Project Meal Suggestion

Patrick Bonack, Sebastian Heinemann, Nick Wagner, Felix Lapp, Ya Jiang

March 2021

Table of Contents

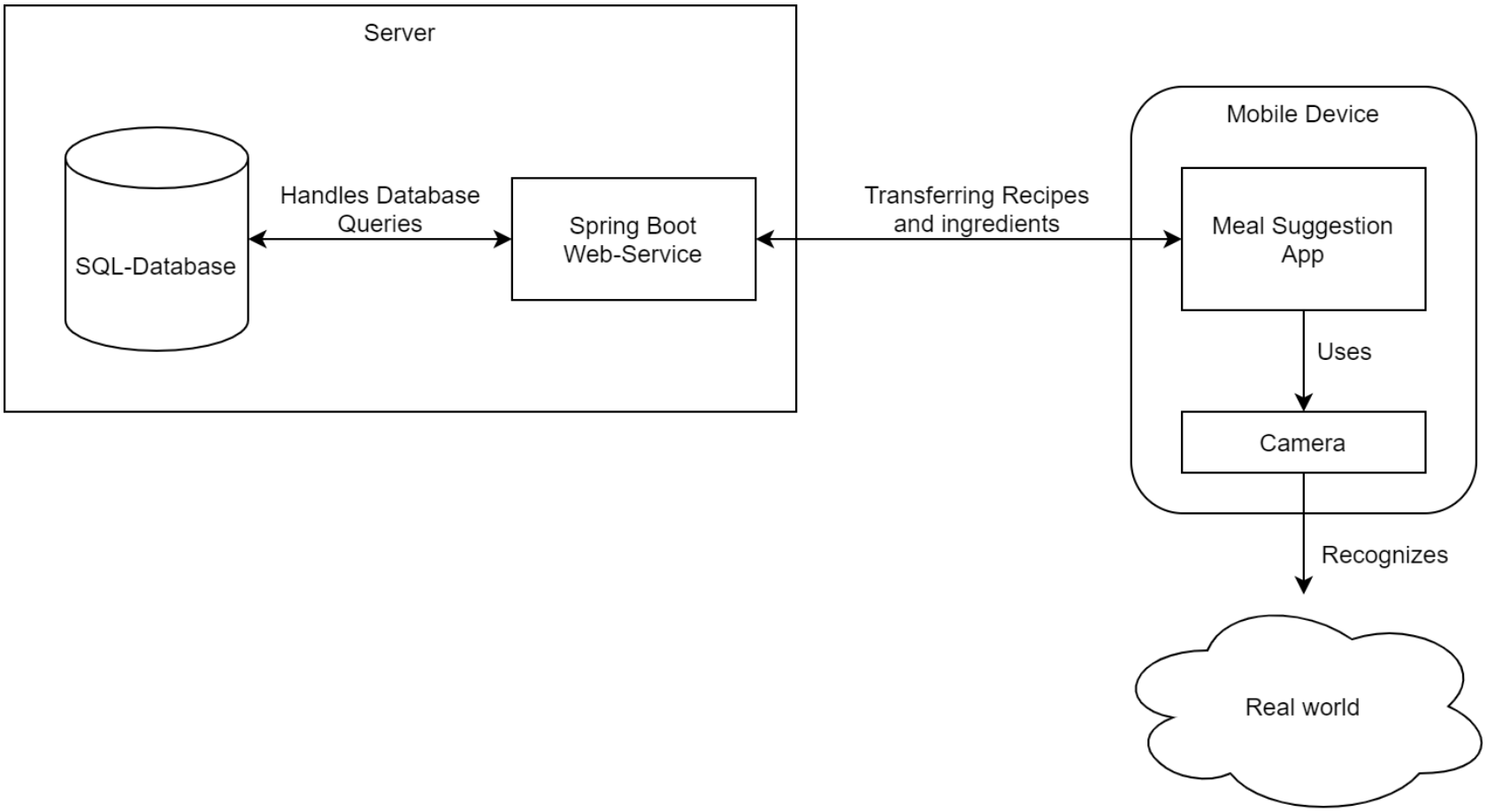
Intro

Introduction

- Meal Suggestion App helps the user decide what to cook based on the ingredients the user has at their disposal
- User selects list of ingredients
  - Either by taking a picture using AI for image recognition
  - or by explicit search
- App displays recipes containing selected ingredients
- database for recipes and ingredients
- simulation for testing image recognition

Design

System


Server
SQL-Database
Handles Database Queries
Spring Boot Web-Service
Transferring Recipes and ingredients
Mobile Device
Meal Suggestion App
Uses
Camera
Recognizes
Real world

Frontend

- Lets user select ingredients
  - By explicit search
  - By taking pictures of ingredients
- Shows list of recipes which can be cooked using selected ingredients
- Provides a manual for each recipe

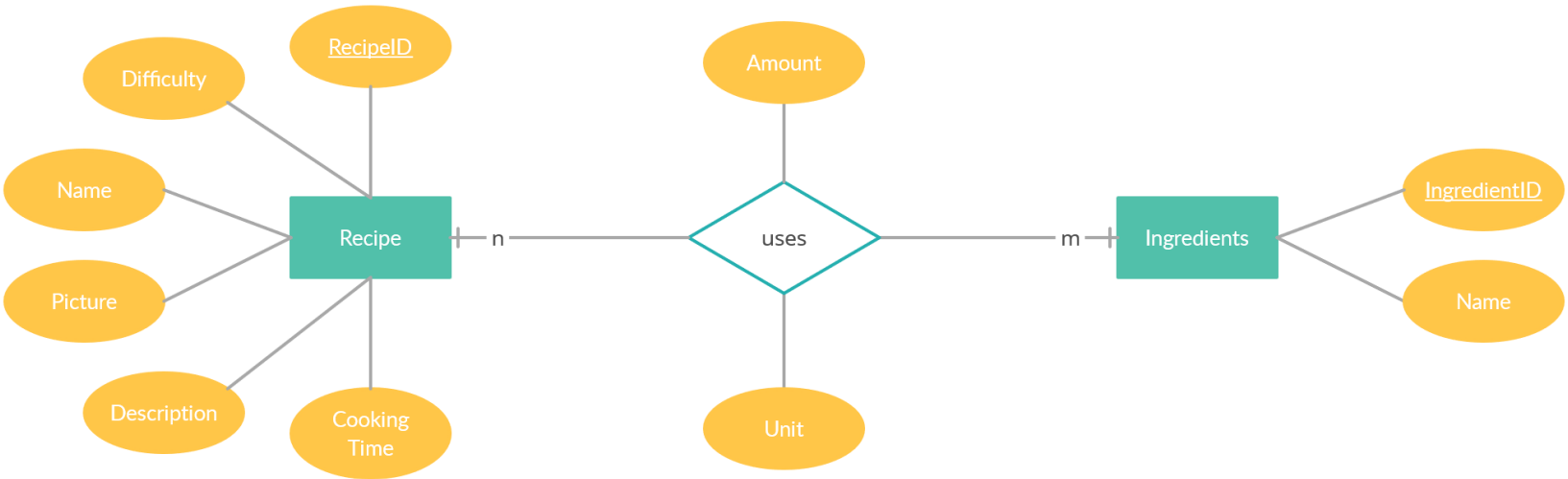
Backend

- Connects database and mobile devices
- Provides HTTP-endpoints for accessing ingredients and recipes
- On same device as database
- Implemented using a Spring Boot service
- Manages database queries
- Prepares data to be processed by the mobile devices

Backend - Endpoints

Endpoint	Method	Parameters	Description
`/recipes`	GET	None	List of recipes formatted using JSON
`/recipes`	POST	Request Body: List of IDs formatted using JSON	List of recipes which are containing all ingredients represented by the given IDs
`/ingredients`	GET	None	List of ingredients formatted using JSON

Database


RecipeID
Difficulty
Name
Picture
Description
Cooking Time
Recipe
n
uses
m
Amount
Unit
Ingredients
IngredientID
Name

Database

- Ingredients which can be recognized by the AI need to be a subset of the ingredients in the database
- Database uses MariaDB
- App can access database through the backend

Simulation

Task

1. Simulate making pictures that are used as input data
2. Annotate these pictures to enable the models training and validation

Simulation of ingredients and food environment

Requirements

1. Realtime is not needed
2. Physically based rendering
3. Automatically generation
4. Providing Annotations
5. Open source

The decision was made in favour of **Blender** (version 2.92) with its included **Python** version 3.7

Simulation of ingredients and food environment

To be general as possible the image generator should cover all possible scenarios. The scenarios differ in the set of World Parameter

World Parameter

1. Camera: location, rotation, focus, resolution, bloom
2. Illumination: color, intensity, spectacular
3. Food: position, location, rotation, size, visibility

Simulated Images

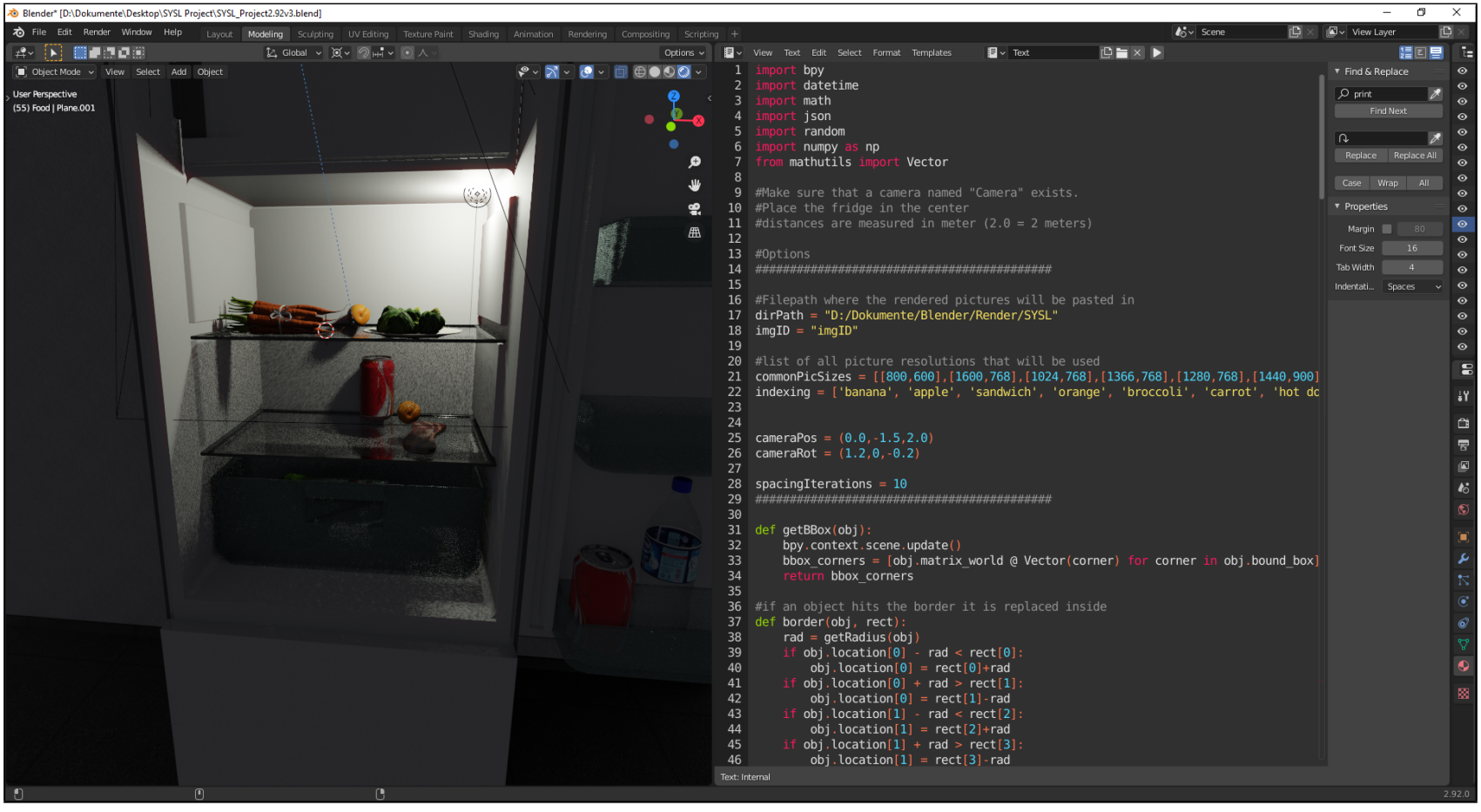
Simulation of ingredients and food environment

How does the simulator work?

1. Run the python script from the Blender project to create the images and masks
2. On the image output directory, Run the annotation script to create the annotation files for each image

The mask are required for creating annotations

The Simulator




```python
import bpy
import datetime
import math
import json
import random
import numpy as np
from mathutils import Vector

#Make sure that a camera named "Camera" exists.
#Place the fridge in the center
#distances are measured in meter (2.0 = 2 meters)

#Options
########################################

#Filepath where the rendered pictures will be pasted in
dirPath = "D:/Dokumente/Blender/Render/SYSL"
imgID = "imgID"

#list of all picture resolutions that will be used
commonPicSizes = [[800,600],[1600,768],[1024,768],[1366,768],[1280,768],[1440,900]
indexing = ['banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dc


cameraPos = (0.0,-1.5,2.0)
cameraRot = (1.2,0,-0.2)

spacingIterations = 10
########################################

def getBBox(obj):
    bpy.context.scene.update()
    bbox_corners = [obj.matrix_world @ Vector(corner) for corner in obj.bound_box]
    return bbox_corners

#if an object hits the border it is replaced inside
def border(obj, rect):
    rad = getRadius(obj)
    if obj.location[0] - rad < rect[0]:
        obj.location[0] = rect[0]+rad
    if obj.location[0] + rad > rect[1]:
        obj.location[0] = rect[1]-rad
    if obj.location[1] - rad < rect[2]:
        obj.location[1] = rect[2]+rad
    if obj.location[1] + rad > rect[3]:
        obj.location[1] = rect[3]-rad
```

Creating Images

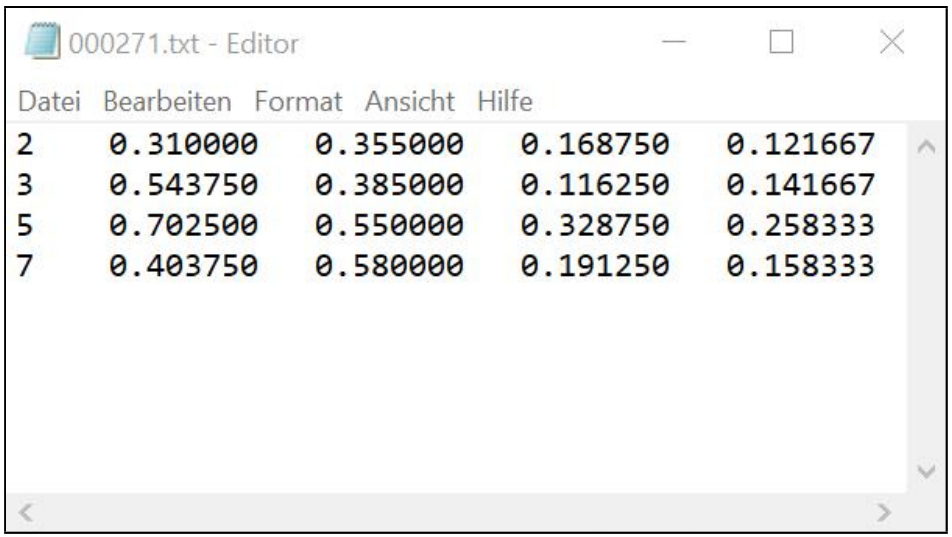
The blender script provides following functionalities

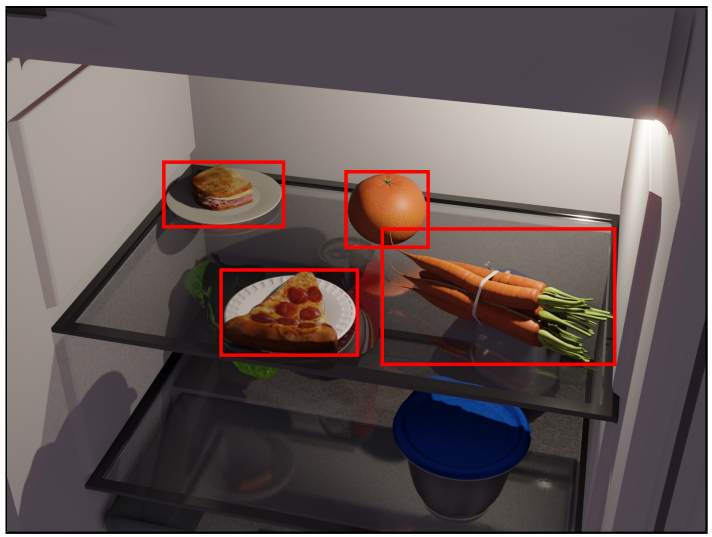
1. Choose and set the world parameter randomly
2. Choose a set of the food elements and place it in the fridge
3. Render and save the images
4. For every ingredient of the image create a mask
5. Loop that for all images

Creating Annotations

Models object array:
['banana', 'apple', 'sandwich', 'orange', 'broccoli','carrot', 'hot dog', 'pizza', 'donut', 'cake']

Annotation:

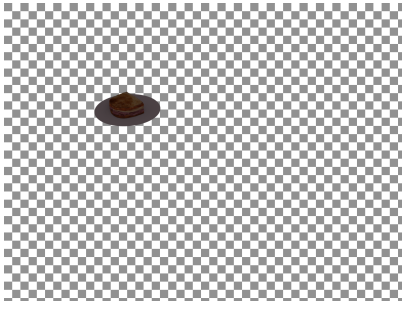




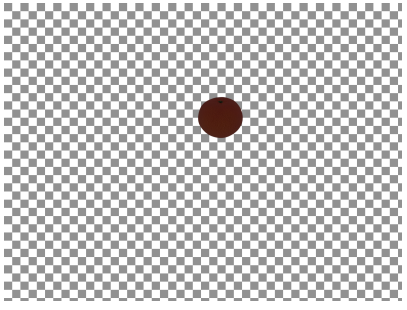
Creating Annotations

image:

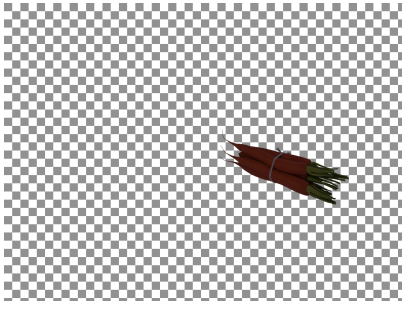
Masks:



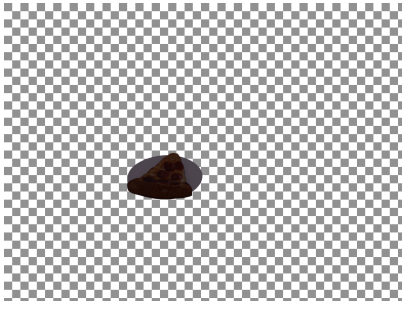
2.png



3.png



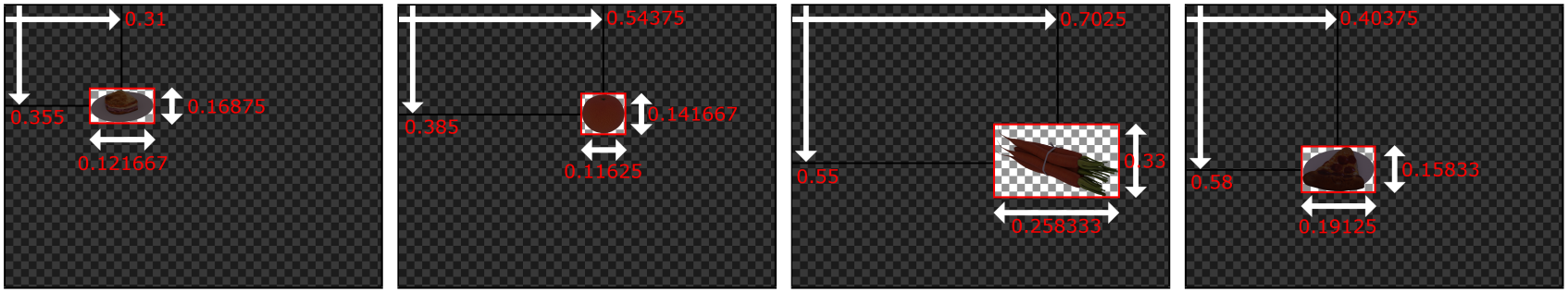
5.png



7.png

Creating Annotations

Masks:



Annotation:

000271.txt - Editor

Datei Bearbeiten Format Ansicht Hilfe

2	0.310000	0.355000	0.168750	0.121667
3	0.543750	0.385000	0.116250	0.141667
5	0.702500	0.550000	0.328750	0.258333
7	0.403750	0.580000	0.191250	0.158333

Detection

Ingredient detection

Scope:

1. Modifying a pre-trained YOLOv5 model
2. Extracting food-relevant data from Coco Dataset
3. Retrain model on coco data
4. Compare models on coco and simulation data
5. Create an android-app able to detect and highlight ingredients

The base model

YOLOv5

- Object detection model
- Open-source implementation of latest version of YOLO
- Based on Darknet
- Open Source

Yolov5s

- Smallest and fastest of pre-trained models
- Trained on training dataset provided by COCO
- Uses all 80 COCO classes as output
- Problem: We are only interested in 10 food-classes
- Solution: Ignore every non-food prediction

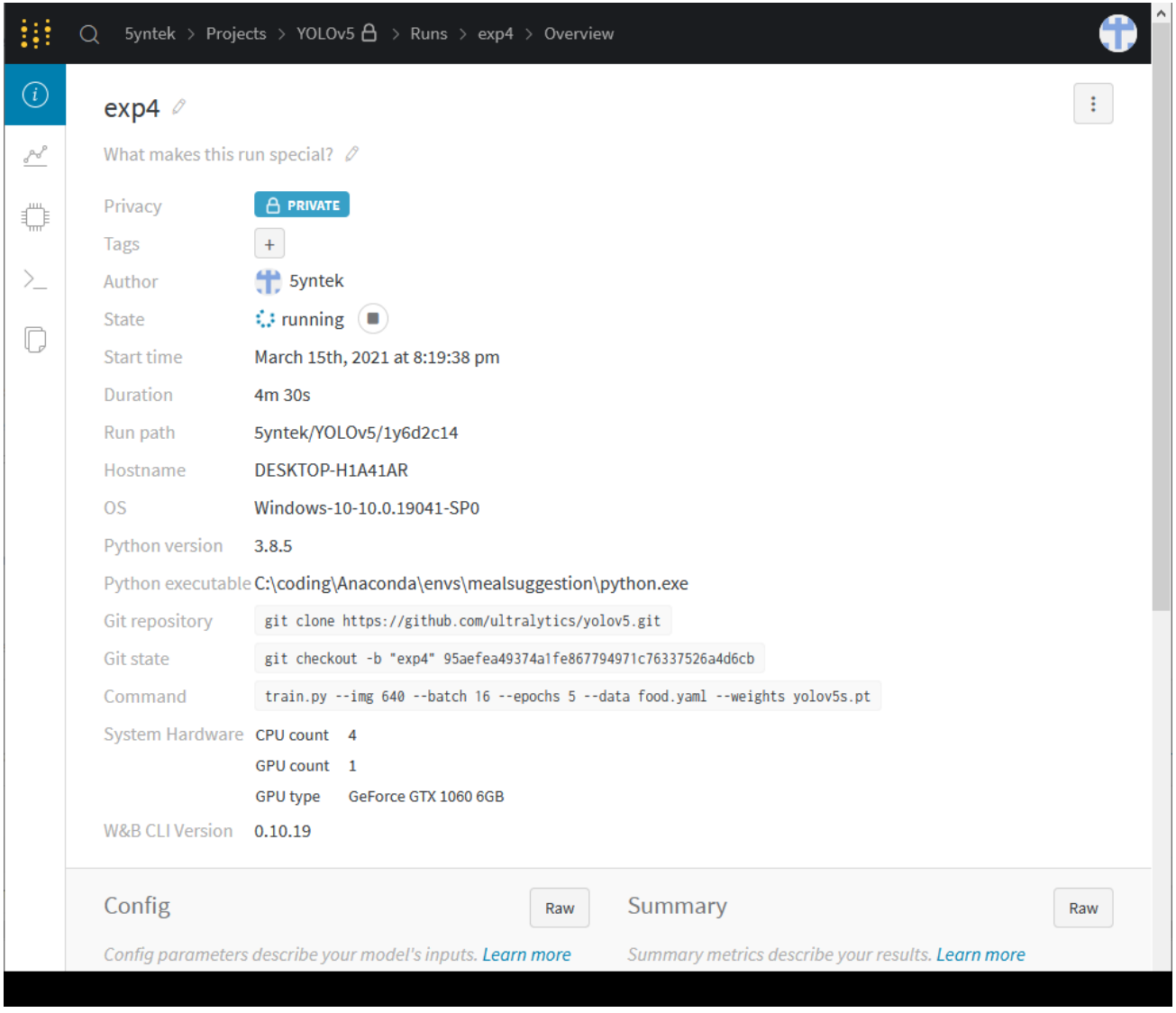
Training-pipeline

- Download last Coco-Data from 2017
- Extract only relevant Data
- Create labels that are usable for YOLO which is using the Darknet format
- Organize Directories of Data
- Create a `.yaml` file describing the classes and Data-Location
- Train the YOLOv5 model on COCO-Data

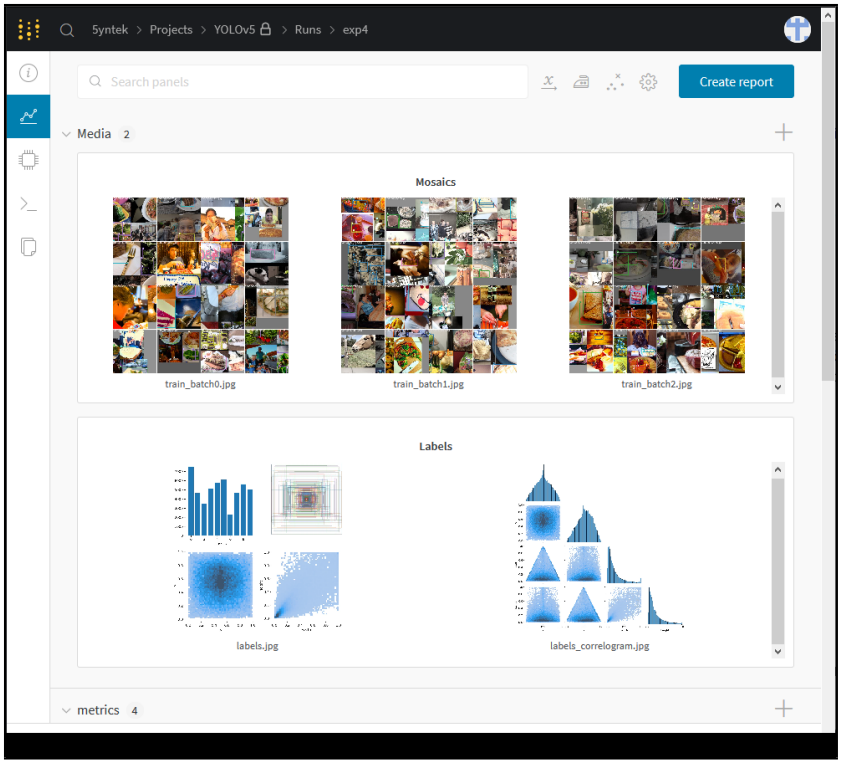
Training

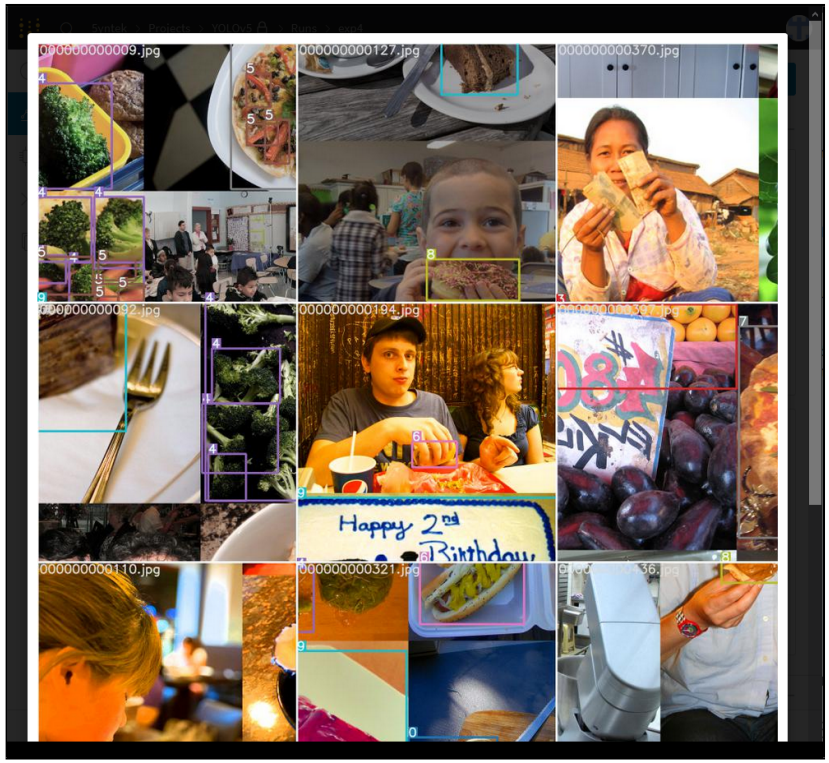
- 16.255 training images
- 708 validation images
- 10 classes: 'banana', 'apple', 'sandwich', 'orange', 'broccoli','carrot', 'hot dog', 'pizza', 'donut', 'cake'
- `python train.py --img 640 --batch 16 --epochs 5 --data food.yaml --weights yolov5s.pt`
- Automated logging and visualisation with Wandb

Wandb: run overview



Wandb: Analysis while training





Evaluation

Evaluation of implementation

Two models to compare:

- Pre-trained YOLOv5s
- Our retrained model

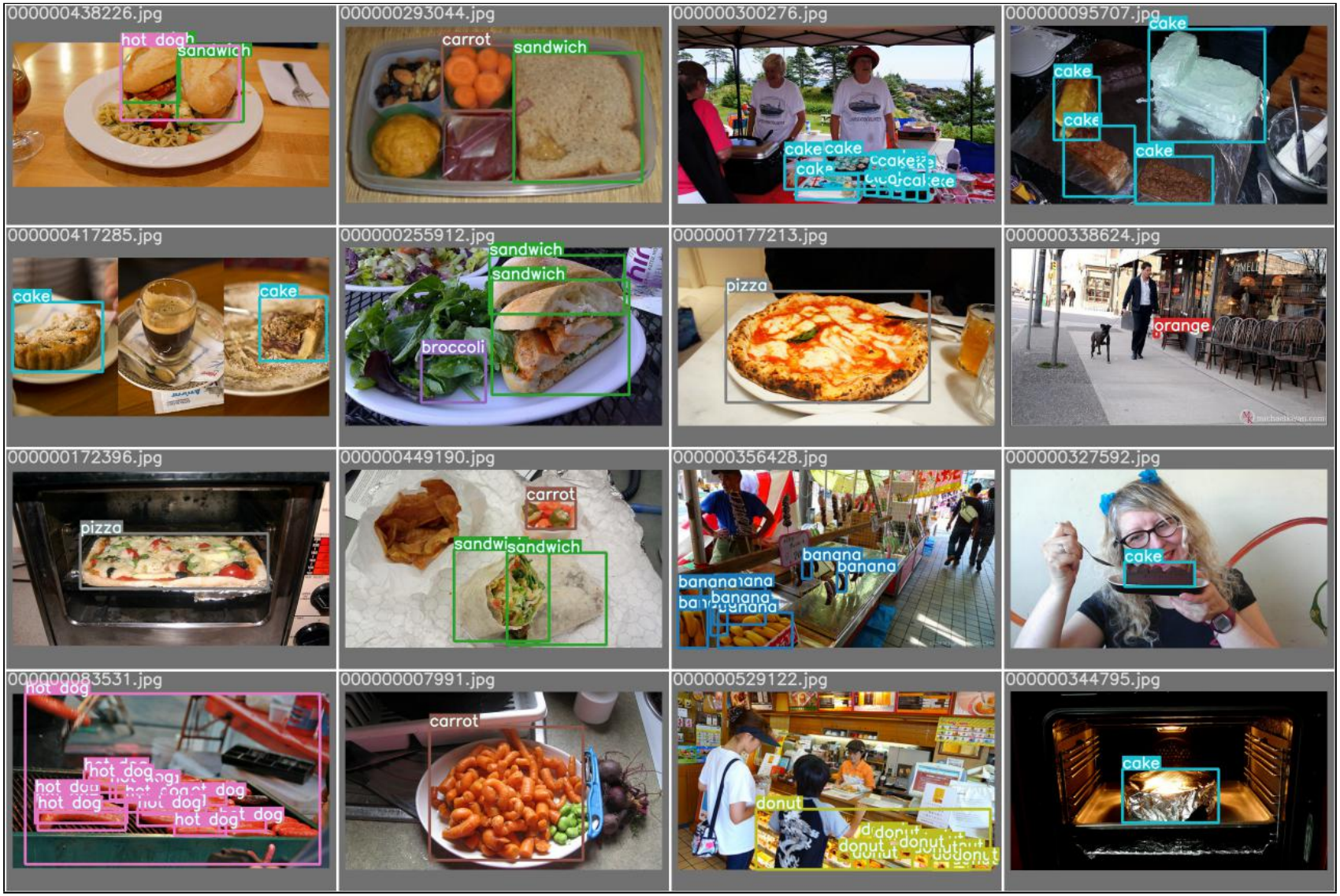
Two Data Sources:

- COCO validation set
- Simulated pictures

Evaluation with YOLOv5

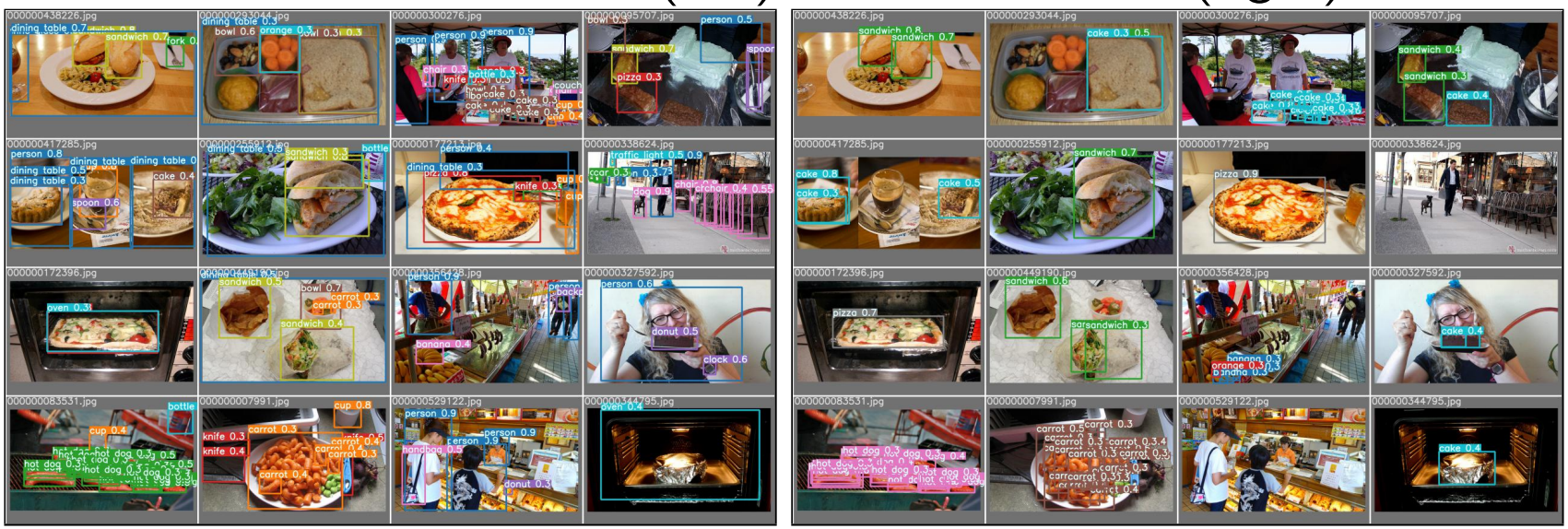
- Provided evaluation tool `yolov5/test.py`
- `!python test.py --weights <name>.pt --data <name>.yaml --img 640 --iou 0.65 weights`
- One `.yaml` file for each combination
- Two model files `.pt`

Batch comparison COCO



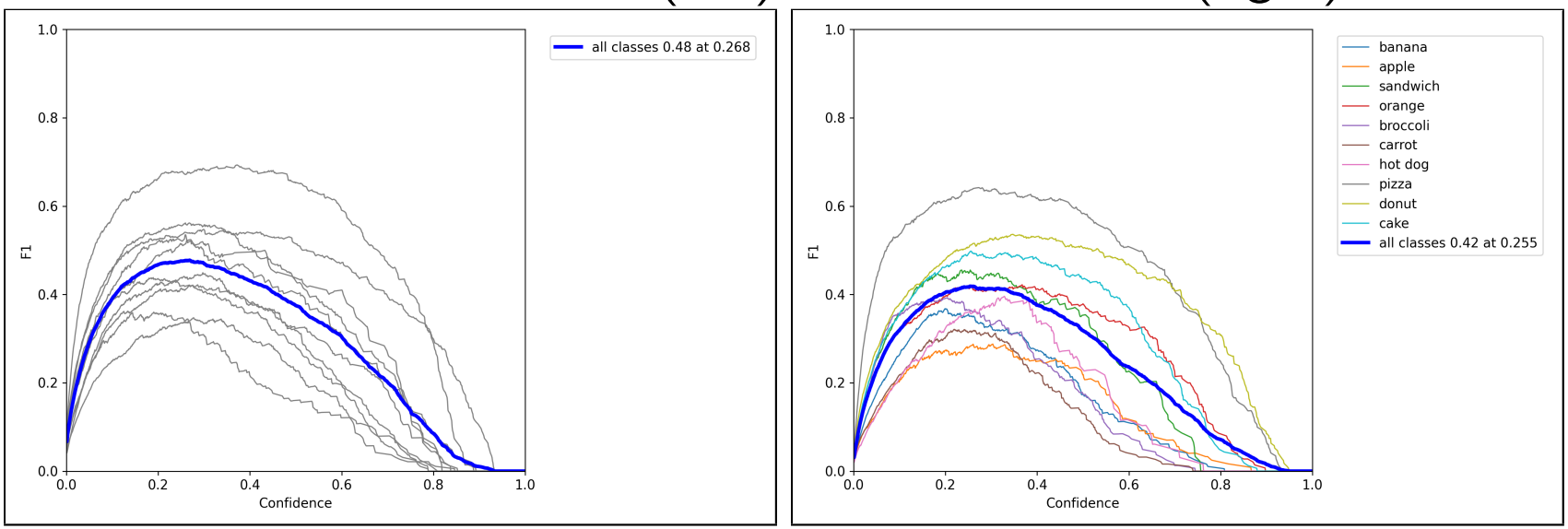
Batch comparison COCO

Pre-trained model (left) & trained model (right)

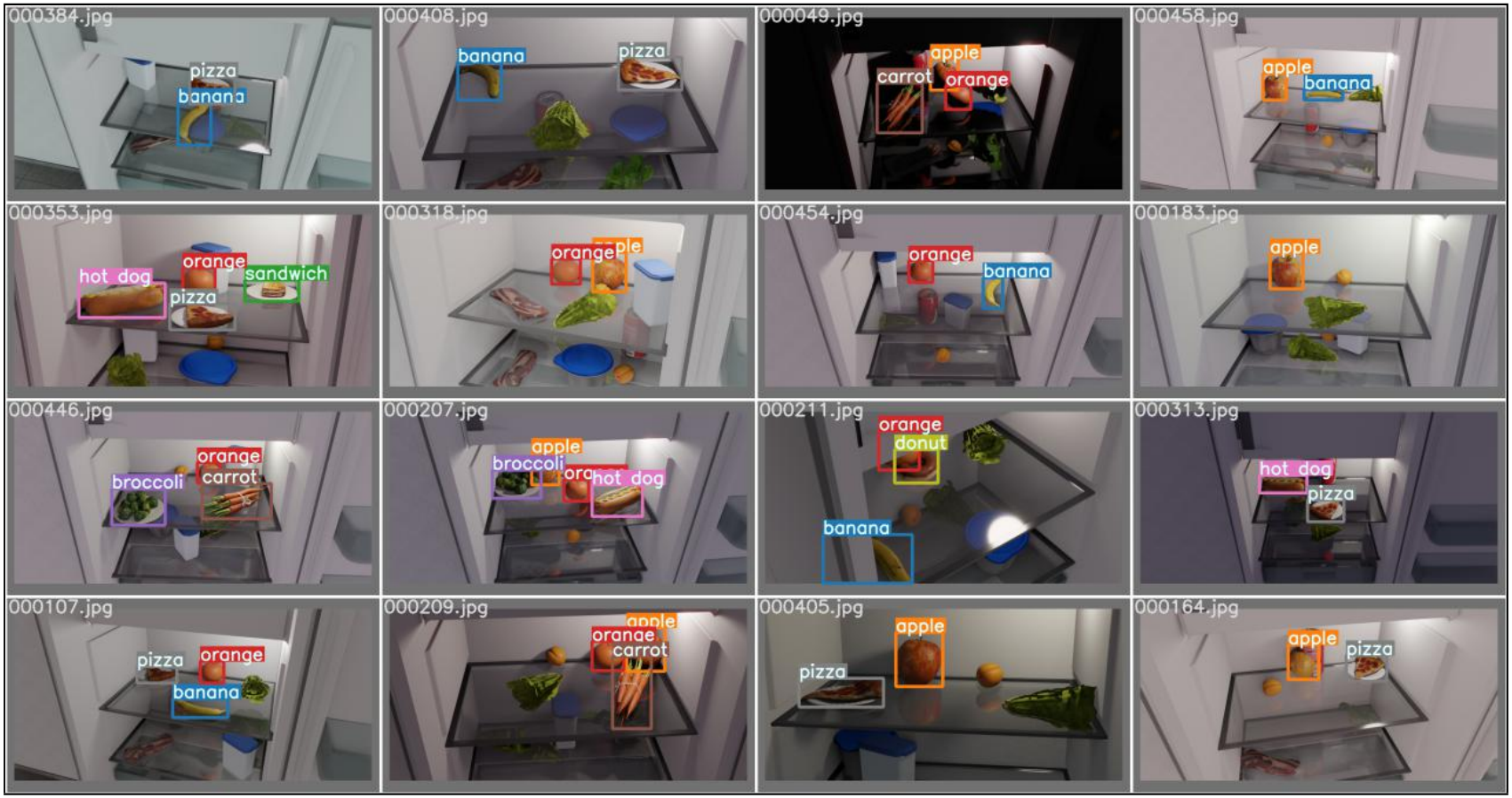


F1 COCO

Pre-trained model (left) & trained model (right)



Batch comparison Simulated

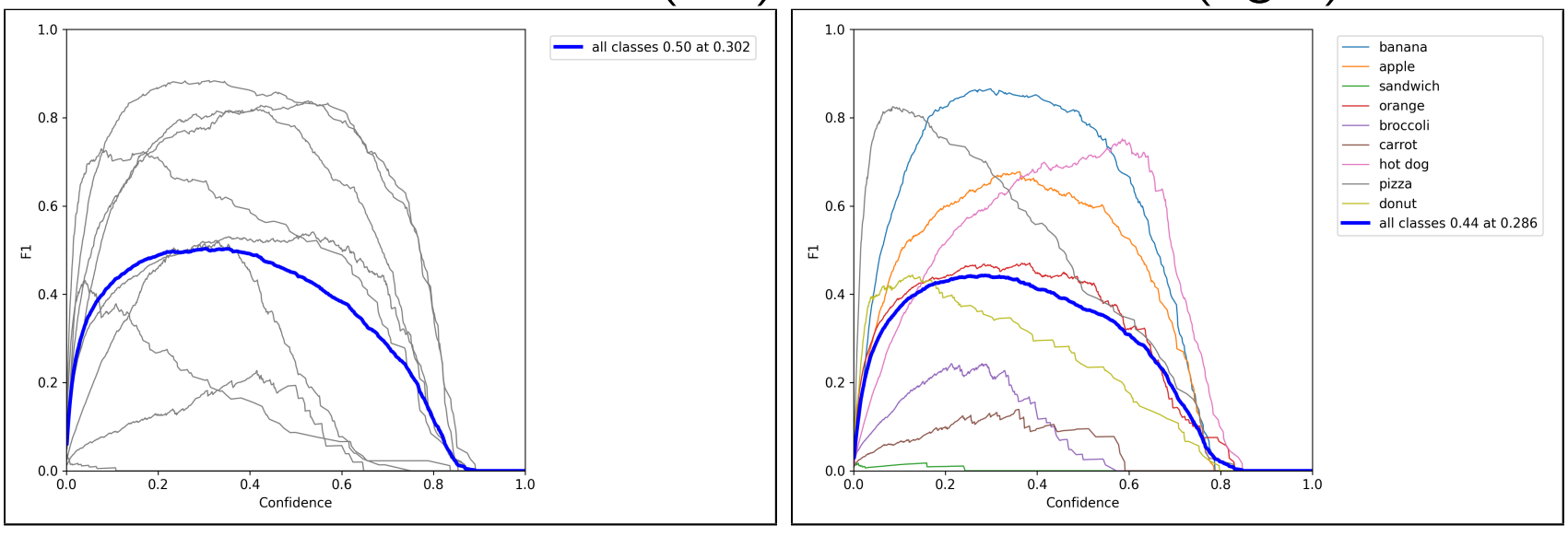


Batch comparison Simulated

Pre-trained model (top) & trained model (bottom)

F1 Simulated

Pre-trained model (left) & trained model (right)



Manual comparison

- 3×16 simulated pictures with total of 128 labeled objects
  - FP: objects being predicted as a wrong class
  - FN: missing predictions of labeled objects.
  - TP: objects being predicted correctly
- Every predicted class not being an ingredient is ignored
- Only object with highest score considered

	True	False	*ground truth*
True	92	30	
False	37	-	
prediction			

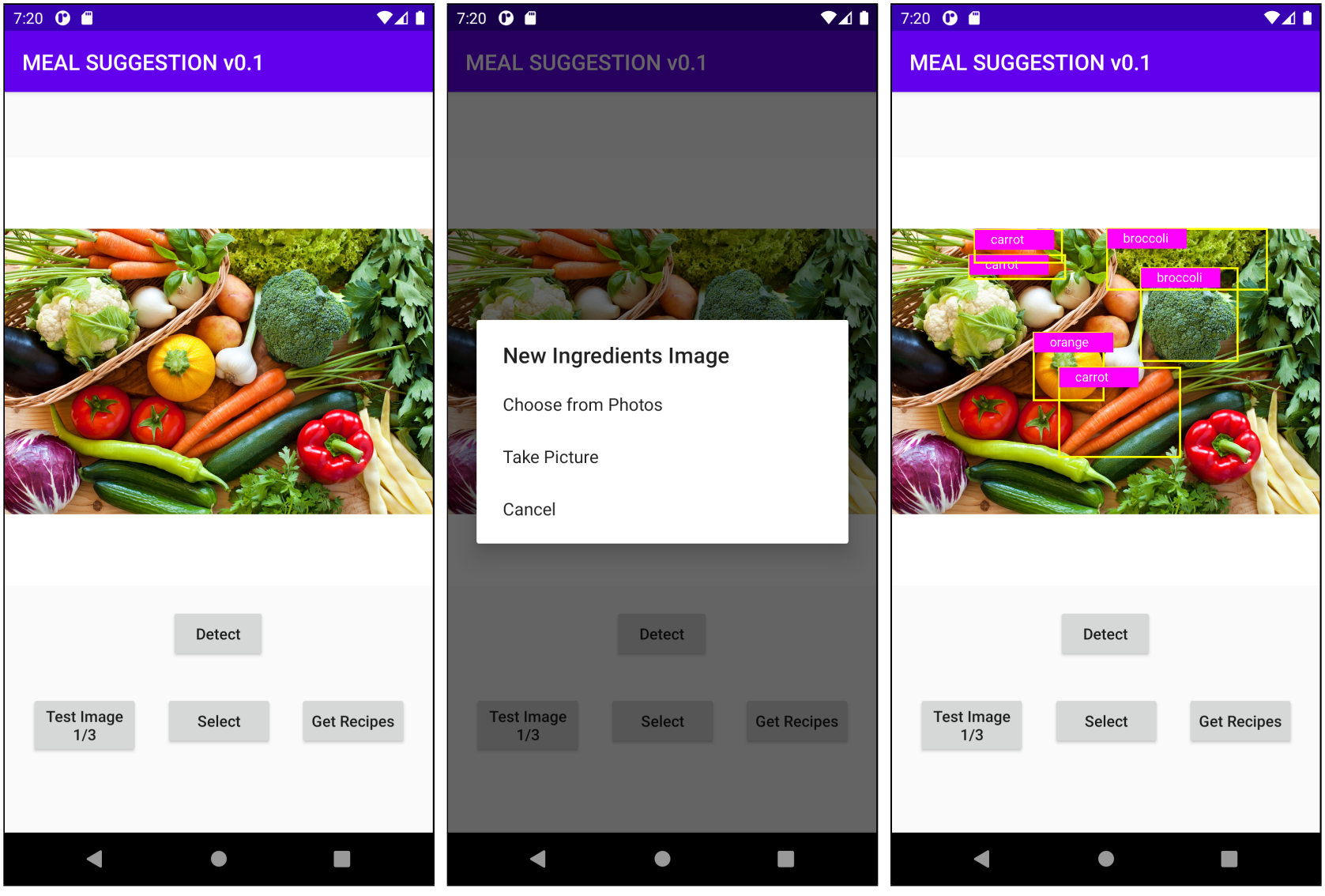
pre-trained model

	True	False	*ground truth*
True	78	20	
False	50	-	
prediction			

trained model

App

Detection: App



Live Demo