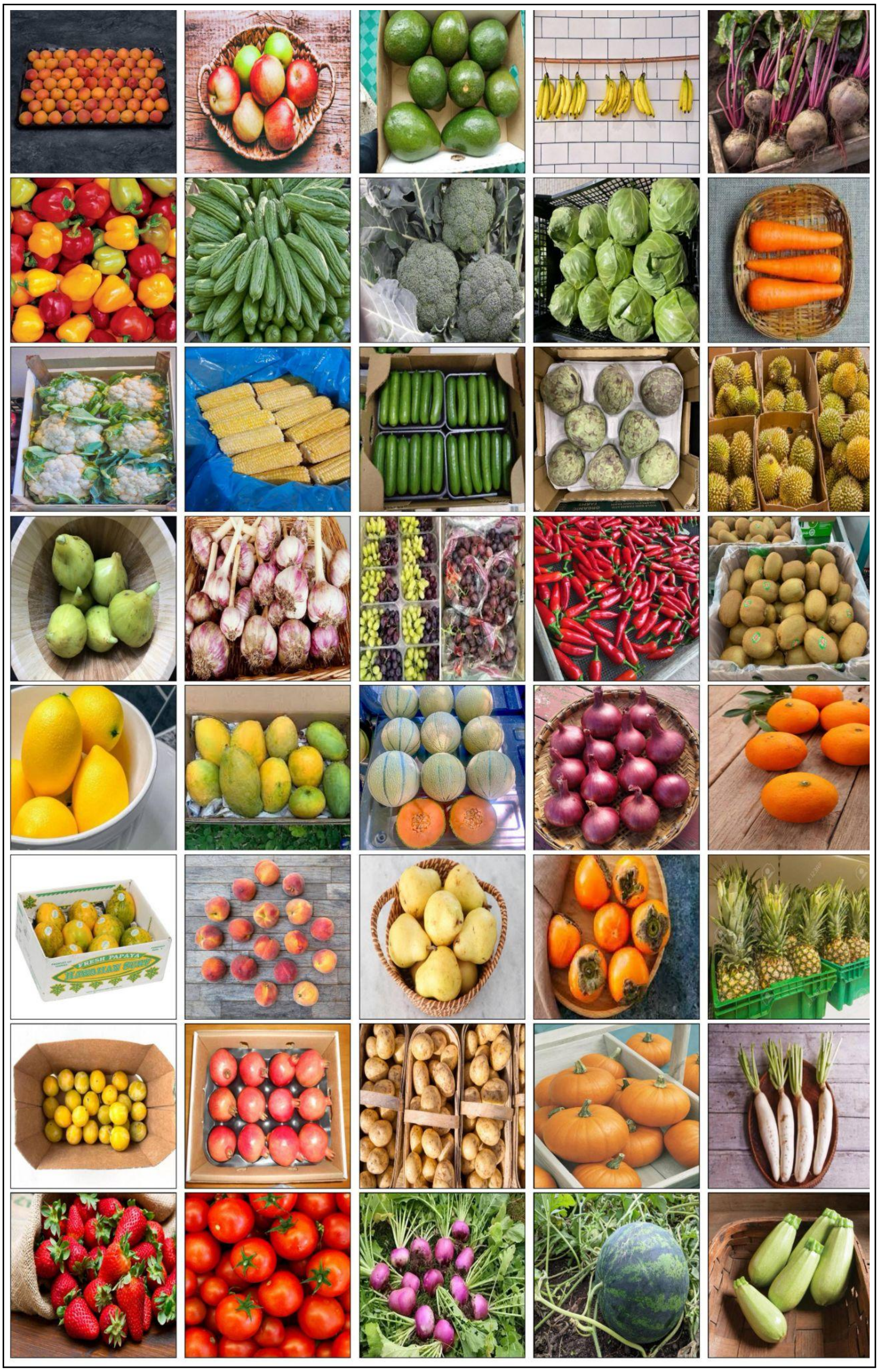


Figure 15: Illustrative Samples from the dataset

5.3. Training Setup

5.3.1. Supervised

For the supervised models, including both detection-based and density-based approaches, the training process will involve using the entire training set for training purposes. Additionally, the test set will be used as the validation set to evaluate model performance during training. Here is an outline of the training process:

1. **Hyperparameter Tuning**: Before the final training run, preliminary runs will be conducted to fine-tune hyperparameters. This includes optimizing parameters such as learning rate, number of epochs, weight decay, and other relevant settings. The goal is to find the best configuration that maximizes model performance.
2. **Final Training Run**: Once the optimal hyperparameters have been determined, the final training run will take place. During this run, the model will be trained on the entire training dataset while monitoring its performance on the test (validation) set.
3. **Weight Saving**: After the final training run, the model's weights will be saved. These saved weights will be used for later evaluation, particularly for counting tasks.

5.3.2. Semi-Supervised

The semi-supervised experimental setup differs from the supervised setup, as our primary objective is not merely to train a model and evaluate its performance. Instead, our aim is to investigate the impact of data scarcity by assessing the efficacy of the Efficient-Teacher method in augmenting the performance of a supervised model through the incorporation of unlabeled data. In parallel, we seek to examine the merits of utilizing pretrained model weights versus starting with random weight initializations. To this end, we intend to partition our dataset into five splits, as delineated in *Table 2*, each comprising a combination of labeled and unlabeled data. The images for these experiments will be drawn from the training set consisting of 10,000 labeled images and the 3,200 unlabeled images.
For instance, in the case of the 10% split, our dataset will be partitioned into 1,000 labeled images and 12,200 unlabeled images. Specifically, the 1,000 labeled images will be selected from the original training set, while the remaining 9,000 images from the training set will be included as unlabeled data. Additionally, the 3,200 unlabeled images will be added, resulting in a total of 12,200.
We have ensured a balanced distribution of labeled examples within each split to maintain a representative dataset, as depicted in *Figure 17*.
For each of these five splits, we plan to train three models:

- YOLOv5 (l) trained exclusively with random weight initialization on labeled data.
- YOLOv5 (l) initialized with COCO weights and trained solely on labeled data.
- YOLOv5 (l) initialized with COCO weights and trained on a combination of labeled and unlabeled data.

It is important to note that the COCO weights refer to the model parameters obtained from pretraining on the COCO[14] dataset provided by the authors.

Our hypothesis is that both pretraining and the inclusion of unlabeled data will contribute to an improvement in the performance compared to the model trained solely on labeled data with random weight initialization.

[14] COCO is a renowned publicly available dataset. *https://cocodataset.org*

Split	Labeled Images	Unlabeled Images
10 %	1000	12200
30 %	3000	10200
50 %	5000	8200
70 %	7000	6200
100 %	10000	3200

Table 2: Composition of each experimental split

5.3.3. Codes & Libraries

The training of supervised and semi-supervised models, along with the utilization of few-shot and zero-shot models, was accomplished by harnessing a variety of resources. These resources encompass author-provided codes, libraries, and documentations.

For YOLOv5 and YOLOv8 models, we used the Ultralytics[15] library, which provided us with the necessary tools for effective training. In the case of RetinaNet and Faster R-CNN, we employed the Detectron2[16] library. For all other models, we used the respective GitHub repositories maintained by the authors. Furthermore, our research benefited from the helpful resources and tutorials offered by Roboflow blog and Hugging Face[17].

5.4. Evaluation Metrics

5.4.1. Errors

For each image in the test set, we denote $\hat{y}$ as the true number of elements and y as the predicted number of elements. Let n represent the total number of images, and $avg(n)$ represent the average number of objects per image. As depicted in *Figure 16*, the Mean Absolute Error (MAE) quantifies the average absolute difference across the entire test set between the actual and predicted counts. This metric is commonly employed in counting evaluations due to its intuitive nature and widespread use.

However, relying solely on MAE may not provide a comprehensive assessment of model performance. For instance, an error of 3 when estimating a box of 100 apples amounts to only 3%, whereas the same error on a box containing 10 watermelons equates to 30%. To address this, we introduce the concept of Relative Error (Rel) by normalizing MAE with respect to the average number of objects. By doing so, we can gauge model performance based on a percentage scale, allowing us to set relevant thresholds for different use cases, such as 10% or 20%.

Consider a scenario with two test images: the first containing 100 apples and the second featuring 10 watermelons, resulting in an average object count of 55. Suppose our model achieves a MAE of 3. In

[15] *https://ultralytics.com*
[16] *https://github.com/facebookresearch/detectron2*
[17] *https://huggingface.co*

this case, the Relative Error (Rel) is calculated as 3/55*100 = 5.45%, indicating a low error while the model made 30% error on the second image. Therefore, Rel alone may not adequately convey model performance when dealing with sparse object distributions within images. To address this limitation, we introduce the Mean Relative Error (MRE), which takes into account the individual error percentages and averages them. In the example provided, it results in an MRE of 16.5%, offering a more balanced assessment.
Nonetheless, relying solely on MRE may introduce issues when dealing with outliers. An excellent model performance on most images can be overshadowed by a single instance where the model makes a substantial error, leading to a high MRE and potentially misleading conclusions. To strike a balance and harness the advantages of both Rel and MRE, we propose a combined error metric, as outlined in *Figure 16*. This composite metric, calculated as 8.2% in our example, synthesizes the strengths of both Rel and MRE, by considering the higher error in the second image, while also being cautious not to overly emphasize it, recognizing that it may potentially be an outlier. The formula employed is a commonly used two-variable function that aligns with the goal of minimizing both metrics (or maximizing, as the case may be). This approach provides a more comprehensive evaluation of model performance, accommodating varying scenarios and accounting for potential outliers. For the rest of this thesis, the terms 'combined error' and 'error' will be used interchangeably.

5.4.2. Size & Speed

To gauge the speed of our models, we will perform the counting of objects within all the images in our test set. The time taken to perform these predictions will then be divided by the total number of images, yielding the average prediction time per image. It is important to highlight that this timing data was collected using a mid-range laptop equipped with an Nvidia T600 mobile GPU. Consequently, it is essential to acknowledge that results may vary depending on the hardware configuration, and they should be interpreted in light of this variability. Nevertheless, this information will enable us to compare the relative speed of different models effectively. Additionally, we will provide insights into the size of each model employed in our study. This is especially valuable since some models, due to their substantial size, may pose practical challenges when deployed in real-world retail store scenarios. Conversely, lighter models are more conducive to practical implementation and will be favored for their efficiency.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

$$Rel = \frac{MAE}{avg(n)} \times 100$$

$$MRE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{\hat{y}_i}\right| \times 100$$

$$CombinedError = 2 \times \frac{Rel \times MRE}{Rel + MRE}$$

Figure 16: Mathematical definitions of used metrics: the mean absolute error quantifies the discrepancy between actual values and predictions, and it becomes more insightful when converted into the relative error, which when combined with the mean relative error offers improved robustness against outliers and provides a more accurate representation of the model's performance.

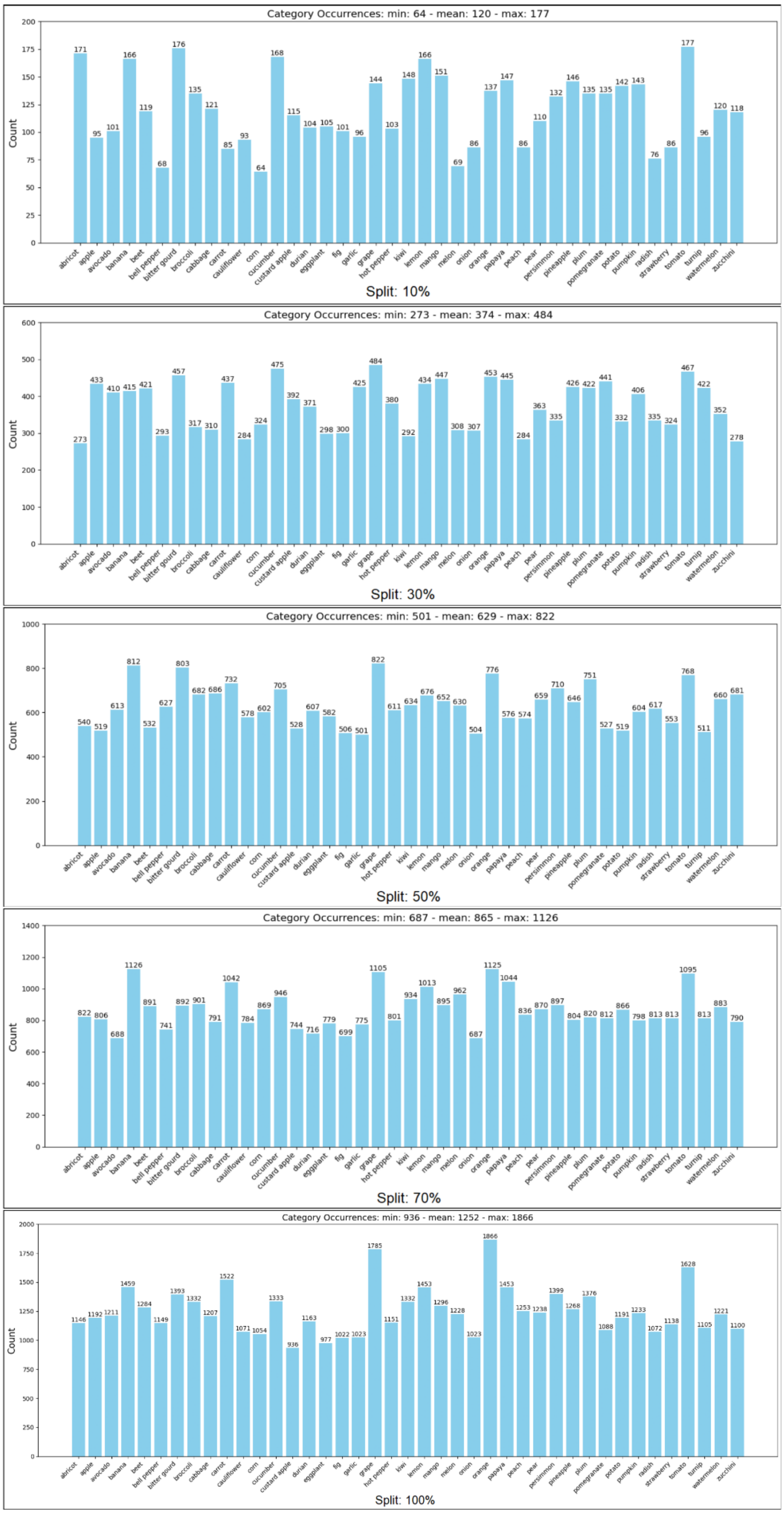


Figure 17: Labels statistics of each split

6. Results and Analysis

In this section, we present the evaluation results based on the experimental design previously outlined for each framework, and discuss the results. We also discuss the counting challenges we encountered and the strategies we used to surmount them.

6.1. Supervised

6.1.1. Counting Difficulties Arising from Obscuration and Visual Similarity

In our dataset comprising 41 distinct categories of fruits and vegetables, we frequently encounter situations where deciding whether an object in an image is a fruit or a vegetable is manageable, but categorizing it accurately becomes challenging due to obscured or partial views. Even human observers may struggle with such identifications. This challenge is illustrated in *Figure 18*, where our model adeptly identified pears and oranges but inaccurately categorized them as mangoes and abricots, respectively. This misclassification is natural given the difficulty of distinguishing these categories without a complete and unobstructed view of the entire piece, which is typically not available.

However, it is important to note that this classification discrepancy does not affect the overall object count within each image. If we count all objects in the image, we arrive at the same total count, which is the primary focus of our analysis. This approach aligns seamlessly with the automation framework illustrated in *Figure 2*, as we assume that each image contains only one known type of fruits and vegetables, and our objective is solely to determine the quantity of these objects. Consequently, by tallying all objects within the image, we effectively overcome the challenges associated with categorization ambiguity.

One might question whether, given our approach of counting all objects in the image regardless of their categories as long as they are detected as fruits and vegetables, it would be more efficient to treat all 41 categories as a single category and train the model accordingly, transforming the problem into a binary detection task. In fact, we did explore this option by training models on all 41 categories as a single category. However, our findings revealed that training on the 41 distinct categories and counting all of them during inference yielded superior results compared to the alternative method. As a result, we made the decision to stick with the initial approach.

It is worth noting that density models are not trained to differentiate among the 41 distinct categories, as they support only a single possible class. Instead, we amalgamate all classes into one to ensure the model's functionality.

6.1.2. Counting Results

Now, let's turn our attention to the outcomes yielded by supervised counting models.
For detection-based models, as shown in *Figure 19*, most models have managed to maintain an error rate below 15%. Except for RetinaNet, which exhibits a somewhat higher error rate, ranging from 17% to 18%. Faster R-CNN, on the other hand, has achieved a lower error rate of 13%, especially when coupled with the bigger Resnet101 backbone. In contrast, YOLO models, specifically YOLOv7 and YOLOv5, achieve better results with an error rate below

12%. In addition to their accuracy, YOLO models also offer advantages in terms of model size, ranging from 25 MB to 150 MB, and faster inference speeds, falling within the range of 14 ms to 115 ms per image. In comparison, RetinaNet and Faster R-CNN models have larger model sizes, spanning from 280 MB to 460 MB, and require more time for inference, ranging from 160 ms to 220 ms per image. More comprehensive results can be found in *Table 3*.

Regarding density-based models, as indicated in *Table 3*, MAC exhibited a low error rate of 13.88%, in contrast to the significantly higher error rate of 39.95% observed with P2P. Both models share similar attributes in terms of size, ranging from 250-300 MB, and inference times averaging between 173ms and 185ms. For MAC, we explored both counting methods: summation and contour detection. Interestingly, both methods produced comparable results. However, P2P being a point-based approach, it supports only the default summation way.
It is important to emphasize that the poor performance of P2P in this specific context does not necessarily reflect its overall effectiveness in other scenarios. This discrepancy may be attributed to the fact that objects detected in our dataset exhibit a wide range of shapes and characteristics, spanning across 41 distinct categories. In contrast, density models are typically designed to count objects of a single class.

In summary, it is evident that the majority of models exhibited lower error rates, with YOLO models consistently delivering the lowest errors. Moreover, YOLO models stand out for their speed and efficiency, making them particularly well-suited for retail store scenarios. Notably, YOLOv5 (m6) shines with its impressive inference time of 30 ms, compact size of 70 MB, and an error rate of 12%, making it a compelling choice that strikes a favorable balance between performance and deployment constraints.

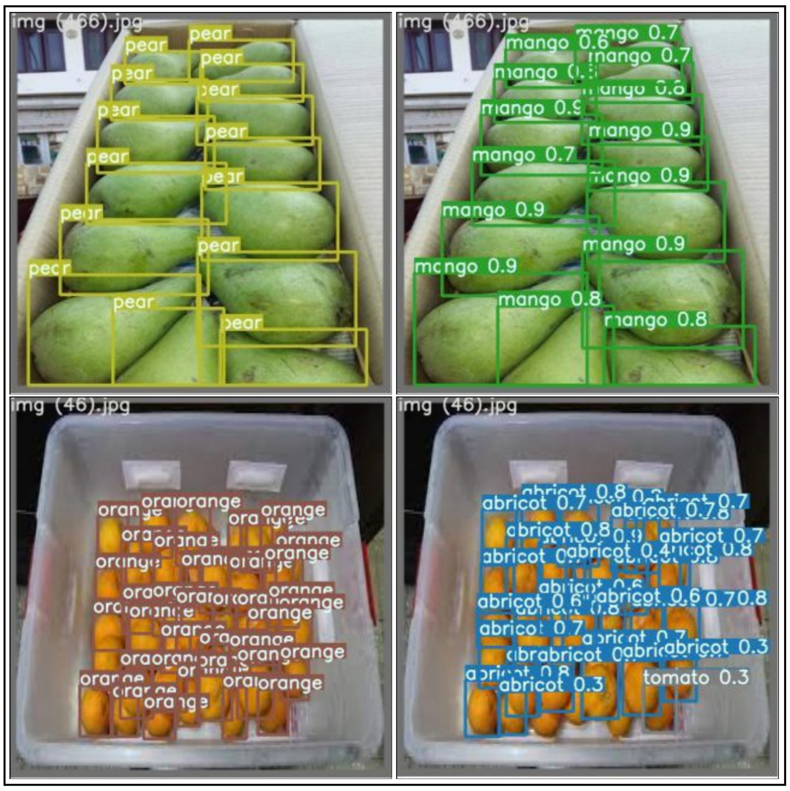
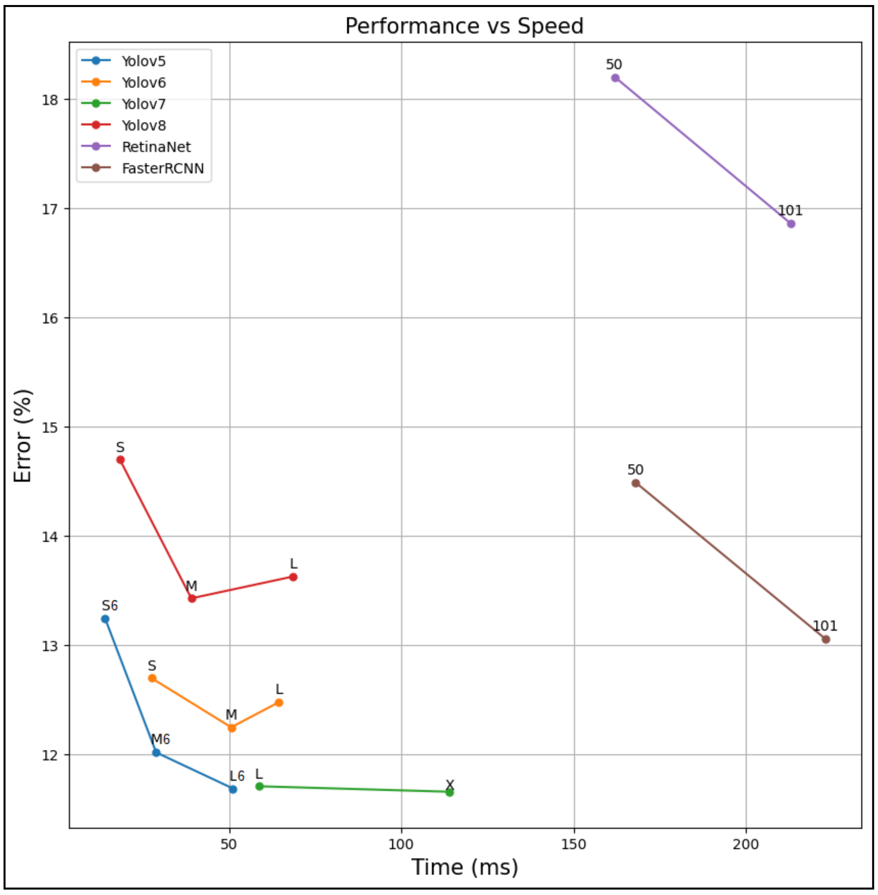


Figure 18: Comparison of actual labels (left) and YOLOv7l predictions (right): the model successfully identified the presence of objects of interest and their spatial locations within the image. But, it struggled to accurately determine their categories due to their similarity and partial obscuration.



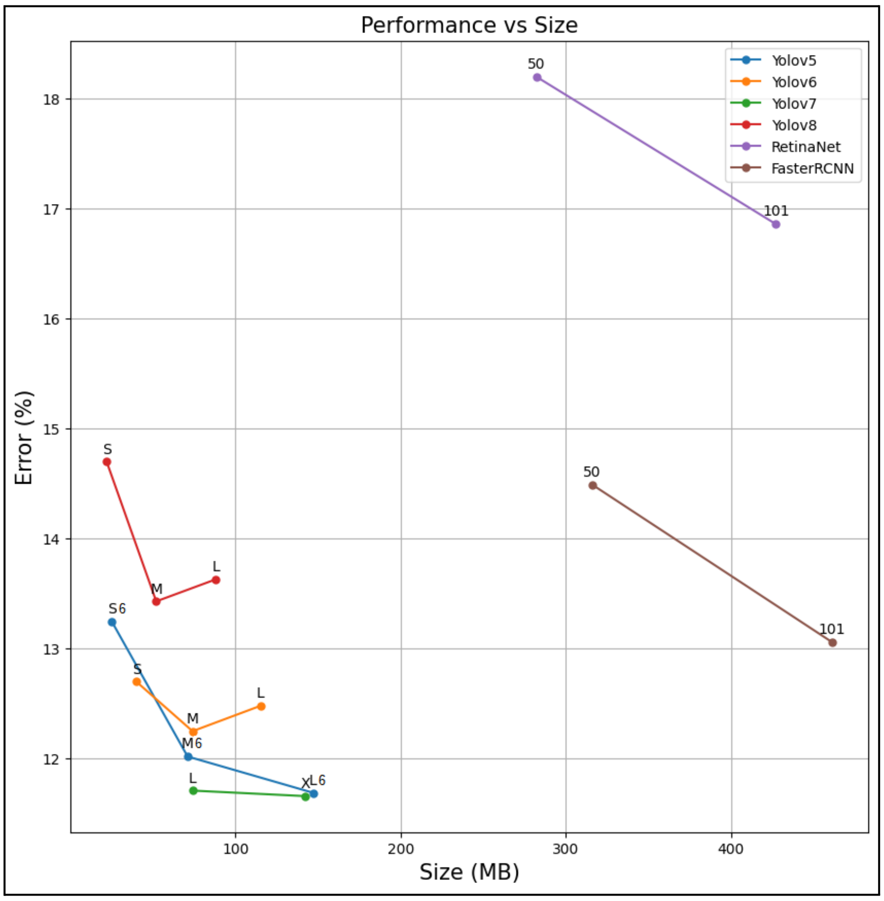


Figure 19: Performance, size, and inference speed for detection models

Rank	Model	Size	Time	MAE	Rel	MRE	Combined Error
9	Yolov5s6	25 MB	14.04 ms	3	14.65 %	11.8 %	13.25 %
4	Yolov5m6	71 MB	28.89 ms	2.78	13.46 %	10.54 %	12.02 %
2	Yolov5l6	147 MB	51.15 ms	2.71	13.12 %	10.21 %	11.69 %
7	Yolov6s	40 MB	27.55 ms	2.83	13.7 %	11.67 %	12.7 %
5	Yolov6m	74 MB	50.66 ms	2.8	13.54 %	10.93 %	12.25 %
6	Yolov6l	115 MB	64.52 ms	2.81	13.61 %	11.31 %	12.48 %
3	Yolov7l	74 MB	58.74 ms	2.65	12.86 %	10.52 %	11.71 %
1	Yolov7x	142 MB	114 ms	2.64	12.81 %	10.48 %	11.66 %
14	Yolov8s	22 MB	18.36 ms	3.28	15.91 %	13.46 %	14.7 %
10	Yolov8m	52 MB	39.10 ms	3.05	14.76 %	12.05 %	13.43 %
11	Yolov8l	88 MB	68.64 ms	3.01	14.59 %	12.65 %	13.63 %
16	Retina 50	282 MB	162 ms	3.74	18.12 %	18.28 %	18.2 %
15	Retina 101	427 MB	213 ms	3.84	18.59 %	15.42 %	16.86 %
13	Faster 50	316 MB	168 ms	2.98	14.42 %	14.57 %	14.49 %
8	Faster 101	461 MB	223 ms	2.97	14.38 %	11.96 %	13.06 %
12	MAC	300 MB	185 ms	2.97	14.39 %	13.4 %	13.88 %
17	P2P	250 MB	173 ms	7.06	34.18 %	48.06 %	39.95 %

Table 3: Benchmark results of supervised models

6.1.3. Analysis of Performance Limitations

From our previous findings, it is evident that the performance of all models under consideration was consistently constrained by an 11% error margin. This limitation can be attributed to various factors, including model architecture, training data volume, and hyperparameter settings. However, upon closer examination of prediction instances, we discerned that this issue predominantly stemmed from confusion between specific categories.

As exemplified in *Figure 20*, instances where multiple pieces of plum were arranged closely resembled a cluster of grapes. Consequently, the models, instead of detecting individual plum pieces, often identified the entire cluster as a single entity, resulting in significantly higher errors. To investigate this issue further, we devised a modified training and testing dataset that excluded grape and hot pepper categories. The rationale behind omitting hot pepper was its real-world distribution, which is typically sold in large quantities, making a per-piece count inaccurate. Therefore, any error in predicting hot pepper quantity could result in substantial error inflation. Additionally, we amalgamated several categories with similar shapes and appearances into unified categories, reducing the overall class count to 31, as indicated depicted in *Figure 21*.

Subsequently, we conducted a re-evaluation of existing models without retraining, specifically YOLOv5m6, YOLOv6m, YOLOv7l, and YOLOv8m, with the modified dataset excluding grape and hot pepper. Afterward, we trained new models using the updated dataset with 31 classes. The outcomes of these assessments are presented in *Table 4*.

Notably, the removal of grape and hot pepper categories yielded a discernible reduction in error across all models. For instance, YOLOv5 exhibited an initial error of 12.02%, which decreased to 10.2% upon exclusion of these categories. Furthermore, training a new model on the revised dataset directly led to a further error reduction to 8.73%, representing an overall reduction of approximately 3.5%. Similar improvements were observed in YOLOv6, with a 2.8% error reduction, YOLOv7 with a 2.81% reduction, and YOLOv8 with a 2.71% reduction.

In summary, the removal of categories that presented counting difficulties due to their intricate shapes, like grapes and hot peppers, along with the consolidation of visually similar classes, has improved the model performance.

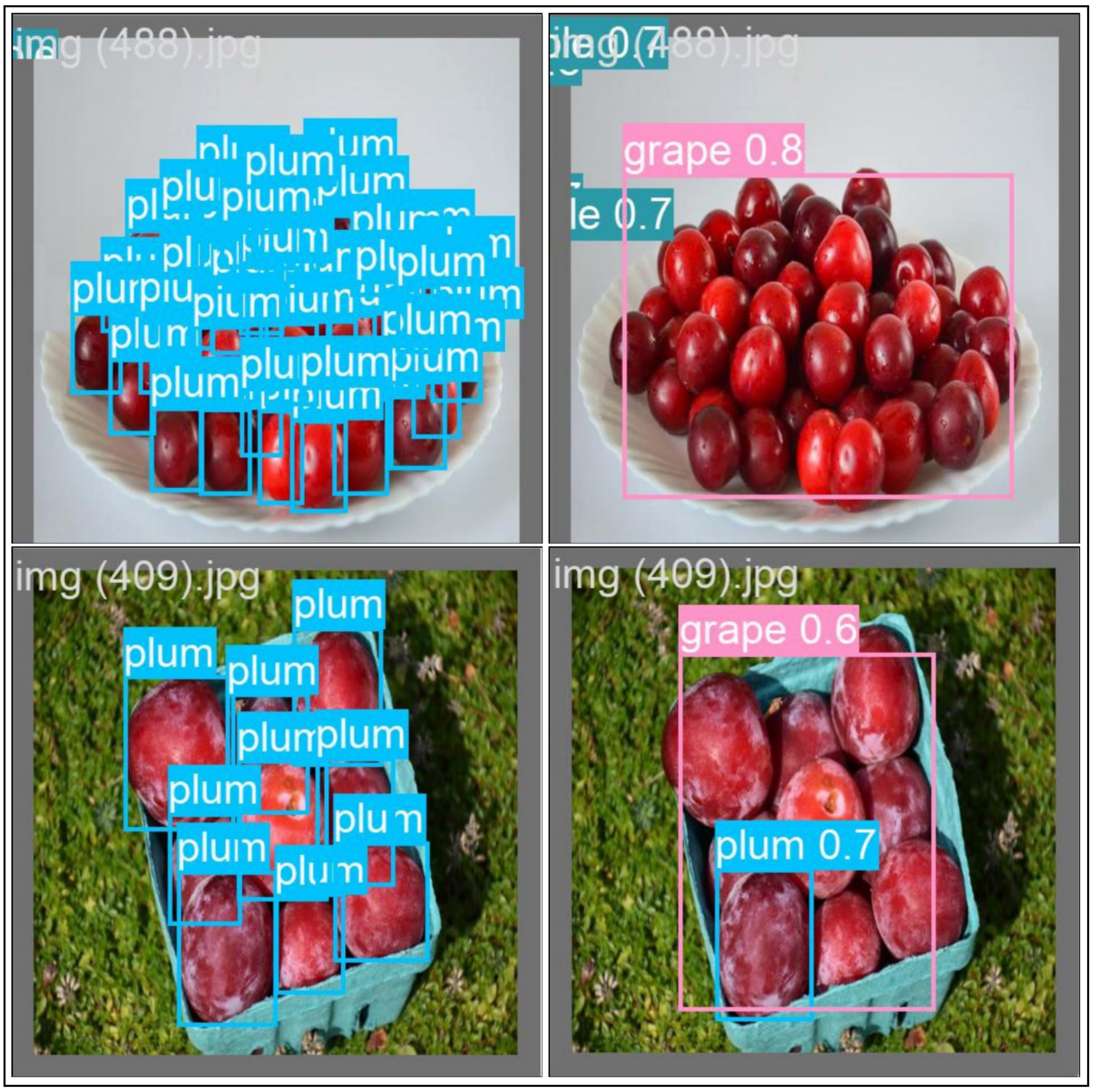
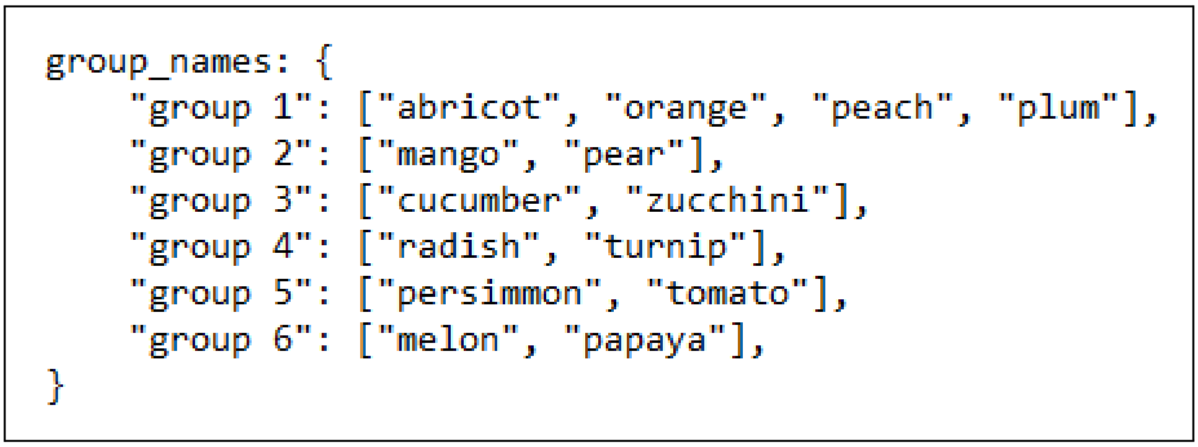


Figure 20: Visualization of performance limitations: mis-detecting plum pieces as grape branches due to their visual resemblance (Actual vs. Predicted)



```
group_names: {
    "group 1": ["abricot", "orange", "peach", "plum"],
    "group 2": ["mango", "pear"],
    "group 3": ["cucumber", "zucchini"],
    "group 4": ["radish", "turnip"],
    "group 5": ["persimmon", "tomato"],
    "group 6": ["melon", "papaya"],
}
```

Figure 21: Formation of novel categories through visual similarity grouping

Model	MAE	Rel	MRE	Combined Error
Yolov5m6	2.23	10.93%	9.57%	10.2 %
Yolov5m6-31	1.88	9.22%	8.30%	8.73 %
Yolov6m	2.28	11.22%	10.57%	10.88 %
Yolov6m-31	1.95	9.60%	9.30%	9.45 %
Yolov7l	2.22	10.93%	9.59%	10.22 %
Yolov7l-31	1.84	9.05%	8.75%	8.9 %
Yolov8m	2.53	12.43%	11.10%	11.73 %
Yolov8m-31	2.18	10.76%	10.69%	10.72 %

Table 4: Benchmark results with new settings (suffix '31' denotes model retained on new dataset)

6.2. Semi-Supervised

After conducting exhaustive training sessions on three different models across five distinct data splits, the results are graphically presented in *Figure 22*. Notably, the impact of pre-training on model performance was particularly prominent, especially within the 10% split, where labeled data were significantly limited. Without pre-training, the model exhibited an error rate of 24%, which was notably reduced to around 20% with pre-training. What is more intriguing is the subsequent application of semi-supervised learning, which further lowered the error rate to 14%, a substantial improvement. This trend continued as the dataset proportion increased, with the error decreasing from 18% to 13% in the case of 30% split, and from 14% to 12% in the case of 50% split. However, an anomaly emerged with the 70% split, as all three models exhibited higher error rates. To contextualize this anomaly, it is crucial to understand that achieving the favorable results seen in previous splits was not an instantaneous process. It involved numerous training iterations and the fine-tuning of hyperparameters, particularly those related to Efficient-Teacher settings, which consumed a significant amount of time. While we achieved impressive results for the previous splits, the 70% split did exhibit reduced error rates but not to the same extent as the other splits. This is also observed in the pretrained model in the 100% split, which displayed a slightly higher error rate compared to the model with random initialization. However, we believe that with more time and additional training resources, further exploration of hyperparameters could yield improved results, potentially aligning the anomalous outcomes with the overall trend, as depicted in *Figure 23*.

It is noteworthy to consider the diminishing returns observed in the gains provided by pre-training and semi-supervision when transitioning from 10% to 30% and 50% data splits. This can be attributed to the fact that in the 10% split, we introduced 9,000 unlabeled images, which is nine times the quantity of labeled data utilized. In contrast, for the 100% split, we added only 3,200 unlabeled images, which represents 32% of labeled images already in use. This phenomenon is indicative of the model's increasing difficulty in leveraging additional data to substantially boost its performance beyond a certain point. Additionally, it is worth highlighting that the semi-supervised model trained in the 100% split achieved the best performance across all frameworks examined with an error of 11%. More comprehensive results can be found in *Table 5*.

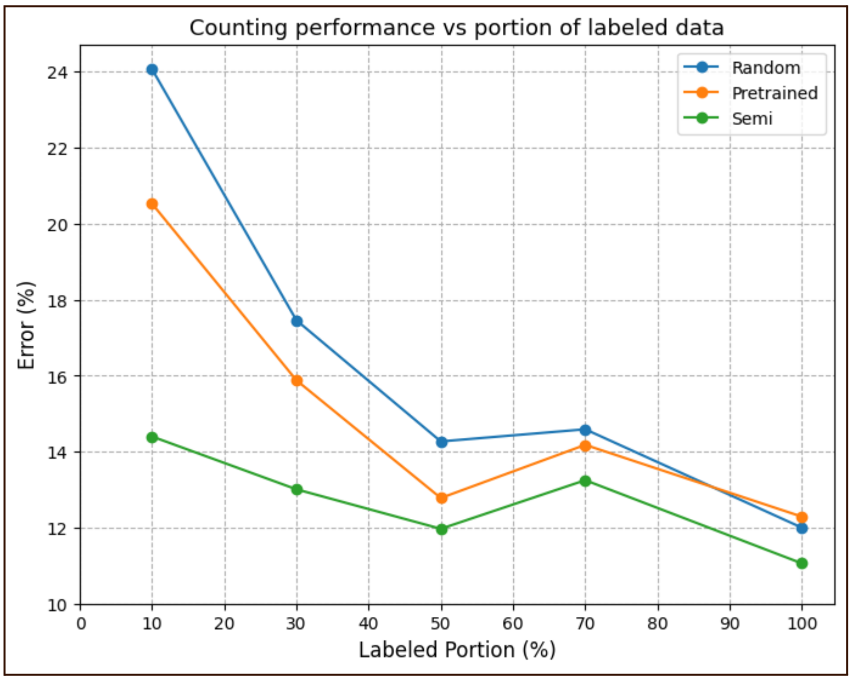


Figure 22: Semi-supervision results across different data split

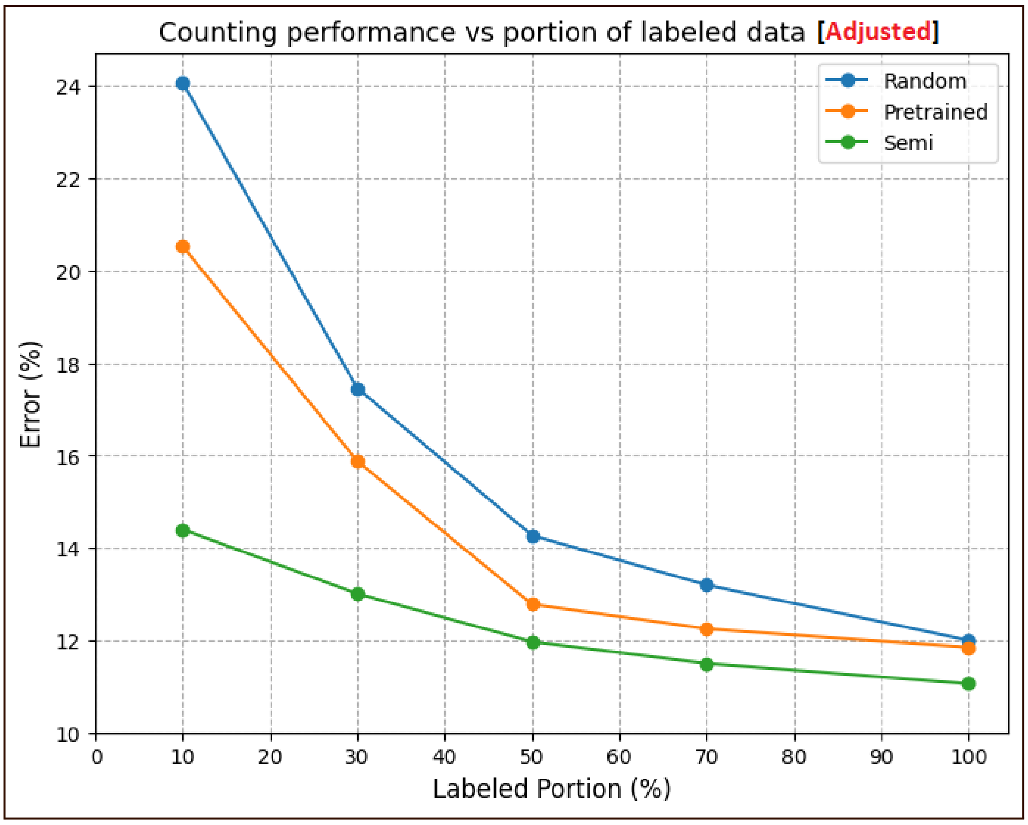


Figure 23: Semi-supervision results across different data split with **adjustment**

Split	Type	MAE	Rel	MRE	Combined Error
10 %	Random	5.05	24.44 %	23.72 %	24.07 %
	Pretrained	4.44	21.5 %	19.65 %	20.53 %
	Semi	3.24	15.7 %	13.3 %	14.4 %
30 %	Random	3.88	18.8 %	16.3 %	17.46 %
	Pretrained	3.51	17.01 %	14.89 %	15.88 %
	Semi	2.92	14.14 %	12.05 %	13.01 %
50 %	Random	3.11	15.08 %	13.54 %	14.27 %
	Pretrained	2.77	13.44 %	12.18 %	12.78 %
	Semi	2.67	12.92 %	11.15 %	11.97 %
70 %	Random	3.27	15.83 %	13.53 %	14.59 %
	Pretrained	3.18	15.43 %	13.12 %	14.18 %
	Semi	2.86	13.84 %	12.71 %	13.25 %
100 %	Random	2.8	13.56 %	10.76 %	12 %
	Pretrained	2.8	13.58 %	12.22 %	12.29 %
	Semi	2.52	12.21 %	10.11 %	11.06 %

Table 5: Detailed benchmark results of semi-supervised training

6.3. Few-Shot

6.3.1. Shots Preparation

The two models we employed in our study require both an image and the coordinates of rectangles (x1, y1, x2, y2) denoting specific objects or "shots" **within** the image. It is important to maintain an image size close to 512x512, although 640x640 can also be used. To accommodate our models' requirements, we adopted an image resolution of (512+64) for the height and 512 for the width. The additional height of 64 pixels allows us to stack eight images of size 64x64 within a single image, facilitating the positioning of these shots. Consequently, we can input this composite image to the models to generate density maps.

For the purpose of model evaluation, we have prepared templates for each category. These templates are images with dimensions of (512+64)x512, and they include the eight shots corresponding to the specific category, pre-positioned within the template. When we want to make predictions for a given image, we first resize it to 512x512, and then we place it within the template corresponding to its category. Subsequently, we feed this composite image into the models for prediction, after which we count the results excluding the 8 already present objects.

The process of selecting the eight representative shots for each category involved careful visual inspection of the images. We aimed to ensure that the selected shots were diverse and effectively represented the category in question.

6.3.2. Counting Results

As outlined in *Table 6*, we employed two counting strategies for each model: summation and contours counting. SAFECount exhibited an error rate of 54.5% when using the summation method, which was notably reduced to 47.23% when employing the contour-based counting method. On the other hand, FamNet demonstrated even higher errors, especially when applying the summation method, with an error rate of 105.8%. This error was considerably improved to 61.57% when utilizing the contour-based counting approach. It is important to note that both models have comparable sizes, ranging from 108MB to 125MB, with SAFECount being slower in terms of inference time, taking 588ms, while FamNet has a faster inference time of 207ms.

These few-shot models, as demonstrated, exhibit relatively high error rates, even with the best-performing method achieving an error rate of 47%. This error rate is substantial and indicates that these models fall short when compared to supervised or semi-supervised methods, which generally yield superior results in object counting tasks.

Model	**Size**	**Time**	**Method**	**MAE**	**Rel**	**MRE**	**Combined Error**
SAFECount	125 MB	588 ms	Sum	8.44	40.88 %	81.76 %	54.5 %
			Contours	8.4	40.67 %	56.33 %	47.23 %
FamNet	105 MB	207 ms	Sum	16.63	80.55 %	154.1 %	105.8 %
			Contours	10.86	52.61 %	74.22 %	61.57 %

Table 6: Benchmark results of few-shot models

6.4. Zero-Shot

After conducting a comprehensive evaluation of zero-shot models, both detection-based and density-based, we present the results in *Table 7* and *Table 8*. It is evident that all the models in this category exhibit error rates exceeding 30%. Among these models, FastSAM and SAM emerge as the top performers, with error rates of 32.94% and 34.52%, respectively. Notably, FastSAM stands out for its superior speed and lighter resource requirements compared to SAM. Following closely behind are CLIP-Count with the contours method, Detic, and Grounding Dino, which report combined error rates of 36.58%, 39.10%, and 43.10%, respectively. These models offer faster inference times, clocking in at less than 200ms, and are more resource-efficient, with sizes below 1GB. Conversely, the OWL-VIT model delivers the poorest performance in this evaluation, with a staggering error rate of 90.82%. It is noteworthy that this model generated numerous zero predictions, contributing to its exceptionally high error rate.

Upon considering the results, CLIP-Count with the contours method appears to be a favorable choice, as it offers an error rate close to that of FastSAM while maintaining higher speed and resource efficiency.

It is crucial to acknowledge that all the models in this category are considered large, which may pose constraints during deployment. Furthermore, while these zero-shot models outperform few-shot models (with the best few-shot model achieving an error rate of 47%), they still fall short of the performance achieved by supervised and semi-supervised models.

Model	**Size**	**Time**	**MAE**	**Rel**	**MRE**	**Combined Error**
FastSAM	0.75 GB	> 1 sec	5.6	28 %	40 %	32.94 %
SAM	3 GB	> 2 sec	5.78	28 %	45 %	34.52 %
CLIP-Count	0.75 GB	< 200 ms	5.67	27 %	54 %	36.58 %
Detic	1 GB		7.06	34 %	46 %	39.10 %
Grounding Dino	0.7 GB		9.5	46 %	42 %	43.10 %
OWL-VIT	0.6 GB	> 1 sec	19.7	95 %	87 %	90.82 %

Table 7: Benchmark results of zero-shot models

Model	**Method**	**MAE**	**Rel**	**MRE**	**Combined Error**
CLIP-Count	Sum	15.39	74 %	242 %	114 %
	Contours	5.67	27 %	54 %	36.58 %

Table 8: Detailed results of CLIP-Count method

7. Discussion

The primary objective of this thesis is to establish a comprehensive benchmark for various counting methodologies across four distinct data scenarios, facilitating a discerning evaluation of their respective strengths and limitations. As outlined in the results section, our analysis indicates that trained models have demonstrated superior performance, particularly with regard to YOLO models, which exhibited an error rate below 12%. Notably, YOLO models, with their compact and efficient nature, further improved their performance when specific problematic shape classes, such as grapes and hot peppers, were excluded from the training data, achieving an impressive error rate as low as 8.73%. Additionally, the utilization of unlabeled data, particularly in scenarios with limited annotated data (e.g., 10% and 30% semi-supervised training cases), substantially enhanced model performance. The utilization of Efficient-Teacher, a YOLO-based approach, optimized the results without resorting to less accurate [few-shot] or bigger [zero-shot] models.

Furthermore, our investigation revealed that few-shot models exhibited inferior performance compared to other categories, with the best-performing model having a relatively high error rate of 47%. On the other hand, zero-shot models demonstrated a lower error rate, approximately 30%, despite the expense of larger model size and slower inference times. These trade-offs may pose practical challenges, particularly in contexts like retail stores, where access to high-end computational resources may be limited.

In light of these findings, we propose a guide to aid users in selecting the most suitable framework and model based on their specific use case:

1. **Supervised**: When a sufficient amount of annotated data is available, we recommend opting for supervised learning, with a particular emphasis on the YOLO series. Notably, models like YOLOv5(m6) or YOLOv5(l6) offer a compelling combination of fast inference times and a small model size, making them ideal for deployment on resource-constrained devices commonly found in retail settings.
2. **Semi-Supervised**: In scenarios where only a portion of the data is labeled, the semi-supervised learning approach is advised, with a highlight on the Efficient-Teacher approach applied to YOLO models. This approach stands among the state-of-the-art and has demonstrated impressive results, especially when dealing with limited labeled data.
3. **Zero-Shot**: If data is scarce but robust computational resources are available, the preference should lean towards zero-shot learning, specifically the CLIP-Count method. This approach offers a balance between speed and acceptable error rates, making it a viable choice in such circumstances.
4. **Few-Shot**: As a last resort, when data is insufficient and computational resources are limited, few-shot learning can be considered, with the SAFECount approach being a suitable option. This approach can work on less powerful devices, ensuring compatibility with resource-constrained environments.

To summarize, this thesis has developed a benchmark designed to assist in the selection of the most suitable counting framework and model based on the specific demands of the use case, accommodating situations characterized by diverse levels of labeled data and computational resources.

8. Conclusion

Throughout this study, we have undertaken a comprehensive exploration of the subject matter and its underlying motivation. We initially provided an overview and motivation for our research, followed by an examination of the working principles and prior literature relevant to our work. Subsequently, we conducted a series of experiments and thoroughly discussed the obtained results.

Returning to our research questions:

1. **Experimental setup and evaluation criteria:** In *Section 5* (Experimental Design), we delineated our approach to establishing a benchmark, including the creation of datasets for both training and testing purposes, along with the presentation of evaluation criteria encompassing accuracy, speed, and model size.
2. **Performance of counting models:** *Section 6* (Results and Analysis) extensively detailed the performance of each method within the four data frameworks, featuring illustrative examples of encountered challenges and proposed solutions.
3. **Strengths and limitations of counting techniques:** Our exploration of the advantages and drawbacks of each method across various frameworks was addressed in both the Results and Analysis *(Section 6)* and Discussion *(Section 7)* sections.
4. **Guidance for framework and model selection:** We have indeed put forth a comprehensive guide in *Section 7* (Discussion), informed by our research findings, to assist users in selecting the most appropriate framework and model for their specific retail store needs.

During the course of this study, we confronted several noteworthy challenges. Firstly, comprehending recent, cutting-edge approaches, particularly those emerging in 2023, posed difficulties due to the necessity for manual experimentation to grasp fundamental concepts. Additionally, replicating the working environment of provided codes, especially when dealing with updates to Python libraries, presented significant challenges. Furthermore, data compilation for this thesis involved annotating numerous classes manually, which was a laborious and time-consuming task. Coupled with extensive model training times, which could span up to a full day for a single run, these challenges added complexity to our research process.

As a natural extension of this work, there exists room for enhancement. Acquiring larger datasets and exploring the utility of more substantial models can improve generalization and prediction accuracy. Continuous monitoring of new model releases to ascertain state-of-the-art performance is advisable. Furthermore, the implementation of ensemble learning or distillation techniques, along with rigorous hyperparameter optimization, could further enhance model performance, contingent upon the availability of resources and time, as discussed previously.

Another potential enhancement lies in addressing the challenge of differentiating between visually similar categories of fruits and vegetables. The development of novel architectures that enable the detection model to classify objects with higher accuracy could streamline the counting process. Specifically, we envision a unified approach where a single model can take an image of a fruit and vegetable shelf and provide both object counts per category and category verification. This approach can be extended to scenarios involving multiple shelves captured by high-resolution cameras, enabling precise counting across all shelves. Such advancements could lead to increased efficiency and cost reduction by minimizing the need for multiple cameras and the initial step of box identification, as depicted in *Figure 2*.

9. References

1. YOLOv5 does not have a paper. (2020). Retrieved from *https://ultralytics.com*

2. Lin, H., Ma, Z., Ji, R., Wang, Y., & Hong, X. (2022). Boosting crowd counting via multifaceted attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 19628-19637).

3. Lee, D. H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning*, ICML (Vol. 3, No. 2, p. 896).

4. Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., & Krueger, G. (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning* (pp. 8748-8763). PMLR.

5. Snell, J., Swersky, K., & Zemel, R. (2017). Prototypical networks for few-shot learning. *Advances in neural information processing systems, 30.*

6. Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 580-587).

7. Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (pp. 1440-1448).

8. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems, 28.*

9. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

10. Redmon, J., & Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 7263-7271).

11. Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767.*

12. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934.*

13. Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., & Li, Y. (2022). YOLOv6: A single-stage object detection framework for industrial applications. *arXiv preprint arXiv:2209.02976.*

14. Wang, C. Y., Bochkovskiy, A., & Liao, H. Y. M. (2023). YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 7464-7475).

15. YOLOv8 does not have a paper. (2023). Retrieved from *https://ultralytics.com*

16. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 1*4 (pp. 21-37). Springer International Publishing.

17. Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 2980-2988).

18. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European conference on computer vision* (pp. 213-229). Cham: Springer International Publishing.

19. Zhu, X., Su, W., Lu, L., Li, B., Wang, X., & Dai, J. (2020). Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*.

20. Lempitsky, V., & Zisserman, A. (2010). Learning to count objects in images. *Advances in neural information processing systems, 23.*

21. Zhang, Y., Zhou, D., Chen, S., Gao, S., & Ma, Y. (2016). Single-image crowd counting via multi-column convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 589-597).

22. Kannadi Valloli, V., & Mehta, K. (2019). W-Net: Reinforced U-Net for Density Map Estimation. *arXiv e-prints*, arXiv-1903.

23. Song, Q., Wang, C., Jiang, Z., Wang, Y., Tai, Y., Wang, C., Li, J., Huang, F., & Wu, Y. (2021). Rethinking counting and localization in crowds: A purely point-based framework. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 3365-3374).

24. Wang, X., Yang, X., Zhang, S., Li, Y., Feng, L., Fang, S., Lyu, C., Chen, K., & Zhang, W. (2023). Consistent-Teacher: Towards Reducing Inconsistent Pseudo-Targets in Semi-Supervised Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 3240-3249).

25. Xu, B., Chen, M., Guan, W., & Hu, L. (2023). Efficient Teacher: Semi-Supervised Object Detection for YOLOv5. *arXiv preprint arXiv:2302.07577*.

26. Lin, W., & Chan, A. B. (2023). Optimal Transport Minimization: Crowd Localization on Density Maps for Semi-Supervised Counting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 21663-21673).

27. Qiao, L., Zhao, Y., Li, Z., Qiu, X., Wu, J., & Zhang, C. (2021). Defrcn: Decoupled faster r-cnn for few-shot object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 8681-8690).

28. Zhang, G., Luo, Z., Cui, K., & Lu, S. (2021). Meta-DETR: Image-level few-shot object detection with inter-class correlation exploitation. *arXiv preprint arXiv:2103.11731*.

29. Chu, P., & Ling, H. (2019). Famnet: Joint learning of feature, affinity and multi-dimensional assignment for online multiple object tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 6172-6181).

30. You, Z., Yang, K., Luo, W., Lu, X., Cui, L., & Le, X. (2023). Few-shot object counting with similarity-aware feature enhancement. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 6315-6324).

31. Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., Xiao, T., Whitehead, S., Berg, A. C., Lo, W. Y., & Dollár, P. (2023). Segment anything. *arXiv preprint arXiv:2304.02643.*

32. Zhao, X., Ding, W., An, Y., Du, Y., Yu, T., Li, M., Tang, M., & Wang, J. (2023). Fast Segment Anything. *arXiv preprint arXiv:2306.12156.*

33. Zhou, X., Girdhar, R., Joulin, A., Krähenbühl, P., & Misra, I. (2022). Detecting twenty-thousand classes using image-level supervision. In *European Conference on Computer Vision* (pp. 350-368). Cham: Springer Nature Switzerland.

34. Liu, S., Zeng, Z., Ren, T., Li, F., Zhang, H., Yang, J., Li, C., Yang, J., Su, H., Zhu, J., & Zhang, L. (2023). Grounding DINO: Marrying DINO with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499.*

35. Minderer, M., Gritsenko, A., Stone, A., Neumann, M., Weissenborn, D., Dosovitskiy, A., Mahendran, A., Arnab, A., Dehghani, M., Shen, Z., & Wang, X. (2022). Simple open-vocabulary object detection. In *European Conference on Computer Vision* (pp. 728-755). Cham: Springer Nature Switzerland.

36. Jiang, R., Liu, L., & Chen, C. (2023). CLIP-Count: Towards Text-Guided Zero-Shot Object Counting. *arXiv preprint arXiv:2305.07304.*