# Project 3: Report

***Submitted by***

- ***Soufiane Lamchoudi 106816***
- ***Meryem Abdallah 75657***

***Supervised by: Dr. Asmae Mourhir***

Computational Neural Network
CSC5351-01

# Table of Contents

جامعة الأخوين
AL AKHAWAYN
UNIVERSITY

## Problem Statement Introduction:

Over recent years, emerging interest has occurred in integrating computer vision technology into the retail industry. Automatic checkout (ACO) is one of the critical problems in this area which aims to automatically generate the shopping list from the images of the products to purchase. The main challenge of this problem comes from the large scale and the fine-grained nature of the product categories as well as the difficulty for collecting training images that reflect the realistic checkout scenarios due to continuous update of the products. Despite its significant practical and research value, this problem is not extensively studied in the computer vision community, largely due to the lack of a high-quality dataset. To fill this gap, in this work we propose a new dataset to facilitate relevant research. Our dataset enjoys the following characteristics: (1) It is by far the largest dataset in terms of both product image quantity and product categories. (2) It includes single-product images taken in a controlled environment and multi-product images taken by the checkout system. (3) It provides different levels of annotations for the checkout images. Comparing with the existing datasets, ours is closer to the realistic setting and can derive a variety of research problems. Besides the dataset, we also benchmark the performance on this dataset with various approaches.

## Converting a custom dataset from COCO format to YOLO format:

RPC dataset is using the same format as COCO. However, To train the yolov7 model, our custom dataset must be in the YOLO format and if not, online tools are available that will convert our custom dataset into the required format. Similarly, if the dataset is in COCO format, we can use online tools to convert it from COCO (JSON) format into YOLO format. In this project, instead of using online tools, we will build our own functions to convert the dataset into a YOLO format step by step.

In [13]:

```
#Load image data for trainset
def load_images_from_folder_train(folder):
  """
  Reading source trainig image from the current dataset, renaming it according to the relative labels file
  and saving to the new dataset location.
  Also saving image’s filenames for further use.
  """
  count = 0
  for filename in os.listdir(folder):
        source = os.path.join(folder,filename)
        destination = f"{output_path}images/img{count}.jpg"

        try:
            shutil.copy(source, destination)
            print("File copied successfully.")
        # If source and destination are same
        except shutil.SameFileError:
            print("Source and destination represents the same file.")

        file_names.append(filename)
        count += 1

load_images_from_folder_train(input_path)
```

otebooks/Assignment_3.ipynb#

In [16]:
```python
#for training set
def get_img_ann_train(image_id):
  """
  This function takes an image_id as a parameter and returns the annotations of that image.
  """
  img_ann = []
  isFound = False
  for ann in train_js['annotations']:
    if ann['image_id'] == image_id:
      img_ann.append(ann)
      isFound = True
  if isFound:
    return img_ann
  else:
    return None
```

#### Processing Labels

Following are the steps we are going to perform in conversion:

- Extracting image information such as image_id, image_width, image_height, etc.
- Get annotations for this image using image_id.
- Open a text file for this image in the output path given by the user.
- Extract bounding box properties for each object in the image.
- Finding midpoint coordinates.
- Apply Normalization.
- Setting precision.
- Writing the updated annotations for this image into a text file.
- After processing through all the annotations for the current image, close the text file.
- Repeat the steps for all images.

In [23]:
```python
#Labeling For Training Set
count = 0

for filename in file_names:
  # Extracting image
  img = get_img_train(filename)
  img_id = img['id']
  img_w = img['width']
```

# Fine Tuning the Model:

## Creating the Custom Dataset YAML File

Like many of the recent YOLO versions, we will need a dataset YAML file to train any of the YOLOv7 models. This .yaml file contains the paths to the image sets, the number of classes, and the name of the classes. So, basically, we change the classes of the soft max.

In [29]:
```
%%writefile ./data/costum.yaml
train: ../yolo_train_dataset/images
val: ../yolo_val_dataset/images
test: ../yolo_test_dataset/images

# Classes
nc: 200  # number of classes
names:  ['1_puffed_food', '2_puffed_food', '3_puffed_food', '4_puffed_food','5_puffed_food', '6_puffed_food', '7_puffed_food
        '15_dried_fruit', '16_dried_fruit', '17_dried_fruit',
        '18_dried_fruit', '19_dried_fruit', '20_dried_fruit',
        '21_dried_fruit', '22_dried_food', '23_dried_food',
        '24_dried_food', '25_dried_food', '26_dried_food', '27_dried_food',
        '28_dried_food', '29_dried_food', '30_dried_food',
        '31_instant_drink', '32_instant_drink', '33_instant_drink',
        '34_instant_drink', '35_instant_drink', '36_instant_drink',
        '37_instant_drink', '38_instant_drink', '39_instant_drink',
        '40_instant_drink', '41_instant_drink', '42_instant_noodles',
        '43_instant_noodles', '44_instant_noodles', '45_instant_noodles',
        '46_instant_noodles', '47_instant_noodles', '48_instant_noodles',
        '49_instant_noodles', '50_instant_noodles', '51_instant_noodles',
        '52_instant_noodles', '53_instant_noodles', '54_dessert',
        '55_dessert', '56_dessert', '57_dessert', '58_dessert',
        '59_dessert', '60_dessert', '61_dessert', '62_dessert',
        '63_dessert', '64_dessert', '65_dessert', '66_dessert',
```

## Tiny YOLOv7 Fine Tuning:

Next let's define the network architecture for YOLOv7 that we will use for our project. In this section, we will train the YOLOv7-Tiny model. The tiny model contains just over 6 million parameters. We will use as a resolution images for training the model, which is 416×416. But before we can start the training, there are a few other details that we need to take care of.

**Note:** We would like to mention that we got inspired from an existing architecture that was used almost for the same purpose which is detecting retail store items.

## Download the Tiny Model Weights:

First, we need to download the YOLOv7-tiny model.

In [ ]:

```
# Download the Tiny model weights.
!wget https://github.com/WongKinYiu/yolov7/releases/download/v0.1/yolov7-tiny.pt
```

This will download the latest version of the YOLOv7-tiny model which has been pre-trained on the COCO dataset.

## Define the Architecture:

Next, we need to configure the YOLOv7-tiny model for product detection training. There are several default configuration files inside yolov7/cfg/training/ directory. All these contain the model configuration. We need to configure the yolov7-tiny.yaml file. For that, we will create a copy of that file, rename it, and configure it accordingly.

The following code block creates a yolov7s.yaml file.

[31]:

```
%%writefile ./cfg/training/yolov7s.yaml
# parameters
nc: 200  # number of classes
depth_multiple: 0.33  # model depth multiple
width_multiple: 0.50  # layer channel multiple

# anchors
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32

# yolov7-tiny backbone
backbone:
  # [from, number, module, args] c2, k=1, s=1, p=None, g=1, act=True
  [[-1, 1, Conv, [32, 3, 2, None, 1, nn.LeakyReLU(0.1)]],  # 0-P1/2

  [-1, 1, Conv, [64, 3, 2, None, 1, nn.LeakyReLU(0.1)]],  # 1-P2/4

  [-1, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
  [-2, 1, Conv, [32, 1, 1, None, 1, nn.LeakyReLU(0.1)]],
  [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
  [-1, 1, Conv, [32, 3, 1, None, 1, nn.LeakyReLU(0.1)]],
  [[-1, -2, -3, -4], 1, Concat, [1]],
  [-1, 1, Conv, [64, 1, 1, None, 1, nn.LeakyReLU(0.1)]],  # 7
```

## Transfer Learning:

Now we start the training process. we defined the image size (img) to be 416x416, batch size 32 and the model is run for 10 epochs. Since we will use transfer learning, we use the tiny model weights for our detection system.

```
In [37]:
!python ./train.py --epochs 10 --workers 4 --device 0,1,2,3  --batch-size 32 --data ./data/costum.yaml --img 416 --cfg ./cfg
```

```
YOLOR 🚀 v0.1-116-g8c0bf3f torch 1.13.0+cu117 CUDA:3 (NVIDIA RTX A4000, 16117.3125MB)

Namespace(adam=False, artifact_alias='latest', batch_size=32, bbox_interval=-1, bucket='', cache_images=False, cfg='./cfg/training/yolov7s.yaml', data='./data/costum.yaml', device='3', entity=None, epochs=10, evolve=False, exist_ok=False, freeze=[0], global_rank=-1, hyp='./data/hyp.scratch.tiny.yaml', image_weights=False, img_size=[416, 416], label_smoothing=0.0, linear_lr=False, local_rank=-1, multi_scale=False, name='yolov7_results', noautoanchor=False, nosave=False, notest=False, project='runs/train', quad=False, rect=False, resume=False, save_dir='runs/train/yolov7_results3', save_period=-1, single_cls=False, sync_bn=False, total_batch_size=32, upload_dataset=False, v5_metric=False, weights='', workers=4, world_size=1)
tensorboard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, fliplr=0.5, mosaic=1.0, mixup=0.05, copy_paste=0.0, paste_in=0.05, loss_ota=1
```

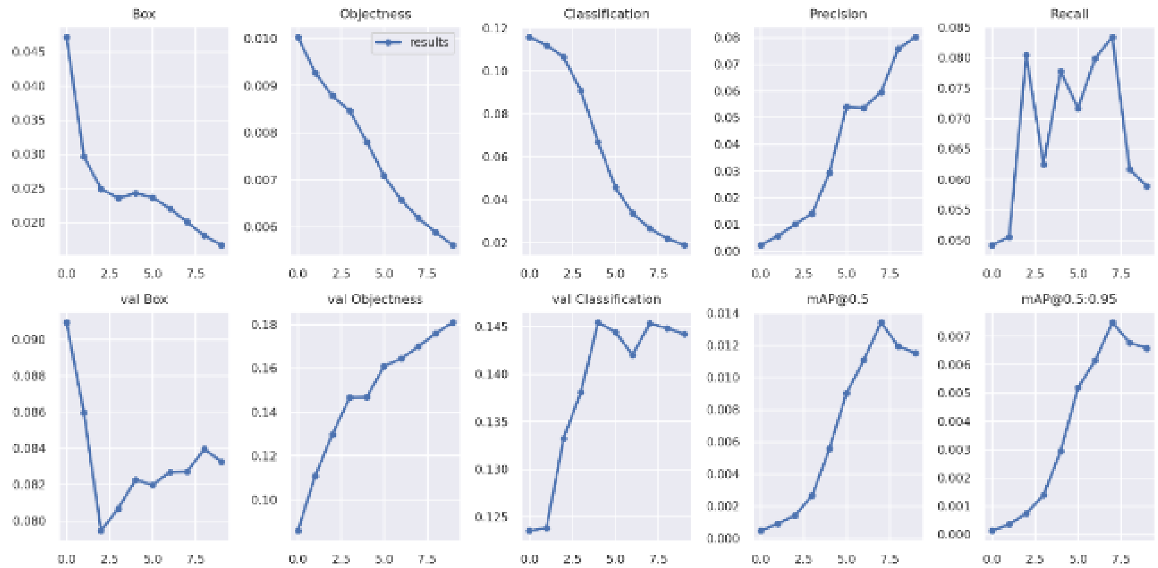
# Results and Evaluation:

## Original Labels:

## Prediction:


We can clearly see; our model did not predict the bounding box with high accuracy. However, the explanation behind this problem is we did not train our model with high number of epochs. In the following we will see the evaluation of our model.

جامعة الأخوين
AL AKHAWAYN
UNIVERSITY




- mAP is the mean Average Precision telling how correct are our bounding box prediction s on average. It is area under curve of precision-recall curve.
- Objectness shows the probability that an object exists in an image. Here it is used as los s function.

It is seen that loss and objectness loss decrease both for training and validation. Mean Average Precision (mAP) however is at 0.012 for bounding box IoU threshold of 0.5. Recall stands at 0.06 as shown below: