SiMagna: Your guide to Italian cuisine using Object Detection with YOLOv11 model

CAROLINA TORBA[1], PIETRO PICCOLO[2], AND EGNALD CELA[3, *]

[1] *StudentID 2150103*
[2] *StudentID 2107512*
[3] *StudentID 2108230*
[*] *Submission: cela.2108230@studenti.uniroma1.it*

Compiled July 3, 2025

This work is submitted as a final project on the subject of AI Lab: Computer Vision and NLP by professor Daniele Pannone, on the course of Applied Computer Science and Artificial Intelligence (ACSAI). We trained a YOLOv11 model on our self-composed dataset made of 10 datasets over sources on Google Datasets, Kaggle and Roboflow. This model is able to recognize over 15 essential ingredients used in Italian cuisine and after such recognition a recipe matcher algorithm will suggest recipes to cook.

https://github.com/ppietropiccolo/siMagna

1. INTRODUCTION

SiMagna is a trained YOLO (You Only Look Once) computer vision model that aims to aid solving a simple, yet daily but crucial task such as deciding on your meals based on what you already have at home. We provided an interface that makes the whole process user-friendly and accessible. The idea is simple, take a picture of your fridge and our model will not only detect what's on your fridge but also recommend top three recipes you can prepare at home based on what you already have. These recipes are fully Italian, as a motivation to attract you deeper into Italian cuisine. This project is a nice blend of Computer Vision (specifically object detection) and a simple but efficient recommendation system linked to an external database for Italian recipes.

A. Motivation

Our project tends to address two real-world issues such as *waste reduction* and *produce management*, but it also offers an ease for an everyday struggle such as efficient meal preparation. Furthermore, this project can be seen as a soft launch into Italian culinary culture, easing its reachability and accessibility.

B. On the topic

The topic of object detection is not new to the Computer Vision community, but its application to household ingredient detection has been tackled on several papers. [1] YOLO models are the current state of the art in terms of object detection.

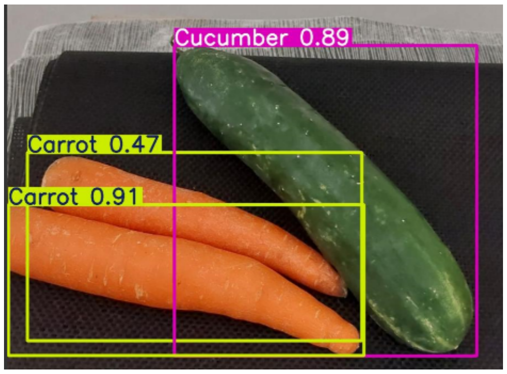


Fig. 1. Example of model prediction

2. OUR PIPELINE

A. General overview of our model

We trained a YOLOv11 model on our own composed dataset made of 7,000 images. This dataset is made of images from over 10 other such datasets found mainly on Roboflow. The goal was to feed enough data to the model to ensure performance. This model is able to recognize household ingredients at very fast inference time (0.004s). This model is then used as a backbone to recognize household ingredients.

```
SIMAGNA directory structure

 data
 italian gastronomic recipes dataset
 runs
 main.py
```

```
README.md
recipe_data_cleaning.ipynb
recipe_matcher.py
reorder_data.py
training.py
```

- *data* contains the dataset with all the ingredients prepared into the format YOLO model wants data to be fed
- *italian gastronomic recipes dataset* is the dataset listing all ingredients and recipes, with additional information such as preparation time, difficulty level and cost, which we display to the user for a better user experience
- *main* is the main file that calls everything together and initializes the interface in a public URL
- *reorder data* contains the scripts responsible of merging the 10 datasets into a single one, handling label renaming and compatibility with YOLO folder structure
- *recipe matcher* is the file responsible for getting the ingredients predicted by the model from the input image, then fetching recipes from the dataset done with such ingredients and returning it into a compatible gradio-like format
- *training* is the file responsible for handling the training of the model, importing the pre-trained weights and setting up hyperparameters

B. Dataset

B.1. Ingredients dataset

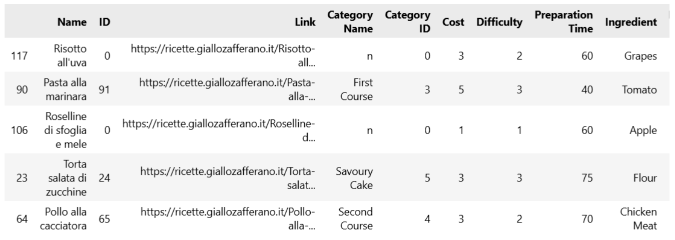
This is the main dataset used to train the model. It is a composition of over 10 different datasets found on Roboflow, which were then merged into the following structure to train the model.

```
dataset/
 train/
    images/
       img1.jpg
       img2.jpg
    labels/
        img1.txt
        img2.txt
 valid/
     images/
        img3.jpg
        img4.jpg
     labels/
         img3.txt
         img4.txt
```

This dataset contains images with bounding boxes for several ingredients, spanning fruits and vegetables such as apples, bananas, tomatoes, onions, cucumbers, eggs, etc. We also trained on some items being inside plastic bags, to achieve better results for partially covered objects.

B.2. Italian recipes dataset

This is the dataset used to fetch the recipes given a list of ingredients. It will fetch such recipes based on the matching it does between ingredients and the overall ingredients in a recipe.
Modification: We modified the dataset slightly to avoid some naming compatibility issues and we enriched it with a few new recipes based on the ingredients we trained on.



	Name	ID	Link	Category Name	Category ID	Cost	Difficulty	Preparation Time	Ingredient
117	Risotto all'uva	0	https://ricette.giallozafferano.it/Risotto-all...	n	0	3	2	60	Grapes
90	Pasta alla marinara	91	https://ricette.giallozafferano.it/Pasta-alla-...	First Course	3	5	3	40	Tomato
106	Roselline di sfoglia e mele	0	https://ricette.giallozafferano.it/Roselline-d...	n	0	1	1	60	Apple
23	Torta salata di zucchine	24	https://ricette.giallozafferano.it/Torta-salat...	Savoury Cake	5	3	3	75	Flour
64	Pollo alla cacciatora	65	https://ricette.giallozafferano.it/Pollo-alla-...	Second Course	4	3	2	70	Chicken Meat

Fig. 2. Sample from the recipes dataset

C. Training the model

We trained the model on our local machines, specifically on a NVIDIA RTX 3060 graphics card with 12GB of VRAM.
Data Augmentation: YOLO11 uses several data augmentation techniques to help the model generalize better. These include color changes (such as brightness and saturation), random movement and resizing of images, and flipping images sideways. It also combines multiple images using mosaic augmentation and applies random erasing to hide small parts of objects.

Hyperparameter	Value
Number of Epochs	150
Batch Size	16
Image Size	640

Table 1. Key Training Hyperparameters

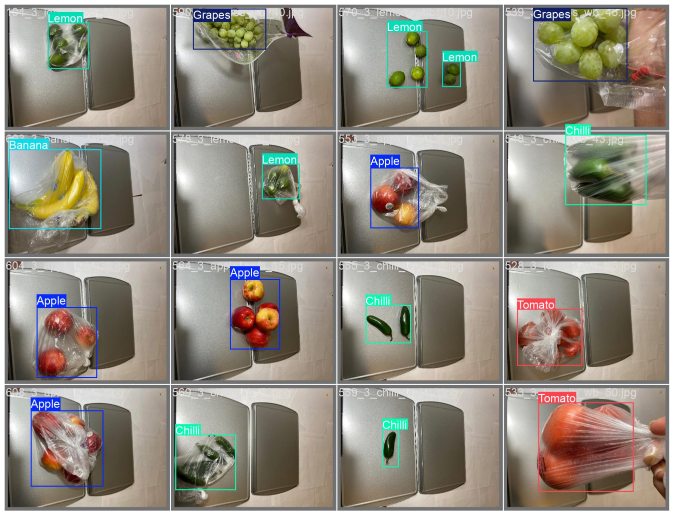
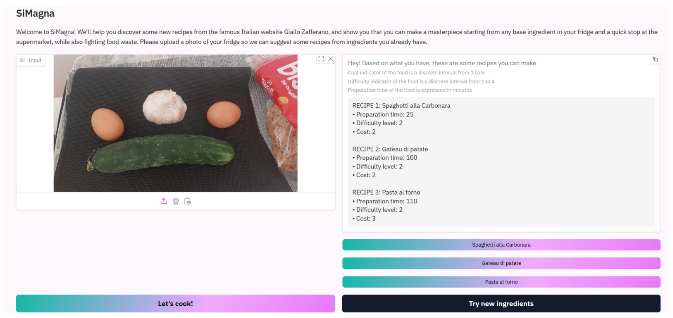


Fig. 3. Model succesfully recognizing some ingredients on validation set

D. Creating the interface

To build the user interface for our project, we used Python's library Gradio. We designed a clean and user-friendly interface that allows users to upload an image of their ingredients and receive recipe suggestions, accessible from any device (computer, mobile phone, or tablet) and easily shareable with others. We also customized a theme to enhance the interface's aesthetics, and Gradio's Blocks allowed us to organize the layout conveniently. At the top of the interface, we display the project title and a brief description of our goals. The middle section contains the input component on the left and the output components on the right. We used a placeholder image for the input to make the interface visually appealing upon launch. Since the image was loaded with OpenCV, we had to convert its color space for proper display. The output section remains hidden until recipe suggestions are generated. It includes a text box, allowing users to copy and share the recipe information, and three buttons that become visible once results are available, each linking to a full recipe. At the bottom of the interface are the main interaction buttons. The first, labeled "Let's cook", processes the uploaded image to recognize ingredients and find three recipes. The second button, "Try new ingredients", becomes active after a recipe search and resets the interface by clearing inputs and outputs, restoring the initial layout with the placeholder image and hidden output components.


SiMagna

Welcome to SiMagna! We'll help you discover some new recipes from the famous Italian website Giallo Zafferano, and show you that you can make a masterpiece starting from any base ingredient in your fridge and a quick stop at the supermarket, while also fighting food waste. Please upload a photo of your fridge so we can suggest some recipes from ingredients you already have.

Hey! Based on what you have, these are some recipes you can make
Cost indicator of the food is a discrete interval from 1 to 5
Difficulty indicator of the food is a discrete interval from 1 to 4
Preparation time of the food is expressed in minutes

RECIPE 1: Spaghetti alla Carbonara
• Preparation time: 25
• Difficulty level: 2
• Cost: 2

RECIPE 2: Gateau di patate
• Preparation time: 100
• Difficulty level: 2
• Cost: 2

RECIPE 3: Pasta al forno
• Preparation time: 110
• Difficulty level: 2
• Cost: 3

Fig. 4. Interface working in action

E. Putting it all together

At this point, we have a trained model (*weights loaded at best.pt file*) which is able to recognize ingredients. The *recipe matcher* file loads the trained model, then relies on designed functions that can output a list of ingredients from an input image. This ingredient list is fed into another function that can fetch recipes from the dataset together with other necessary data for each recipe such as preparation time, cost, difficulty, which will be outputed to the user.
Model input: An image
Model output: 3 recipes to cook with ingredients found on the image, together with the preparation time, difficulty level, cost and links to detailed cooking sites and visuals.

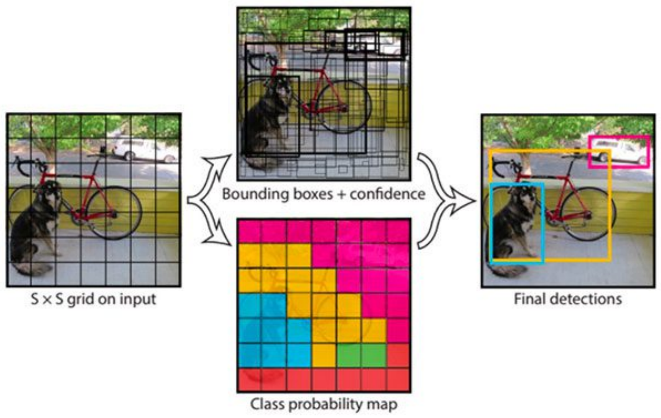


Fig. 5. How YOLO works in a nutshell

3. YOLO11 MODEL ARCHITECTURE

YOLOv11 is the latest in the YOLO series developed by Ultralytics, introducing several architectural enhancements aimed at improving both speed and accuracy in real-time object detection tasks.

A. Architecture Overview

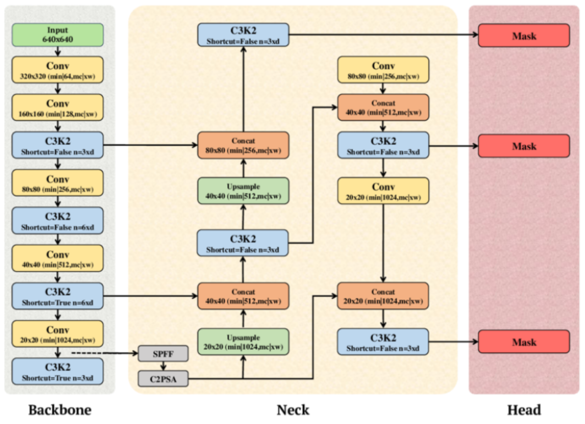

Input 640x640
Conv 320x320 (min|64,mc|xw)
Conv 160x160 (min|128,mc|xw)
C3K2 Shortcut=False n=3xd
Conv 80x80 (min|256,mc|xw)
C3K2 Shortcut=False n=6xd
Conv 40x40 (min|512,mc|xw)
C3K2 Shortcut=True n=6xd
Conv 20x20 (min|1024,mc|xw)
C3K2 Shortcut=True n=3xd
SPFF
C2PSA
C3K2 Shortcut=False n=3xd
Concat 80x80 (min|256,mc|xw)
Upsample 40x40 (min|512,mc|xw)
C3K2 Shortcut=False n=3xd
Concat 40x40 (min|512,mc|xw)
Upsample 20x20 (min|1024,mc|xw)
Conv 80x80 (min|256,mc|xw)
Concat 40x40 (min|512,mc|xw)
C3K2 Shortcut=False n=3xd
Conv 20x20 (min|1024,mc|xw)
Concat 20x20 (min|1024,mc|xw)
C3K2 Shortcut=False n=3xd
Mask
Mask
Mask
Backbone
Neck
Head

Fig. 6. Yolo11 architecture

The architecture of YOLOv11 [2] is structured into three primary components:

- **Backbone**: Responsible for feature extraction from input images.
- **Neck**: Aggregates and refines features to enhance representation.
- **Head**: Performs the final detection tasks, including classification and bounding box regression.

Key innovations in YOLOv11 include:

- **C3k2 Block**: A convolutional block with kernel size 2, enhancing feature extraction capabilities.
- **SPPF (Spatial Pyramid Pooling - Fast)**: Improves multi-scale feature extraction, aiding in detecting objects at various scales.
- **C2PSA (Convolutional block with Parallel Spatial Attention)**: Enhances spatial feature representation by focusing on important regions.

B. Mathematical Representation

Let $x \in \mathbb{R}^{H \times W \times 3}$ denote the input image with height H, width W, and 3 color channels.

B.1. Backbone Operation

The backbone applies a series of convolutional layers to extract hierarchical features:

$$F_1 = \sigma(W_1 * x + b_1)$$

$$F_i = \sigma(W_i * F_{i-1} + b_i), \quad i = 2, \ldots, n$$

where $*$ denotes convolution, W_i and b_i are weights and biases of the i-th convolutional layer, and σ is a nonlinear activation function (e.g., SiLU or ReLU). The final backbone feature map is

$$F = F_n \in \mathbb{R}^{H' \times W' \times C}$$

with reduced spatial dimensions H', W' and feature channels C.

B.2. Neck Operation

The neck refines the features extracted by the backbone using modules like SPPF and C2PSA. The SPPF block helps the model capture information at different scales, making it better at detecting both large and small objects. The C2PSA block adds attention, helping the model focus on the most important regions in the image for improved detection accuracy.

B.3. Head Operation

The head predicts bounding boxes, objectness scores, and class probabilities at each spatial location (i, j):

$$\hat{y}_{i,j} = \sigma(W_h * F_{\text{att}} + b_h)$$

where

$$\hat{y}_{i,j} = (\hat{x}, \hat{y}, \hat{w}, \hat{h}, \hat{o}, \hat{c}_1, \ldots, \hat{c}_K)_{i,j}$$

consists of:

- $\hat{x}, \hat{y}$: bounding box center coordinates (normalized relative to cell),
- $\hat{w}, \hat{h}$: bounding box width and height,
- $\hat{o}$: objectness score (probability an object exists),
- $\hat{c}_k$: class probabilities for K classes.

Bounding box coordinates are decoded as:

$$x = \frac{i + \sigma(\hat{x})}{W'}, \quad y = \frac{j + \sigma(\hat{y})}{H'}$$

$$w = p_w e^{\hat{w}}, \quad h = p_h e^{\hat{h}}$$

where (p_w, p_h) are prior anchor dimensions.

C. Loss Function

YOLOv11 uses a composite loss function that combines multiple terms to supervise the detection of objects effectively:

$$\mathcal{L} = \lambda_{\text{loc}}\mathcal{L}_{\text{loc}} + \lambda_{\text{obj}}\mathcal{L}_{\text{obj}} + \lambda_{\text{cls}}\mathcal{L}_{\text{cls}}$$

where

- $\mathcal{L}_{\text{loc}}$ is the localization loss, measuring bounding box regression error,
- $\mathcal{L}_{\text{obj}}$ is the objectness loss, measuring confidence of object presence,
- $\mathcal{L}_{\text{cls}}$ is the classification loss, measuring accuracy of predicted class probabilities,
- $\lambda_{\text{loc}}, \lambda_{\text{obj}}, \lambda_{\text{cls}}$ are hyperparameters controlling the relative weights of each loss component.

Localization Loss The localization loss typically uses the Complete Intersection over Union (CIoU) loss between predicted box $\hat{b}$ and ground truth box b:

$$\mathcal{L}_{\text{loc}} = 1 - \text{CIoU}(\hat{b}, b)$$

where CIoU accounts for overlap area, center point distance, and aspect ratio consistency:

$$\text{CIoU} = \text{IoU} - \frac{\rho^2(\hat{b}, b)}{c^2} - \alpha v$$

- IoU is the Intersection over Union between predicted and ground truth boxes, - ρ is the Euclidean distance between box centers, - c is the diagonal length of the smallest enclosing box covering both $\hat{b}$ and b, - v measures aspect ratio consistency, - α balances the aspect ratio term.

Objectness Loss The objectness loss is computed as binary cross-entropy between predicted objectness score $\hat{o}$ and ground truth $o \in \{0, 1\}$:

$$\mathcal{L}_{\text{obj}} = -[o \log(\hat{o}) + (1 - o)\log(1 - \hat{o})]$$

Classification Loss For K classes, classification loss is the binary cross-entropy (or sometimes focal loss) applied to predicted class probabilities $\hat{c}_k$ and ground truth classes c_k:

$$\mathcal{L}_{\text{cls}} = -\sum_{k=1}^{K} [c_k \log(\hat{c}_k) + (1 - c_k)\log(1 - \hat{c}_k)]$$

Together, these losses guide YOLOv11 to predict accurate bounding boxes, confidently detect object presence, and correctly classify objects in a unified framework optimized end-to-end.

4. RESULTS

A. Model performance

After training, we scored an accuracy of 0.91. An important metric [3] for us is Mean Average Precision at threshold level 0.5, as it is important to recognize ingredient more than correctly placing the bounding box.

Metric	Mean	Standard Deviation
Precision	0.91	0.05
Recall	0.90	0.05
mAP@.50	0.93	0.06
mAP@.50-0.95	0.86	0.08

Validation Loss	Value
Epoch with Best Loss	150
Box Loss	0.29
Class Loss	0.28
DFL Loss	0.84

Table 2. Summary of Model Performance Metrics

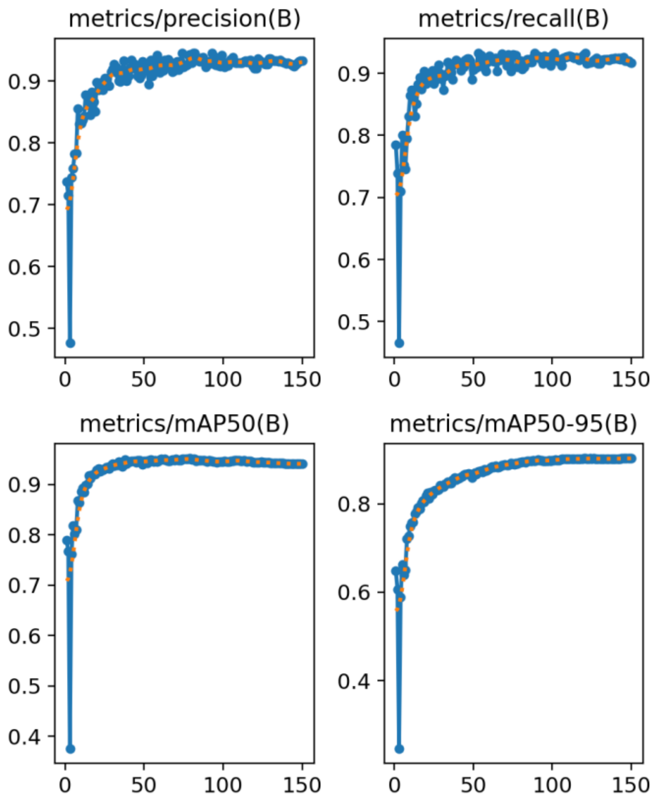


Fig. 7. Plot of performance metrics over epochs

Box Loss: Measures how accurately an object detection model predicts the location and size of bounding boxes around objects.
Class Loss: Quantifies how well the model correctly identifies the category or type of each detected object.
DFL Loss: Distribution Focal Loss improves the precision of bounding box predictions by optimizing the distribution of predicted coordinates rather than single point values.
Precision: The rate of true positives among all detections.
Recall: Recall measures the proportion of actual positive objects in an image that the model successfully identifies, indicating its ability to find all relevant instances.
mAP@50: This metric evaluates an object detection model's average precision across all object classes when a detected object is considered correct if its overlap with the ground truth box is at least 50%.
mAP@50-95: This metric evalutes the model's average precision across various stricter overlap thresholds, ranging from 50% to 95%.

Key fixes: To reduce class imbalance we sampled around the same amount from each class as so not to degrade model's performance.

CONCLUSION: Our model detects with high accuracy the ingredients and predicts their class correctly, showing good performance both in testing and training set.

B. Final product

Fig11. displays how the final product through our GUI looks like.

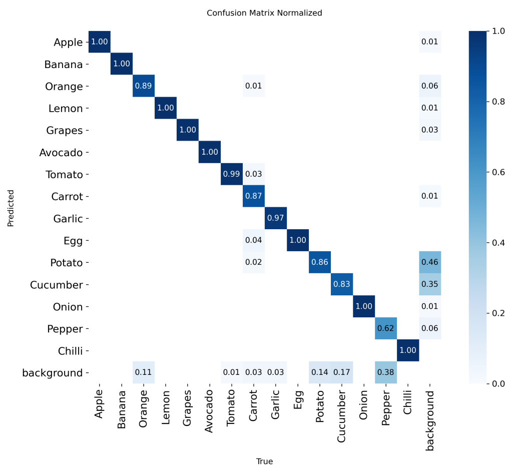


Fig. 8. Confusion matrix of the classes we trained on

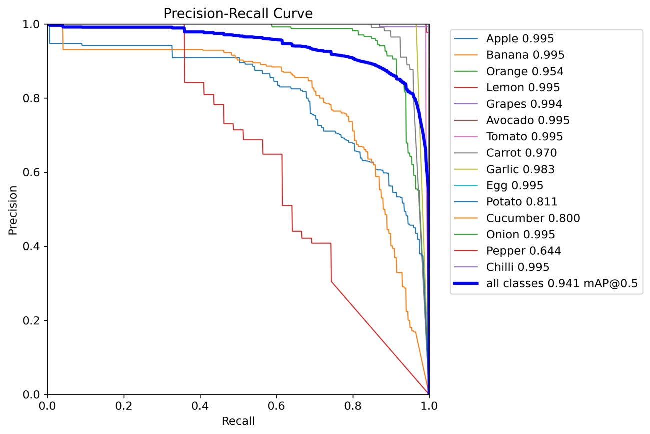


Fig. 9. Precision-Recall curve

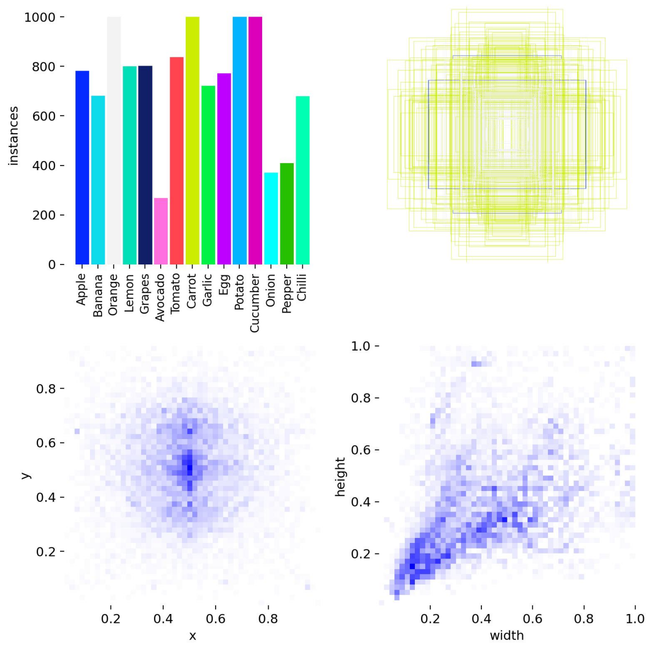
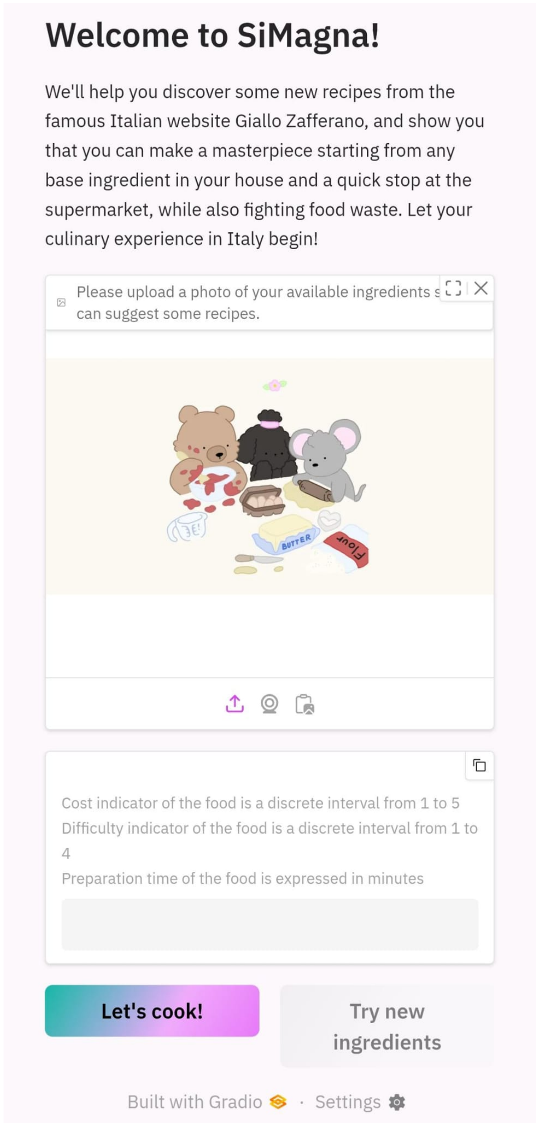


Fig. 10. Label and coordinates distribution

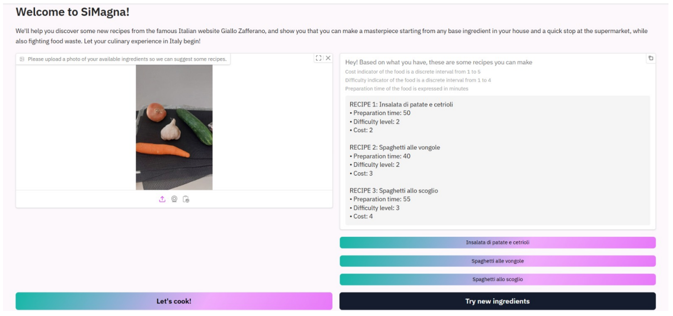

Welcome to SiMagna!

We'll help you discover some new recipes from the famous Italian website Giallo Zafferano, and show you that you can make a masterpiece starting from any base ingredient in your house and a quick stop at the supermarket, while also fighting food waste. Let your culinary experience in Italy begin!

Please upload a photo of your available ingredients s
can suggest some recipes.

Cost indicator of the food is a discrete interval from 1 to 5
Difficulty indicator of the food is a discrete interval from 1 to 4
Preparation time of the food is expressed in minutes

Fig. 11. Interface layout


Welcome to SiMagna!

We'll help you discover some new recipes from the famous Italian website Giallo Zafferano, and show you that you can make a masterpiece starting from any base ingredient in your house and a quick stop at the supermarket, while also fighting food waste. Let your culinary experience in Italy begin!

Please upload a photo of your available ingredients so we can suggest some recipes.

Hey! Based on what you have, these are some recipes you can make
Cost indicator of the food is a discrete interval from 1 to 5
Difficulty indicator of the food is a discrete interval from 1 to 4
Preparation time of the food is expressed in minutes

RECIPE 1: Insalata di patate e cetrioli
• Preparation time: 50
• Difficulty level: 2
• Cost: 2

RECIPE 2: Spaghetti alle vongole
• Preparation time: 40
• Difficulty level: 2
• Cost: 3

RECIPE 3: Spaghetti allo scoglio
• Preparation time: 55
• Difficulty level: 3
• Cost: 4

Fig. 12. Example of image upload and recipe suggestion

C. Future perspective

This project is done keeping in mind some limitations.
Scalability: The hardest challenge of our project was to collect the data. We had to go through many datasets, some of which did not have promising data, or even compatible data for object detection purposes. We believe that having access to labeled clean data plays a significant role in our model's scalability
DL is a blackbox: It is hard to see or visualize what the model is learning. The latent space is often not interpretable, thus leading to the black box curse of deep learning. However, for YOLO models the architecture is still interpretable in a high language manner.
GUI: Gradio is a safe, nice and practical choice to interact with the user. However given more time and hosting power, the user experience can be rethought and made more enjoyable through web or mobile applications.

5. FINAL REMARKS

We strongly suggest referring to out Git repository to look at all the files or even to clone the repository by doing

```
git clone https://github.com/ppietropiccolo/siMagna.git
```

6. REFERENCES

To understand the mechanism behind the YOLO model, but also to create a general idea of the pipeline behind loading and fine-tuning pre-trained models we heavily relied on the following papers, some of which are Master thesis dissertations.

REFERENCES

1. K. Patel, "Fruits & vegetable detection for yolov4," https://scholarworks.calstate.edu/downloads/g158bm85x (2020). California State University ScholarWorks.
2. Ultralytics, "YOLOv11: The latest ultralytics object detection model," https://docs.ultralytics.com/models/yolo11/ (2024).
3. Ultralytics, "Object detection metrics — ultralytics yolo docs," https://docs.ultralytics.com/guides/yolo-performance-metrics/#object-detection-metrics (2024).