Fruit Detection in Orchards - Vision Transformer Vs Well Established Approaches

Introduction

As the worlds population grows and socio-cultural and lifestyle-driven factors raise awareness for a healthy and sustainable diet, the demand for fresh fruit and vegetables is growing. These circumstances put a lot of pressure on the current fruit market to produce fruit quantities that satisfy the growing demand, while at the same time ensuring a low, competitive pricing and ensuring sustainable farming practices (OECD-FAO Agricultural Outlook, 2021). Achieving these goals in a labour-intensive environment such as fruit orchards presents farmers with a lot of challenges (Zhang et al., 2022). In order to reduce cost, farmers have to reduce the manual labour involved in the management of orchards, as well as keeping production at high yields with high quality. In order to achieve this, fruit production has shifted towards the concepts of precision farming and the so-called "Intelligent Orchard". Precision farming describes the management of spatial and temporal variability of fields using Information, Computers and Technology (ICT). Spatial and temporal variability is analysed by investigating crop and soil properties. Based on this data, a field is divided into different management zones, which are used to apply appropriate inputs to achieve the best management / yield and increase profitability while reducing environmental impact (Gemtos et al., 2013). One key application in order to generate management zones is a technique called yield mapping. During yield mapping, fruits have been gathered in bins, which were weighted and tagged with additional GPS data. This data was then used to generate yield maps, which combined with soil data and historical data can be used to generate yield maps (Gemtos et al., 2013). However, this still involved a lot of manual labour, therefore, the concept of an intelligent orchard has been developed, which remains a very active research field today. In an intelligent orchard, many manual labor tasks are replaced by automatic systems, that monitor, manage and even harvest fruits by utilizing robots and unmanned aerial vehicle technology (Lyu et al., 2022), (Zhang et al., 2022). Especially, detecting, counting, tracking and classification of fruits in real time are essential to efficiently and reliably deploy these techniques. Recently, the application of computer vision technology has become a research hotspot to solve these different needs, such as automatic disease detection, fruit detection and classification (Lyu et al., 2022). Current research successfully deployed state-of-the-art computer vision models such as Yolo derivates and Convolutional neural networks in fruit orchards. Zheng et al. proposed YOLO BP, a modified Yolo V4 that includes a trimmed backbone to extract high-quality features and improve real-time performance in green citrus orchards (Zheng et al., 2021). Bargoti & Underwood adapted multiple Faster R-CNN architectures to achieve state-of-the-art performance through data augmentation and transfer learning techniques in apple, mango and almond fruit detection (Bargoti & Underwood, 2017). Koirala et al. compared two stage object detection techniques with single stage detectors for mango detection and based on their findings proposed a MangoYOLO

model (Koirala et al., 2019). Mirhaji et al. adapted YoloV4 for orange detection to provide data for precision farming applications (Mirhaji et al., 2021). All the models mentioned above achieve state-of-the-art performance with high F1-scores at around 0.95 and most of them can also be applied in real time on small edge AI computing plattforms such as the Nvidia Jetson NX. However, newer models such as Vision Transformers (VIT) have not yet gained traction in the domain of fruit detection, even though it has been shown that they can outperform traditional CNN based approaches in various computer vision tasks such as fruit quality evaluation, digital holography and image classification in terms of performance and robustness (Kumar & R, 2022)(Cuenat & Couturier, 2022) (Uparkar et al., 2023). Therefore, in this paper, current state-of-the-art vision transformers for object detection are adapted for fruit detection in orchards on the acfr-fruit-dataset (Bargoti & Underwood, 2017) and compared to various Faster-R-CNN and Yolo baselines.

Approach

First the characteristics of the acfr-fruit-dataset are analysed and the dataset is converted into an object detection standard COCO format. Afterwards, two Vision Transformers for object detection (Yolos and Detr) are fine-tuned on the train split of the dataset and are evaluated using the widely used COCO evaluator. The returned precision for the Intersection-over-Union in the $[0.5, 1]$-Intervall for a maximum of 100 Detections and the Recall for the Intersection-over-Union in the $[0.5, 0.95]$-Intervall are used to compute the F1-score. The F1-Score is the harmonic mean of precision and recall and therefore combines both metrics in a relatively durable metric against class imbalance but more importantly a comparable metric that involves both precision and recall (Korstanje, 2021). Both models are then compared against three different variations of Convolutional Neural Networks (CNN), that have been tested and trained by the authors of the dataset, ZFNet, VGG16 and a Pixel-wise CNN. In addition to raw performance, the real-time capabilities of both transformers are also evaluated by taking the average inference time over a batch of 250 samples from the COCO evaluator and comparing it against other state-of-the-art models such as the Yolo derivates mentioned earlier in terms of the processing time per image in milliseconds, the achievable frames per second (fps) and their number parameters.

The Dataset

The acfr-fruit-dataset has been created specifically to help researchers test and design reliable deep learning models for fruit detection in orchards by offering image data of three significantly orchard types across three different datasets (Bargoti & Underwood, 2017):

- Apples
- Mangoes
- Almonds

Each with their own dataset. The data has been captured by a general purpose research ground vehicle, which has been equipped with various image sensors, an external flashlight and a hand-held DSLR camera. The vehicle traversed through the orchard rows collecting full tree image data. The apple trees have been trellised in advance, to enable the ground vehicle to be in close proximity to the fruit and thus reducing the need for a high resolution sensor. For Mango and almond image collection, this was not possible, thus in order to capture mango fruits a higher resolution sensor with external strobe lighting was used with a small exposure time. For the almond trees, this approach was also infeasible, as almond trees have larger canopies and can host up to 10000 almonds per tree, which are smaller than apples and mangoes. To account for that, a hand-held DSLR camera was used to capture the images of the almond trees (Bargoti & Underwood, 2017). Imagery originally span entire trees to efficiently model the primary objective of yield mapping and estimation, however because of hardware constraints using the full resolution images is infeasible, to overcome these constraints, smaller sub-images were randomly sampled from the pool of larger images that have been acquired to build a representative training and testing objective (Bargoti & Underwood, 2017). The whole dataset has been manually annotated with rectangular bounding boxes (bbox) for almonds and mangoes and circular bboxes for apples, as it was deemed that they were more suitable for the characteristics of apples (Bargoti & Underwood, 2017).

The dataset comes in a .csv format: `#item,x,y,dx,dy,label` where each row contains an item identifier, the x and y coordinate along with the width and height of the bounding box and the corresponding label. For apples the same format is used, but x and y are the center coordinates and width and height are exchanged against the radius of the circle. Annotations are stored in a seperate folder from the images. First the dataset is read into memory by building a global dataset representation in form of a dictionary, that holds all three datasets, along with its data split references in one accessible variable.

In [1]:
```
import os.path

import pandas as pd
from os import listdir
from tqdm import tqdm
def build_annotation_dataframe(path: str) -> pd.DataFrame: # function to rea
    annotations = pd.DataFrame()
    for i, annotation in enumerate(tqdm(listdir(path))):
        part = pd.read_csv(os.path.join(path, annotation))
        part["picture_id"] = [annotation[:-4] for i in range(len(part))]
        part["picture_id"] = part["picture_id"].astype("string")
        if i == 0:
            annotations = part
            continue
        annotations = pd.concat([annotations,part])
    annotations.reset_index(inplace=True, drop=True)
    return annotations
```

In [2]:
```
# create global dataset representation
dataset = {
    "almonds": {
        "global_df": build_annotation_dataframe(path="datasets/acfr-fruit-da
        "train_ids": pd.read_csv("datasets/acfr-fruit-dataset/almonds/sets/t
        "test_ids": pd.read_csv("datasets/acfr-fruit-dataset/almonds/sets/te
```

```
        "val_ids": pd.read_csv("datasets/acfr-fruit-dataset/almonds/sets/val
    },
    "apples": {
        "global_df": build_annotation_dataframe(path="datasets/acfr-fruit-da
        "train_ids": pd.read_csv("datasets/acfr-fruit-dataset/apples/sets/tr
        "test_ids": pd.read_csv("datasets/acfr-fruit-dataset/apples/sets/tes
        "val_ids": pd.read_csv("datasets/acfr-fruit-dataset/apples/sets/val.
    },
    "mangoes": {
        "global_df": build_annotation_dataframe(path="datasets/acfr-fruit-da
        "train_ids": pd.read_csv("datasets/acfr-fruit-dataset/mangoes/sets/t
        "test_ids": pd.read_csv("datasets/acfr-fruit-dataset/mangoes/sets/te
        "val_ids": pd.read_csv("datasets/acfr-fruit-dataset/mangoes/sets/val
    }
}
```

```
100%|██████████| 620/620 [00:01<00:00, 355.94it/s]
100%|██████████| 1120/1120 [00:03<00:00, 366.27it/s]
100%|██████████| 1964/1964 [00:05<00:00, 352.79it/s]
```

After reading it into memory, the dataset can be analyzed using pandas and numpy respectively. The almond dataset, consists of a total of 620 images, with the apple set including 1120 and for mangoes there are 1964 images. However not all of them depict fruits. For almonds there are only 48 images without fruits, marking only a small portion of images without annotations. For apples and mangoes on the other hand 50% or more of the images in the dataset are annotationless / fruitless. This goes inline with Bargoti & Underwood explanation about fruit sparsity variations in different fruit types. Thus, this result is expected and approximates the true distribution of fruits in the tree canopies.

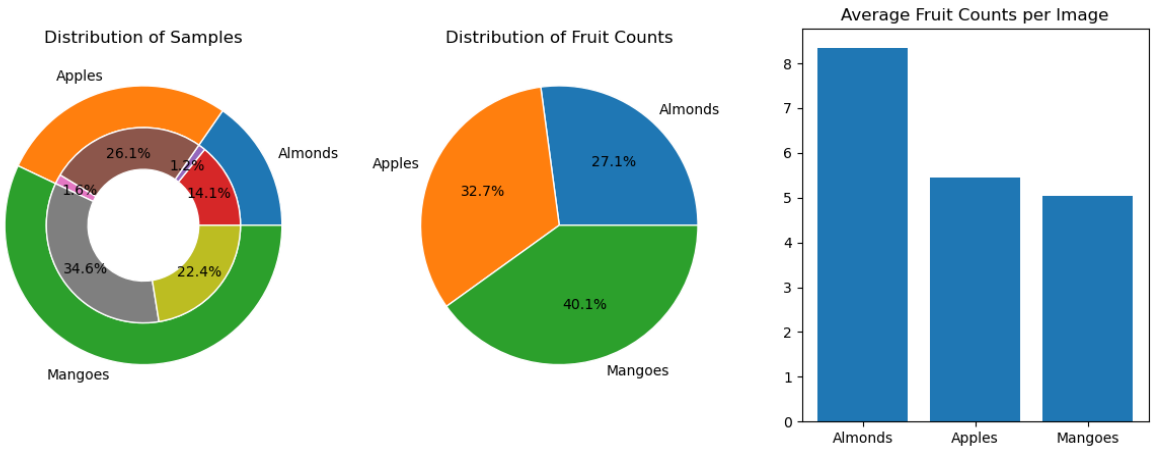
In [3]:
```
print(f" Pictures without Annotations for Almonds: {len(listdir('datasets/ac
print(f" Pictures without Annotations for Apples: {len(listdir('datasets/acf
print(f" Pictures without Annotations for Mangoes: {len(listdir('datasets/ac
```

```
 Pictures without Annotations for Almonds: 48, total number of pictures: 620
 Pictures without Annotations for Apples: 548, total number of pictures: 112
0
 Pictures without Annotations for Mangoes: 1392, total number of pictures: 1
964
```

In the following Figure, the distribution of Samples, Fruit counts and the average fruit count per image is depicted. On the left, the distribution of samples, visualises the above findings again, with respect to its relative counts. Here it gets even more clear, that the global dataset is imbalanced in terms of fruit samples with mangoes marking over 50% of the global amount samples. The inner ring of the pie chart displays the relative amount of unlabeled and labeled data, with the unlabeled data marked in light green, purple and pink respectively. Here it can be observed that for other than mangoes, unlabeled data only marks a small overall percentage of the fruit specific

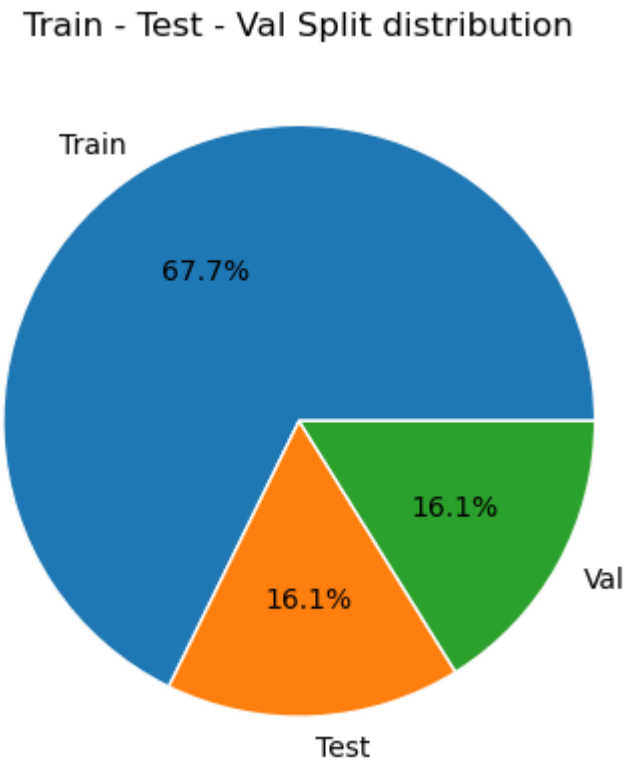
dataset. An explanation as to why the amount of imagery varies so drastically between fruit types has unfortunately not been given by the authors. However, taking the distribution of fruit counts (depicted in the middle), as well as the average fruit count per image (depicted on the right) into account, it can be hypothesized, that the authors wanted to achieve an equal representation of fruit samples across the three datasets. As the distribution of fruit counts is fairly even only varying by about 6.5%. This hypothesis is further underlined by observing the average fruit counts per image, as almonds have on average 8 fruits per image, whereas apples and mangoes have only around 4-5 fruits per image, with mangoes having the lowest fruit density.

In [10]:
```
import matplotlib.pyplot as plt
import numpy as np

# Plot distribution of samples, fruit counts and average count per image for
x_pictures = np.array([
    [len(dataset["almonds"]["global_df"].picture_id.unique()), len(listdir('
    [len(dataset["apples"]["global_df"].picture_id.unique()), len(listdir('d
    [len(dataset["mangoes"]["global_df"].picture_id.unique()), len(listdir('
])
x_fruits = [len(dataset["almonds"]["global_df"]), len(dataset["apples"]["glo
fruit_per_image = [dataset["almonds"]["global_df"].groupby(by="picture_id")[
labels = ["Almonds", "Apples", "Mangoes"]
size = 0.3
fig, ax = plt.subplots(1,3, figsize=(15,5))
ax[0].pie(x_pictures.sum(axis=1), radius=1, labels=labels, wedgeprops=dict(w
ax[0].pie(x_pictures.flatten(), radius=1-size, autopct='%1.1f%%',wedgeprops=
ax[0].set_title("Distribution of Samples")
ax[1].pie(x_fruits, labels=labels, autopct='%1.1f%%', wedgeprops=(dict(edgec
ax[1].set_title("Distribution of Fruit Counts")
ax[2].bar(x=labels, height=fruit_per_image)
ax[2].set_title("Average Fruit Counts per Image")
plt.show()
```



The dataset comes with predefined train, test and validation splits out of the box. The Sample distribution is depicted in the Figure below. As expected, the train set marks the biggest portion of the whole set with 67.7%, followed by the train and validation set respectively, which both use 16.1% of the data. In order to create these splits, Bargoti & Underwood sampled the data, such that each split contained data from different parts of the orchard, to account for different management zones which have different characteristics in order to minimise biased results. Images that do not contain fruits in the train set have been discarded. Because of these characteristics of the splits, they are used unchanged throughout this paper.

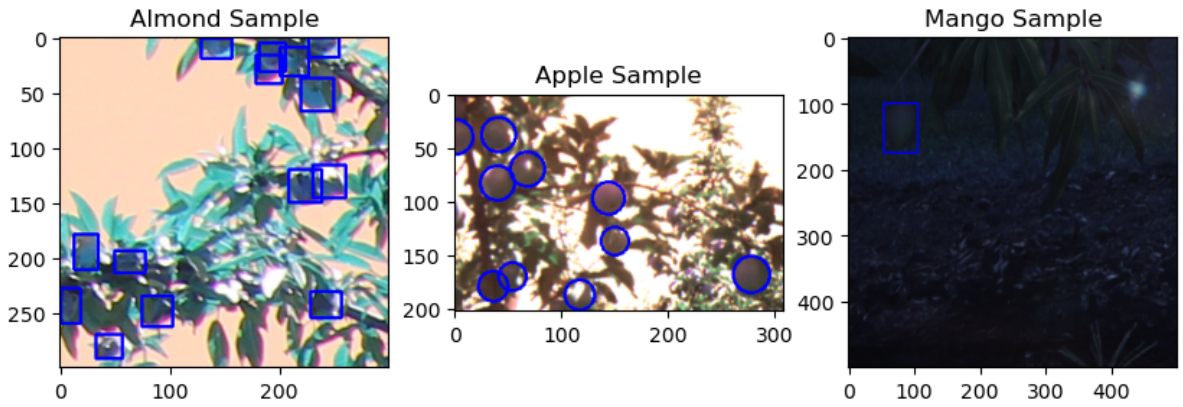
In [53]:
```
# Display distribution of train test and validation set
x = [len(dataset["almonds"]["train_ids"]), len(dataset["almonds"]["test_ids"
plt.pie(x, labels=["Train", "Test", "Val"], autopct="%1.1f%%", wedgeprops=di
plt.title("Train - Test - Val Split distribution")
plt.show()
```



Below random samples from the three different datasets are visualized, along with their annotations. This has been done to personally get familiar with the data. It can be observed, that the lighting conditions, image shape and overall image features are vastly different. The samples from the almond dataset have an orange background with cyan leaves and a lot of small very hard to detect fruits. An apple sample on the other hand is overblown pictures from the sun in the background and apples are clearly visible. Mangoes seem to be captured at night with only a few fruits on the and are thus overall of a darker nature with less contrast.

In [10]:
```
import cv2
import matplotlib.pyplot as plt

# randomly select a sample from each dataset along with its annotations to d
almond_sample = pd.Series(dataset["almonds"]["global_df"].picture_id.unique(
apple_sample = pd.Series(dataset["apples"]["global_df"].picture_id.unique())
```

```
mangoes_sample = pd.Series(dataset["mangoes"]["global_df"].picture_id.unique
almond_sample = dataset["almonds"]["global_df"].loc[dataset["almonds"]["glob
apple_sample = dataset["apples"]["global_df"].loc[dataset["apples"]["global_
mangoes_sample = dataset["mangoes"]["global_df"].loc[dataset["mangoes"]["glo

# draw ROI on random almond sample
almond_sample_img = cv2.imread(os.path.join("datasets/acfr-fruit-dataset/alm
almond_sample.reset_index(inplace=True, drop=True)
for idx, row in almond_sample.iterrows():
    cv2.rectangle(almond_sample_img, (int(row.x), int(row.y)), (int(row.x) +

# draw ROI on random apple sample
apple_sample_img = cv2.imread(os.path.join("datasets/acfr-fruit-dataset/appl
apple_sample.reset_index(inplace=True, drop=True)
for idx, row in apple_sample.iterrows():
    cv2.circle(apple_sample_img, (int(row["c-x"]), int(row["c-y"])), int(row

# draw ROI on random mango example
mangoes_sample_img = cv2.imread(os.path.join("datasets/acfr-fruit-dataset/ma
mangoes_sample.reset_index(inplace=True, drop=True)
for idx, row in mangoes_sample.iterrows():
    cv2.rectangle(mangoes_sample_img, (int(row.x), int(row.y)), (int(row.x)

# display image
fig, ax = plt.subplots(1,3, figsize=(10,8))
ax[0].imshow(almond_sample_img)
ax[0].set_title("Almond Sample")
ax[1].imshow(apple_sample_img)
ax[1].set_title("Apple Sample")
ax[2].imshow(mangoes_sample_img)
ax[2].set_title("Mango Sample")
plt.show()
```



In order to further analyse the characteristics of samples in the datasets, each picture has been serialized into a multidimensional numpy array, which represents the grid of pixel values and their width, height, overall pixel count, and pixel mean have been extracted and stored in a dataframe.

In [12]:
```
from tqdm import tqdm
import os
import pandas as pd
import cv2

# Build Dataframe for all 3 datasets that contains image metadata to analyse

apple_shapes = {
    "x": list(), # width
    "y": list(), # height
    "pixels": list(), # number of pixels
```

```
    "mean": list(), # average brightness of pixels across the image
    "picture_id": list() # file name
}
for picture_name in tqdm(os.listdir("datasets/acfr-fruit-dataset/apples/imag
    pic = cv2.imread(os.path.join("datasets/acfr-fruit-dataset/apples/images
    apple_shapes["x"].append(pic.shape[1])
    apple_shapes["y"].append(pic.shape[0])
    apple_shapes["pixels"].append(pic.shape[0] * pic.shape[1])
    apple_shapes["picture_id"].append(picture_name[:-4])
    apple_shapes["mean"].append(pic.mean())
apple_shapes = pd.DataFrame(apple_shapes)

mango_shapes = {
    "x": list(),
    "y": list(),
    "pixels": list(),
    "mean": list(),
    "picture_id": list()
}
for picture_name in tqdm(os.listdir("datasets/acfr-fruit-dataset/mangoes/ima
    pic = cv2.imread(os.path.join("datasets/acfr-fruit-dataset/mangoes/image
    mango_shapes["x"].append(pic.shape[1])
    mango_shapes["y"].append(pic.shape[0])
    mango_shapes["pixels"].append(pic.shape[0] * pic.shape[1])
    mango_shapes["picture_id"].append(picture_name[:-4])
    mango_shapes["mean"].append(pic.mean())
mango_shapes = pd.DataFrame(mango_shapes)

almond_shapes = {
    "x": list(),
    "y": list(),
    "pixels": list(),
    "mean": list(),
    "picture_id": list()
}
for picture_name in tqdm(os.listdir("datasets/acfr-fruit-dataset/almonds/ima
    pic = cv2.imread(os.path.join("datasets/acfr-fruit-dataset/almonds/image
    almond_shapes["x"].append(pic.shape[1])
    almond_shapes["y"].append(pic.shape[0])
    almond_shapes["pixels"].append(pic.shape[0] * pic.shape[1])
    almond_shapes["picture_id"].append(picture_name[:-4])
    almond_shapes["mean"].append(pic.mean())
almond_shapes = pd.DataFrame(almond_shapes)

apple_shapes
```

```
100%|████████████████████████████████████████████████████████████
████████████████████| 1120/1120 [00:03<00:00, 283.87it/s]
100%|████████████████████████████████████████████████████████████
████████████████████| 1964/1964 [00:36<00:00, 53.55it/s]
100%|████████████████████████████████████████████████████████████
████████████████████| 620/620 [00:02<00:00, 238.40it/s]
```

Out[12]:

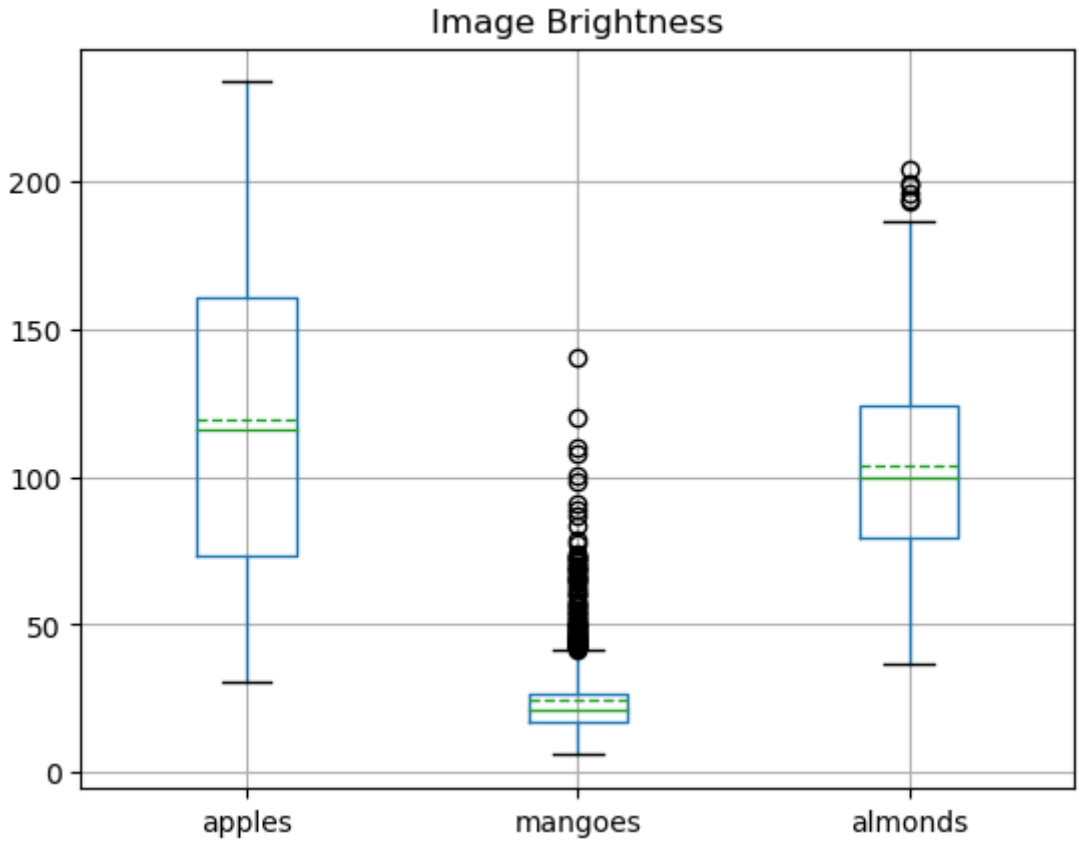
	x	y	pixels	mean	picture_id
0	308	202	62216	97.444896	20130320T004615.614293.Cam6_51
1	308	202	62216	184.350623	20130320T013657.200420_21
2	308	202	62216	97.161363	20130320T013024.621036_62
3	308	202	62216	139.973153	20130320T004807.235605.Cam6_22
4	308	202	62216	212.991605	20130320T005104.572657.Cam6_14
...	...	...	...	...	...
1115	308	202	62216	166.783914	20130320T005423.814872.Cam6_12
1116	308	202	62216	112.020059	20130320T013321.386510_31
1117	308	202	62216	52.170096	20130320T005816.391049.Cam6_51
1118	308	202	62216	118.021908	20130320T005239.812601.Cam6_34
1119	308	202	62216	59.769989	20130320T004918.284713.Cam6_62

1120 rows × 5 columns

First up the distribution of the average image brightness has been analyzed. As a representative value for image brightness, the average brightness of each pixel across the image is used. In the box plot below, it can be observed that the apple dataset almost spans across the whole brightness spectrum with the median and mean both at around 120, suggesting, that it includes images with good and poor lighting conditions as well as day and night imagery. The mango datasets images on the other hand are extremely dark with the mean and median at around 25, with the maximum whisker of the plot also being below 50, however, this was calculated with a maximum constraint of $1.5\mathrm{IQR},\ \text{with}\ \mathrm{IQR} = \mathrm{Q3} - \mathrm{Q1}$ and the rest are marked as outliers and depicted with circles. The brightness in the almond dataset is also fairly balanced with dark and bright images.

In [60]:
```
# Display Brightness distribution data for all 3 datasets
means = pd.DataFrame()
means["apples"] = apple_shapes["mean"]
means["mangoes"] = mango_shapes["mean"]
means["almonds"] = almond_shapes["mean"]
means.boxplot(showmeans=True, meanline=True)
plt.title("Image Brightness")
```

Out[60]:
```
Text(0.5, 1.0, 'Image Brightness')
```



In addition to image brightness, the variance between the overall shape of the image has also been determined, to account for different aspect ratios and resolutions between images in each dataset. However, as expected because the data has been captured by the same sensor and entity for each dataset, there is no variance between images.

In [58]:
```
# calculate variance across the image shapes
variances = pd.DataFrame()
variances["apple_shapes"] = apple_shapes.var(numeric_only=True)
variances["almond_shapes"] = almond_shapes.var(numeric_only=True)
variances["mango_shapes"] = mango_shapes.var(numeric_only=True)
variances
```

Out[58]:

	apple_shapes	almond_shapes	mango_shapes
x	0.000000	0.000000	0.000000
y	0.000000	0.000000	0.000000
pixels	0.000000	0.000000	0.000000
mean	2461.866215	1128.139201	198.633097

In [59]:
```
mango_shapes.x.unique(), mango_shapes.y.unique(), mango_shapes.pixels.unique
```

Out[59]:
```
(array([500]), array([500]), array([250000]))
```

Converting to Coco Format

As the dataset comes in no particular object detection standard format, it is converted to COCO format in this section. This enables the use of the popular COCO evaluator for object detection and allows for easier data loading and use, as many libraries such as pytorch have predefined handlers for the COCO dataset or dataset in the COCO format.

The COCO format provides that, the annotation data is stored in one large JSON-file for each split of the dataset. The JSON file looks as follows:

```
{
    "info": {
        "year": "2021",
        "version": "1.0",
        "description": "Exported from FiftyOne",
        "contributor": "Voxel51",
        "url": "https://fiftyone.ai",
        "date_created": "2021-01-19T09:48:27"
    },
    "licenses": [
        {
          "url": "http://creativecommons.org/licenses/by-nc-
sa/2.0/",
          "id": 1,
          "name": "Attribution-NonCommercial-ShareAlike License"
        },
        ...
    ],
    "categories": [
        ...
        {
            "id": 2,
            "name": "cat",
            "supercategory": "animal"
        },
        ...
    ],
    "images": [
        {
            "id": 0,
            "license": 1,
            "file_name": "<filename0>.<ext>",
            "height": 480,
            "width": 640,
            "date_captured": null
        },
        ...
    ],
    "annotations": [
        {
            "id": 0,
            "image_id": 0,
            "category_id": 2,
            "bbox": [260, 177, 231, 199],
            "segmentation": [...],
            "area": 45969,
            "iscrowd": 0
        },
        ...
    ]
}
```

It includes three main data objects:

- the categories, which include all information about the labels in the dataset
- the images, which includes information about the image sample, such as height, width, the file name etc.
- the annotations, which link the category / label to a particular image and include the coordinates of the bounding box (bbox), the area of the bounding box etc.

Thus, first the categories for the three datasets are defined. As each dataset only includes one label, each category has the id 0 with the corresponding name of the fruit as the label.

In [11]:
```
# Set categories for image labeling
categories_almond = [{
    "id": 0,
    "name": "almond",
    "supercategory": "fruit"
}]
categories_apple = [{
    "id": 0,
    "name": "apple",
    "supercategory": "fruit"
}]
categories_mangoes = [{
    "id": 0,
    "name": "mango",
    "supercategory": "fruit"
}]
```

Now, the image shapes and annotations can be converted into COCO format, by renaming the columns of the original dataframes to match the keys in the JSON of the COCO Format and exporting the dataframe into a dictionary with the record orientation. This results in one large list, with each record as a dictionary with the column names as keys. As python dictionaries are essentially an internal representation equivalent to JSON, this export can directly be attached to the COCO annotation dictionary, that is then serialized as a JSON object later on.

In [13]:
```
from typing import Dict, List, Union
def image_shapes_to_coco(shapes: pd.DataFrame) -> List[Dict[str, Union[int,s
        shapes = shapes.copy() # Create Deep copy to prevent reference chang
        shapes["id"] = shapes.index
        shapes.rename(columns={
            "x": "width",
            "y": "height",
            "picture_id": "file_name"
        }, inplace=True) # rename columns to fit COCO Format
        shapes["license"] = None # Add additional columns
        shapes["date_captured"] = None
        shapes.file_name = shapes.file_name.astype(str)
        shapes.file_name = shapes.file_name.str.cat([".png" for _ in range(l
        shapes.drop(columns=["mean", "pixels"], inplace=True) # remove unnec
        return shapes.to_dict(orient="record") # export as dict which can be
```

The same procedure applies for the annotations, with the exception, that the index of the annotation and image (shapes) dataframes have to be linked together through an image_id and id column. In addition to that, the category id has to be set (to 0) for each annotation to reference the correct category / label and the bounding box coordinates have to be converted into rectangles for apples and then combined into a list instead of

seperate columns for all three datasets. Last but not least, the area of the box is computed, unnecessary columns removed and the dataframe is exported in the same way as for the images.

In [15]:
```
import swifter
def annotations_to_coco(annotations: pd.DataFrame, shapes: pd.DataFrame, app
    annotations = annotations.copy() # Create Deep copy to prevent reference
    shapes = shapes.copy()
    shapes["image_id"] = shapes.index # Create foreign keys to reference dat
    annotations["id"] = annotations.index
    annotations["picture_id"] = annotations.picture_id.astype(str)
    annotations = pd.merge(left=annotations, right=shapes[["picture_id", "im
    annotations["category_id"] = [0 for _ in range(len(annotations))] # Set
    if apples: # account for circular annotations for apples and convert the
        annotations["bbox"] = annotations.swifter.apply(lambda row: [int(row
        annotations["area"] = 2*annotations.radius * 2*annotations.radius
        annotations.drop(columns=["picture_id", "#item", "label", "c-x", "c-
    else:
        annotations["bbox"] = annotations.swifter.apply(lambda row: [int(row
        annotations["area"] = annotations.dx * annotations.dy
        annotations.drop(columns=["picture_id", "#item", "label", "x", "y",
    annotations["iscrowd"] = 0
    annotations["iscrowd"] = 0
    return annotations.to_dict(orient="record")
```

Now for each dataset, annotations in COCO Format can be build using the above defined functions.

In [17]:
```
# Build COCO Format Dataset Representations
apple_coco_train = {
    "categories": categories_apple,
    "images": image_shapes_to_coco(apple_shapes.loc[apple_shapes.picture_id.
    "annotations": annotations_to_coco(annotations=dataset["apples"]["global
    "licenses": None,
    "info": None
}
apple_coco_test = {
    "categories": categories_apple,
    "images": image_shapes_to_coco(apple_shapes.loc[apple_shapes.picture_id.
    "annotations": annotations_to_coco(annotations=dataset["apples"]["global
    "licenses": None,
    "info": None
}
apple_coco_val = {
    "categories": categories_apple,
    "images": image_shapes_to_coco(apple_shapes.loc[apple_shapes.picture_id.
    "annotations": annotations_to_coco(annotations=dataset["apples"]["global
    "licenses": None,
    "info": None
}
almonds_coco_train = {
    "categories": categories_almond,
    "images": image_shapes_to_coco(almond_shapes.loc[almond_shapes.picture_i
    "annotations": annotations_to_coco(annotations=dataset["almonds"]["globa
    "licenses": None,
    "info": None
}
almonds_coco_test = {
    "categories": categories_almond,
    "images": image_shapes_to_coco(almond_shapes.loc[almond_shapes.picture_i
    "annotations": annotations_to_coco(annotations=dataset["almonds"]["globa
```

```
    "licenses": None,
    "info": None
}
almonds_coco_val = {
    "categories": categories_almond,
    "images": image_shapes_to_coco(almond_shapes.loc[almond_shapes.picture_i
    "annotations": annotations_to_coco(annotations=dataset["almonds"]["globa
    "licenses": None,
    "info": None
}
mangoes_coco_train = {
    "categories": categories_mangoes,
    "images": image_shapes_to_coco(mango_shapes.loc[mango_shapes.picture_id.
    "annotations": annotations_to_coco(annotations=dataset["mangoes"]["globa
    "licenses": None,
    "info": None
}
mangoes_coco_test = {
    "categories": categories_mangoes,
    "images": image_shapes_to_coco(mango_shapes.loc[mango_shapes.picture_id.
    "annotations": annotations_to_coco(annotations=dataset["mangoes"]["globa
    "licenses": None,
    "info": None
}
mangoes_coco_val = {
    "categories": categories_mangoes,
    "images": image_shapes_to_coco(mango_shapes.loc[mango_shapes.picture_id.
    "annotations": annotations_to_coco(annotations=dataset["mangoes"]["globa
    "licenses": None,
    "info": None
}
```

```
Pandas Apply:   0%|          | 0/4673 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/554 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/538 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/3185 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/797 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/795 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/5186 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/947 [00:00<?, ?it/s]
Pandas Apply:   0%|          | 0/932 [00:00<?, ?it/s]
```

However, only converting annotations is not enough. The dataset also has to be in a specified directory structure. This structure provides, that each split has its own directory, with the annotation JSON for each split on the toplevel of the directory. To achieve this, the following function creates a "coco" directory in the folder of the given dataset, creates the split subfolders, copies all required images into their corresponding folders and exports the annotation dictionaries as JSON and writes them to disk at the required location.

In [18]:
```
from shutil import copy
import json

def create_coco_dir_struct(label_json_train: dict, label_json_test: dict, la
    # automatically create dirs if not exist
    os.makedirs(name=f"datasets/acfr-fruit-dataset/{fruit}/coco/train", exis
    os.makedirs(name=f"datasets/acfr-fruit-dataset/{fruit}/coco/test", exist
    os.makedirs(name=f"datasets/acfr-fruit-dataset/{fruit}/coco/val", exist_

    for image_struct in label_json_train["images"]:
        copy(src=f"datasets/acfr-fruit-dataset/{fruit}/images/{image_struct[
    for image_struct in label_json_test["images"]:
```

```
        copy(src=f"datasets/acfr-fruit-dataset/{fruit}/images/{image_struct[
    for image_struct in label_json_val["images"]:
        copy(src=f"datasets/acfr-fruit-dataset/{fruit}/images/{image_struct[

    open(f"datasets/acfr-fruit-dataset/{fruit}/coco/train.json", "w+").write
    open(f"datasets/acfr-fruit-dataset/{fruit}/coco/test.json", "w+").write(
    open(f"datasets/acfr-fruit-dataset/{fruit}/coco/val.json", "w+").write(j
```

In [19]:
```
label_jsons = {
    "almonds": [almonds_coco_train, almonds_coco_val, almonds_coco_test],
    "apples": [apple_coco_train, apple_coco_val, apple_coco_test],
    "mangoes": [mangoes_coco_train, mangoes_coco_val, mangoes_coco_test],
}
# Create Directories
for fruit, labels in label_jsons.items():
    create_coco_dir_struct(label_json_train=labels[0], label_json_test=label
```

Now the dataset is available in COCO format and the suite of available COCO tools can be used for a comparable evaluation and training process between different papers and publications.

Pytorch Dataloader & Dataset

Before the dataset can be used to train a model or to run inference, a pytorch dataset has to be defined for it. Thus, the included CocoDetection dataset class of torchvision, which is a module of pytorch, is extended, such that it returns each image and the corresponding annotation in a format, a vision transformer can understand. The primary extension that has to be made to the standard CocoDetection dataset class, is that the dataset has to account for a custom model specific feature extractor / image processor, which is in charge of preparing the input features for its corresponding vision models (Image Processor, n.d.). This includes resizing and normalization of the input image, as well as padding the image to fit the models input scheme. Thus the `__getitem__` method is overwritten to call the feature extractor before returning the `pixel_values` and the target for the current data item.

In [3]:
```
import torchvision

# define Torch Dataset in COCO Format
class CocoDetection(torchvision.datasets.CocoDetection):

    def __init__(self, img_folder, feature_extractor, train=True):
        annotations_path = os.path.join(img_folder, "../train.json" if train
        super(CocoDetection, self).__init__(img_folder, annotations_path)
        self.feature_extractor = feature_extractor

    def __getitem__(self, idx):
        # read image and target
        img, target = super(CocoDetection, self).__getitem__(idx)

        # preprocess image and target (resizing + normalization), such that
        image_id = self.ids[idx]
        target = {"image_id": image_id, 'annotations': target}
        encoding = self.feature_extractor(images=img, annotations=target, re
        pixel_values = encoding["pixel_values"].squeeze() # remove batch dim
        target = encoding["labels"][0]
```

```
        return pixel_values, target
```

Models

In this paper two Vision Transformers for Object Detection, namely Yolos and Detr will be Fine-Tuned on the acf-fruit-dataset. Their performance is later evaluated against multiple state-of-the-art Models for applications in orchards. To directly asses detection performance, the Faster-R-CNN ZFnet, VGG16 and a pixel-wise CNN adapted by the authors of the dataset and their reported results are used to form a comparison baseline. As a second evaluation step, the real-time performance of both vision transformers is discussed by comparing them to not only the inference times of the mentioned Faster-R-CNN derivatives but also taking multiple state-of-the-art YOLO models into account, that have been developed to perform well in green citrus, orange or dragon fruit orchards. As this comparison focuses on the average inference time per image, the actual dataset objective is not relevant for this comparison. Thus, in the following chapter, Faster-R-CNN, YOLO and Vision Transformers along with their most important derivatives are introduced and YOLOS and Detr are also Fine-Tuned.

Faster-R-CNN

The Faster-R-CNN network consists of two-stage detection process. First, a Region Proposal Network (RPN) predicts Regions of Interest (RoI) along with an objectness score. These predictions are then passed on to an image classifier, which classifies the objects in the proposed RoI (Ren et al., 2016). The Region Proposal Network is a convolutional neural network, that shares a set of convolutional layers with the image classifier. It works by sliding a small network over the convolutional feature map output by the last shared convolutional layer. Each sliding window is then mapped to a lower-dimensional feature, which is fed into two sibling fully connected layers (Ren et al., 2016). One layer is a box regression layer, that predicts the coordinates of a (rectangular) bounding box and the other is a box-classification layer, which predicts an object score for each available label. At each sliding window location multiple region proposals are predicted (Ren et al., 2016). The extracted RoIs are then fed into another Object Detection / Image Classification network such as Fast-R-CNN which poses an end-to-end Convolutional Neural Network image classifier, that takes RoIs from a region proposal module (Ren et al., 2016). Thus Fast-R-CNNs performance is directly dependent on how well the region proposal network performs.

ZFnet and VGG16

In terms of Faster-R-CNN, ZFnet and VGG16 refer to the derivatives of Fast-R-CNN, that are used in the second stage of the Faster-R-CNN model. Both are fully convolutional neural networks of a different size. VGG16 is a 16 layer deep network, that can perform classification tasks on 224x224 pixel RGB images for up to 1000 classes with high level accuracy (Simonyan & Zisserman, 2015). ZFnet is a smaller convolutional neural network, consisting of 5 convolutional layers which originally aimed at visualising how convolutional networks work and what features they extracted (Zeiler & Fergus, 2013), however it can also be used as a simple image classifier.

YOLO

YOLO, or You Only Look Once, is an object detection network, that aims to achieve unified, real-time object detection through unifying the separate object detection steps into one single neural network (Redmon et al., 2016). It achieves this, by modelling the detection task as a regression problem. It first divides the image into a SxS grid. Then it uses each grid cell to predict bounding boxes along with confidence scores that describe how confident the model is that a box contains an object. Simultaneously, it predicts a class probability map, where class probabilities are predicted for each grid cell. The probabilities of the classes are then multiplied with the confidence scores of each bounding box to retrieve class-specific bounding box confidence scores that encode both the probability of the class, and how well the predicted box fits the object. The grid cell at the center of each bounding box is responsible for detecting the object. In reality, this is achieved by using 24 convolutional layers, followed by two fully connected layers (Redmon et al., 2016). To this day, YOLO models and their revisions achieve state-of-the-art realtime object detection performance (Koirala et al., 2019; Zhang et al., 2022).

Vision Transformer

Vision Transformer aim to bring the dominant and outstanding performance of Transformer models from the domain of Natural Language Processing to computer vision tasks (Dosovitskiy et al., 2021). Primarily this includes advantages such as:

- Network pre-training with efficient Fine-Tuning on downstream tasks
- Computational efficiency
- scalability
- easy adaptability

It aims at re-using the original Transformer architecture proposed by Vaswani et al. as much as possible, which takes a 1D sequence of token embeddings as input (Vaswani et al., 2017). The input is then processed by an Encoder, which consists of multiple layers comprised of (Vaswani et al., 2017):

- a Multi-Head Attention module, which runs multiple attention layers in parallel to be able to jointly attend to information accross different representation subspaces and positions,
- residual connections to retain important information, such as positional data
- a Feed Forward network, which acts the same as two convolutions with kernel size 1

The resulting representation is then used by a Decoder to auto-regressively generate a text sequence. However, the architecture of the decoder is not outlined, as it is not needed in Vision Transformers.

As this architecture is designed to work with sequential inputs, it can't work with images out of the box. Therefore, Dosovitskiy et al. proposed a method to map an image to a sequential representation. Each image $x \in \mathbb{R}^{H \times W \times C}$ is reshaped into a sequence of flattened 2D-patches $x_p \in \mathbb{R}^{N \times (P^2 * C}$, where H marks the image height, W the image width, C is the number of channels, P being the width and height of the patches and

$N = \frac{HW}{P^2}$ is the number of patches, which simultaneously marks the effective input sequence length to the transformer encoder (Dosovitskiy et al., 2021). However, as the transformer uses a fixed hidden size d as its latent representation, thus each flattened patch has to be mapped to d dimensions, which is achieved with a trainable linear projection. The output of this projection is referred to as the patch embeddings (Dosovitskiy et al., 2021). A learned positional embedding is then added to the patch embeddings similar to the positional embedding of the original transformer model, to form the complete model input. In addition, the transformer encoder is also extended to include a prepended classification token similar to what is used in the BERT Transformer, whose state at the output of the encoder serves as the image representation (Dosovitskiy et al., 2021). A simple fully connected classification head can then be attached to this classification token to take care of image classification. As an alternative to the raw image patch input, Dosovitskiy et al. also proposed a hybrid architecture that builds the model input from CNN feature maps, by applying the patch embedding projection to patches from the extracted CNN feature map directly (Dosovitskiy et al., 2021). The classification input embedding, and the position embeddings are added as described above as well. As an alternative to the raw image patch input, Dosovitskiy et al. also proposed a hybrid architecture that builds the model input from CNN feature maps, by applying the patch embedding projection to patches from the extracted CNN feature map directly (Dosovitskiy et al., 2021). The classification input embedding, and the position embeddings are added as described above as well.

The model showed to have similar image classification performance, while being scalable to different sizes and taking significantly less ressources to train then the Faster-R-CNN baseline (Dosovitskiy et al., 2021). However, the standard model can only be used for image classification and not object detection, rendering the original model unfeasable for the task presented in this paper. Therefore, two dervatives of the original Vision Transformer are used, YOLOS and DETR.

DETR

DETR is a End-to-End Object Detection Transformer, that aims at performing direct set predictions instead of solving the task indirectly through region proposals and classification tasks, all while being easy to implement and ressource efficient (Carion et al., 2020). The model consists of three main parts:

1. A conventional CNN backbone
2. A Transformer encoder-decoder architecture
3. A prediction Forward Neural Network

The input image is first passed through the conventional Convolutional Neural Network, which generates a lower-resolution activation map $f \in \mathbb{R}^{C \times H \times W}$, typically with $C = 2048$ and $H, W = \frac{H_0}{32}, \frac{W_0}{32}$. The number of channels C is then further reduced by another 1×1 convolution to a smaller dimension d, which matches the fixed hidden size d of the transformer encoder-decoder (Carion et al., 2020). As in the original Vision Transformer, the feature map is now mapped to a sequential input. DETR achieves this,

by collapsing the spatial dimensions of the feature map into one dimension, resulting in a $d \times HW$ sequential feature map. Fixed positional encodings are then added to the feature map, and it is passed on to the Transformer encoder (Carion et al., 2020). The transformer encoder itself remains unchanged compared to the original one proposed by (Vaswani et al., 2017). The Transformer decoder architecture also remains unchanged compared to the original, however, instead of using an autoregressive approach, DETR decodes N objects in parallel at each decoder layer. It achieves this, by feeding N learnt positional encodings, that Carion et al. refer to as *object queries*, into the Transformer decoder and adding them to each attention layer. As the decoder is permutation-invariant, all N object queries have to be different to generate different outputs (Carion et al., 2020). The object queries are then transformed into an output embedding by the decoder, while attending to the encoder output in the encoder-decoder attention layer of the transformer decoder. The output embedding is then passed on to a Forward Neural Network, which independently predicts box coordinates and class labels, resulting in N, with $N > \text{number\_of\_classes}$, final predictions. To account for positions where no class or object might be present, an additional special class label is added to the available classes that represents that no object is detected within a slot (Carion et al., 2020).

In order to Fine-Tune DETR on the acfr-fruit-dataset, a custom collate function has to be defined, that is passed to the torch Dataloader. This function converts each batch into the data structure, the model expects as input. For Detr, this includes padding the image and generating a pixel mask. The pixel mask is used if segmentation heads instead of detection heads are used.

In [7]:
```
def collate_fn(batch): # Custom collate function to convert batch to Detr Fo
  pixel_values = [item[0] for item in batch]
  encoding = feature_extractor.pad(pixel_values, return_tensors="pt") # pad
  labels = [item[1] for item in batch]
  batch = {}
  batch['pixel_values'] = encoding['pixel_values']
  batch['pixel_mask'] = encoding['pixel_mask']
  batch['labels'] = labels
  return batch
```

Then the Detr model is defined as a LightningModule by creating a Wrapper class extending pytorch lightnings LightningModule and mapping the Detr implementation of Huggingface to the functions of the LightningModule. This allows the training using pytorch lightning instead of regular pytorch, wich adds the benefits of automatic logging to Tensorboard and the usage of a unified Trainer. This eliminates the necessity of defining a custom validation and training loop. Instead, a common step is defined, which handles the prediction and loss computation for both the validation and training loops and a validation and training step is defined, which return the losses to the Trainer and define the Tensorboard logging topics for the variables of interest such as the current loss values. Last but not least, the optimizer to be used is defined, as proposed by the authors of Detr (Carion et al., 2020), the torchs AdamW optimizer is used which extends the popular Adam optimizer with weight decay functionality.

In [8]:
```
from transformers import DetrForObjectDetection
import pytorch_lightning as pl
```

```
import torch

# define Detr VIT as a Lightning Module, so it can be trained with pytorch l
class Detr(pl.LightningModule):
     def __init__(self, lr, lr_backbone, weight_decay):
         super().__init__()
         # replace COCO classification head with custom head
         # we specify the "no_timm" variant here to not rely on the timm lib
         # for the convolutional backbone
         self.model = DetrForObjectDetection.from_pretrained("facebook/detr-
                                                             revision="no_ti
                                                             num_labels=1,
                                                             ignore_mismatch

         self.lr = lr
         self.lr_backbone = lr_backbone
         self.weight_decay = weight_decay

     def forward(self, pixel_values, pixel_mask):
       outputs = self.model(pixel_values=pixel_values, pixel_mask=pixel_mask

       return outputs

     def common_step(self, batch, batch_idx):
       pixel_values = batch["pixel_values"]
       pixel_mask = batch["pixel_mask"]
       labels = [{k: v.to(self.device) for k, v in t.items()} for t in batch

       outputs = self.model(pixel_values=pixel_values, pixel_mask=pixel_mask

       loss = outputs.loss
       loss_dict = outputs.loss_dict

       return loss, loss_dict

     def training_step(self, batch, batch_idx):
        loss, loss_dict = self.common_step(batch, batch_idx)
        # logs metrics for each training_step,
        # and the average across the epoch
        self.log("training_loss", loss)
        for k,v in loss_dict.items():
          self.log("train_" + k, v.item())

        return loss

     def validation_step(self, batch, batch_idx):
        loss, loss_dict = self.common_step(batch, batch_idx)
        self.log("validation_loss", loss)
        for k,v in loss_dict.items():
          self.log("validation_" + k, v.item())

        return loss

     def configure_optimizers(self):
        param_dicts = [
              {"params": [p for n, p in self.named_parameters() if "backbone
              {
                  "params": [p for n, p in self.named_parameters() if "backb
                  "lr": self.lr_backbone,
              },
        ] # Configure Parameters for the CNN Backbone of Detr
        optimizer = torch.optim.AdamW(param_dicts, lr=self.lr,
                                      weight_decay=self.weight_decay) # initiali
```

```
        return optimizer

    def train_dataloader(self):
        return train_dataloader

    def val_dataloader(self):
        return val_dataloader
```

Apples

Now, the only thing that remains is to load Detr image processor / feature extractor, initialise the datasets for the current task, in this case for the apple subset.

In [49]:
```
from transformers import DetrImageProcessor
from torch.utils.data import DataLoader

# init dataset and feature extractor
feature_extractor = DetrImageProcessor.from_pretrained("facebook/detr-resnet
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples/c
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
batch = next(iter(train_dataloader))
batch.keys() # check batch keys
```

```
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
```

Out[49]:
```
dict_keys(['pixel_values', 'pixel_mask', 'labels'])
```

Initialize the model and call the `fit()` method of the pytorch lightning Trainer, which will automatically start model training. The model is initialized from a pre-trained DETR checkpoint on the COCO 2017 object detection dataset. All training hyperparameters remain unchanged to the original proposal of Carion et al. and no Hyperparameter tuning is done, except for increasing the maximum number of optimisation steps to 1000, as training with the original proposition of 300 showed significant underfitting behavior. The model is thus trained on an Nvidia Tesla T4 GPU for 1000 optimization steps, which takes approximately 30mins in total.

In [51]:
```
from pytorch_lightning import Trainer
# Train Detr
model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)

max_steps = 1000

trainer = Trainer(max_steps=max_steps, gradient_clip_val=0.1)
trainer.fit(model)
```

```
Some weights of DetrForObjectDetection were not initialized from the model c
heckpoint at facebook/detr-resnet-50 and are newly initialized because the s
hapes did not match:
- class_labels_classifier.weight: found shape torch.Size([92, 256]) in the c
heckpoint and torch.Size([2, 256]) in the model instantiated
- class_labels_classifier.bias: found shape torch.Size([92]) in the checkpoi
nt and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                    | Params
-------------------------------------------------
0 | model | DetrForObjectDetection | 41.5 M
-------------------------------------------------
18.0 M    Trainable params
23.5 M    Non-trainable params
41.5 M    Total params
166.007   Total estimated model params size (MB)
Sanity Checking: 0it [00:00, ?it/s]
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
`Trainer.fit` stopped: `max_steps=1000` reached.
```

In [52]:
```
torch.save(model.state_dict(), "models/detr_apples_v1.pt") # save model for
```

After training has completed, the model is saved for reuse and reproducability and the models predictions on a random sample along with the corresponding ground truth is visualised. The same procedure is repeated for Almonds and Mangoes without significant changes.

In [58]:
```
from transformers import AutoImageProcessor, AutoModelForObjectDetection
import torch
from PIL import Image
import requests
import numpy as np
import cv2

# Visualise prediction and ground truth on a random sample

val_id = np.random.randint(0, len(val_dataset))

file_name = val_dataset.coco.loadImgs(
                                    val_dataset[val_id][1]["image_id"].item(
```

```
image = Image.open(os.path.join("datasets/acfr-fruit-dataset/apples/coco/val
                                file_name
                    ))
image_processor = AutoImageProcessor.from_pretrained("facebook/detr-resnet-5

cats = train_dataset.coco.cats
id2label = {k: v['name'] for k,v in cats.items()}

val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
pixel_values, pixel_mask, labels = list(iter(val_dataloader))[val_id].values

outputs = model(pixel_values=pixel_values, pixel_mask=pixel_mask)


# convert outputs (bounding boxes and class logits) to COCO API
target_sizes = torch.tensor([image.size[::-1]])
results = image_processor.post_process_object_detection(outputs, threshold=0
    0
]

image = np.array(image)
image = image[:,:,::-1].copy()
for score, label, box in zip(results["scores"], results["labels"], results["
    box = [round(i, 2) for i in box.tolist()]
    print(
        f"Detected {id2label[label.item()]} with confidence "
        f"{round(score.item(), 3)} at location {box}"
    )
    cv2.rectangle(image, (int(box[0]), int(box[1])), (int(box[2]), int(box[

for idx, row in dataset["apples"]["global_df"].loc[dataset["apples"]["global
    cv2.circle(image, (int(row["c-x"]), int(row["c-y"])), int(row.radius), c

plt.imshow(image)
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
Could not find image processor class in the image processor config or the mo
del config. Loading based on pattern matching with the model's feature extra
ctor configuration.
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:780: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
```

```
DetrObjectDetectionOutput(loss=None, loss_dict=None, logits=tensor([[[-1.502
3, -0.0542],
         [-0.5519, -0.8863],
         [-2.7177,  0.5763],
         [-1.3145,  0.4309],
         [-0.7623,  0.0162],
         [-1.9191,  0.4033],
         [-1.7564,  0.3799],
         [-0.6828, -1.2782],
         [-0.4672, -0.2590],
         [ 0.6040, -1.4746],
         [-3.3978,  1.6881],
         [-2.3168,  0.3080],
         [-3.2332,  1.8499],
         [-3.4193,  1.6290],
         [-3.0391,  1.8341],
         [-0.8370, -0.4017],
         [-0.4624, -0.4625],
         [-1.8729,  1.3773],
         [-1.9056,  1.0051],
         [-2.5642,  1.2216],
         [-4.2879,  2.7057],
         [-3.4813,  2.0125],
         [-1.4913,  0.6702],
         [-4.4261,  2.6772],
         [ 1.2614, -2.3505],
         [-2.4350,  0.9955],
         [-1.2127,  0.0705],
         [-2.0040, -0.0502],
         [ 0.5708, -1.1972],
         [ 0.6787, -1.0093],
         [-2.2410,  0.5560],
         [-1.0937, -0.3211],
         [-3.4467,  1.2576],
         [-2.2593,  1.1313],
         [-3.1328,  1.3583],
         [-0.8018, -0.4817],
         [-1.9907,  0.9546],
         [-2.3029,  0.6148],
         [-1.9808,  0.0913],
         [-1.5935,  1.3507],
         [ 1.0224, -1.4647],
         [-2.2685,  1.1154],
         [-0.1704, -0.8346],
         [-2.5601,  0.7985],
         [ 0.6171, -1.2422],
         [-1.2920, -0.0607],
         [-1.8392,  1.0068],
         [-2.6355,  1.0235],
         [-2.1890,  0.5842],
         [-1.0643, -0.5295],
         [ 1.5669, -2.1673],
         [ 0.7481, -0.8608],
         [-3.2716,  1.0175],
         [-1.1488, -0.0208],
         [-2.3754,  0.1860],
         [-2.5981,  0.3323],
         [-1.3922, -0.3851],
         [-1.4464,  0.0808],
         [-0.7548, -0.4869],
         [-3.6646,  1.9848],
         [-1.0310,  0.0175],
         [-2.8434,  0.4483],
         [-1.2368,  0.4423],
```

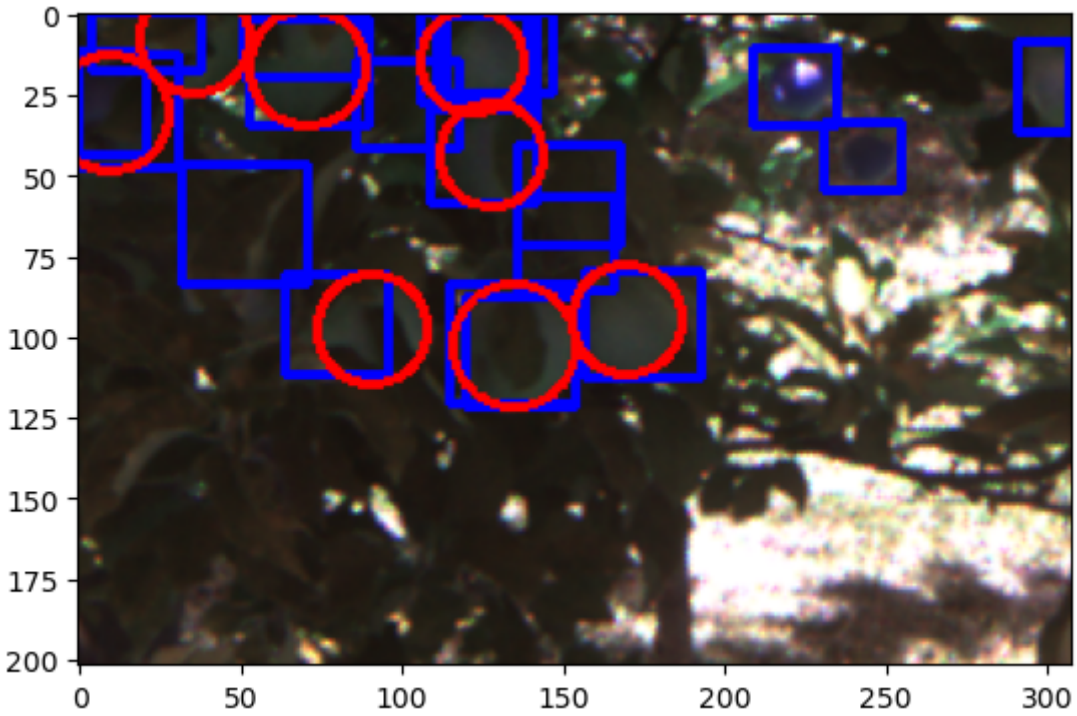
```
                [ 1.6803, -2.3304],
                [-1.9266,  0.8586],
                [-2.9515,  0.8780],
                [-1.5091,  0.4721],
                [-2.8352,  1.0438],
                [-0.1513, -0.7212],
                [ 0.4975, -1.1515],
                [-1.7510,  0.9635],
                [-3.3693,  1.5191],
                [-2.6501,  0.7696],
                [-2.5737,  0.9517],
                [-3.9491,  2.2274],
                [-1.7105,  0.5427],
                [-0.1552, -1.1131],
                [-3.2122,  1.4336],
                [ 1.7039, -2.3447],
                [-0.7723,  0.3358],
                [-3.1100,  1.0912],
                [ 1.3810, -2.3694],
                [-2.4689,  0.7734],
                [-0.0500, -0.4852],
                [-1.5556, -0.1416],
                [-3.8538,  2.5050],
                [-3.1387,  1.2730],
                [-1.5891,  0.1376],
                [-4.4379,  2.6524],
                [ 0.8854, -2.1103],
                [-1.5883,  0.5185],
                [-1.8353,  0.9722],
                [-3.6451,  1.9937],
                [-0.8031, -0.3985],
                [-2.0551,  0.6775],
                [-3.2573,  1.5956],
                [-0.9570, -0.4088],
                [-1.2507,  0.8768],
                [-2.8360,  1.1960],
                [-1.5074,  0.4203]]], grad_fn=<ViewBackward0>), pred_boxes=tensor
        ([[[0.0497, 0.4631, 0.1009, 0.1624],
                [0.4913, 0.3562, 0.0973, 0.1393],
                [0.1720, 0.8196, 0.1182, 0.1814],
                [0.5900, 0.0903, 0.0981, 0.1468],
                [0.4074, 0.2066, 0.1137, 0.1679],
                [0.4333, 0.5284, 0.1139, 0.1646],
                [0.4356, 0.4755, 0.1214, 0.1680],
                [0.2609, 0.4816, 0.1020, 0.1523],
                [0.2282, 0.0607, 0.1102, 0.1231],
                [0.2258, 0.0512, 0.1156, 0.1028],
                [0.6508, 0.6888, 0.1022, 0.1380],
                [0.0683, 0.7250, 0.1177, 0.1881],
                [0.9382, 0.6214, 0.1208, 0.1647],
                [0.9158, 0.6304, 0.1050, 0.1499],
                [0.5006, 0.8839, 0.1092, 0.1439],
                [0.1429, 0.0651, 0.0985, 0.1310],
                [0.0697, 0.0449, 0.1133, 0.0918],
                [0.7727, 0.0766, 0.0952, 0.1276],
                [0.8058, 0.2287, 0.1018, 0.1364],
                [0.4119, 0.2382, 0.1011, 0.1659],
                [0.7120, 0.7564, 0.1255, 0.1391],
                [0.6322, 0.6265, 0.1027, 0.1248],
                [0.9738, 0.1187, 0.0519, 0.1540],
                [0.9632, 0.8866, 0.0723, 0.1879],
                [0.5698, 0.4806, 0.1172, 0.1671],
                [0.5924, 0.6446, 0.0971, 0.1369],
                [0.9233, 0.1636, 0.0895, 0.1323],
```

```
[0.3074, 0.4827, 0.1068, 0.1492],
[0.4126, 0.0808, 0.0991, 0.1432],
[0.0340, 0.1415, 0.0688, 0.1577],
[0.0344, 0.4458, 0.0692, 0.1459],
[0.4283, 0.5469, 0.1067, 0.1514],
[0.0883, 0.9139, 0.1055, 0.1428],
[0.8675, 0.2037, 0.0894, 0.1309],
[0.9530, 0.6269, 0.0927, 0.1483],
[0.4418, 0.3971, 0.0986, 0.1614],
[0.8243, 0.2579, 0.1018, 0.1423],
[0.2531, 0.3049, 0.1084, 0.1638],
[0.3635, 0.5007, 0.1013, 0.1499],
[0.6929, 0.1386, 0.1027, 0.1409],
[0.7904, 0.2219, 0.0772, 0.1040],
[0.8640, 0.0599, 0.1019, 0.1199],
[0.4938, 0.2823, 0.1041, 0.1515],
[0.4045, 0.6440, 0.1048, 0.1366],
[0.0506, 0.1523, 0.1032, 0.1712],
[0.4312, 0.3632, 0.0988, 0.1399],
[0.7936, 0.2365, 0.0979, 0.1448],
[0.5893, 0.6592, 0.0982, 0.1345],
[0.0435, 0.4382, 0.0875, 0.1686],
[0.3083, 0.4483, 0.1071, 0.1556],
[0.3993, 0.0712, 0.1099, 0.1434],
[0.7237, 0.1171, 0.0842, 0.1188],
[0.2752, 0.8448, 0.1100, 0.1696],
[0.4023, 0.2523, 0.1026, 0.1464],
[0.1685, 0.8047, 0.1110, 0.1722],
[0.2467, 0.5692, 0.1086, 0.1486],
[0.1455, 0.3568, 0.1021, 0.1569],
[0.5122, 0.4410, 0.1023, 0.1453],
[0.4501, 0.4152, 0.1005, 0.1569],
[0.2918, 0.4940, 0.1167, 0.1766],
[0.3195, 0.1439, 0.1021, 0.1346],
[0.2746, 0.5968, 0.1128, 0.1692],
[0.9540, 0.0422, 0.0923, 0.0850],
[0.2327, 0.0925, 0.1206, 0.1646],
[0.8176, 0.0432, 0.0907, 0.0872],
[0.2486, 0.7983, 0.1120, 0.1928],
[0.0427, 0.0444, 0.0866, 0.0885],
[0.0773, 0.6438, 0.1047, 0.1587],
[0.3322, 0.1426, 0.1019, 0.1316],
[0.9729, 0.1161, 0.0534, 0.1416],
[0.9199, 0.0688, 0.1041, 0.1381],
[0.4763, 0.6574, 0.1030, 0.1480],
[0.3735, 0.5805, 0.1164, 0.1597],
[0.5984, 0.5361, 0.0934, 0.1306],
[0.6244, 0.7762, 0.1035, 0.0933],
[0.5986, 0.5051, 0.0957, 0.1399],
[0.1697, 0.3266, 0.1254, 0.1864],
[0.0413, 0.9150, 0.0841, 0.1359],
[0.4097, 0.2163, 0.1073, 0.1563],
[0.0378, 0.1295, 0.0762, 0.1470],
[0.9442, 0.6248, 0.1099, 0.1586],
[0.4381, 0.5099, 0.1266, 0.1868],
[0.9716, 0.2355, 0.0564, 0.1411],
[0.4248, 0.0619, 0.1077, 0.1241],
[0.1825, 0.3597, 0.1143, 0.1707],
[0.7873, 0.9083, 0.1269, 0.1491],
[0.9544, 0.6233, 0.0917, 0.1522],
[0.5389, 0.3578, 0.0975, 0.1391],
[0.3185, 0.9128, 0.1393, 0.1502],
[0.4462, 0.5220, 0.1126, 0.1705],
[0.5493, 0.4693, 0.1021, 0.1493],
```

```
          [0.8171, 0.2267, 0.1031, 0.1397],
          [0.5594, 0.6618, 0.1118, 0.1326],
          [0.5262, 0.3538, 0.1027, 0.1497],
          [0.5459, 0.4751, 0.1109, 0.1501],
          [0.4901, 0.8733, 0.1123, 0.1440],
          [0.1564, 0.1315, 0.1059, 0.1444],
          [0.7105, 0.1032, 0.0865, 0.1187],
          [0.6026, 0.4974, 0.0980, 0.1314],
          [0.5412, 0.2465, 0.0986, 0.1404]]], grad_fn=<SigmoidBackward0>), au
xiliary_outputs=None, last_hidden_state=tensor([[[-0.7152,  0.4243, -0.8542,
..., -0.6747,  0.4574, -0.1734],
         [-0.7699,  0.2647, -0.3410,  ..., -0.1171,  1.6243,  0.4770],
         [ 0.2378, -1.0165, -1.2322,  ...,  0.0201,  1.0470, -0.5438],
         ...,
         [-0.0743, -0.1305, -0.8067,  ..., -1.0556,  1.1716,  1.1494],
         [-0.2693, -0.4093, -0.8773,  ..., -0.5500,  0.7151,  0.8878],
         [-0.6821,  0.0145, -0.4526,  ..., -0.3567,  1.3153,  0.4582]]],
       grad_fn=<NativeLayerNormBackward0>), decoder_hidden_states=None, deco
der_attentions=None, cross_attentions=None, encoder_last_hidden_state=tensor
([[[-4.9588e-02, -6.1598e-03, -1.8782e-02,  ...,  1.7698e-01,
          -4.1701e-03,  1.2628e-01],
         [-5.8087e-02, -2.7901e-02, -2.0161e-02,  ...,  2.0968e-01,
          -2.6250e-01,  1.8726e-01],
         [ 8.9099e-03, -1.0147e-02,  5.4673e-03,  ..., -1.8434e-01,
          -2.9992e-01,  1.7289e-01],
         ...,
         [ 3.3067e-02,  2.8096e-04,  5.9991e-03,  ..., -4.5119e-01,
           8.6766e-02, -1.7204e-01],
         [ 3.7236e-02,  1.5046e-02,  8.1606e-04,  ...,  6.2884e-02,
          -2.8772e-01,  3.4947e-02],
         [ 3.5166e-02, -8.5271e-04,  4.7765e-03,  ..., -1.4934e-01,
          -4.1334e-02, -4.0429e-02]]], grad_fn=<NativeLayerNormBackward0>),
encoder_hidden_states=None, encoder_attentions=None)
Detected apple with confidence 0.583 at location [136.35, 57.88, 166.31, 86.
02]
Detected apple with confidence 0.645 at location [64.66, 81.9, 96.07, 112.6
7]
Detected apple with confidence 0.889 at location [51.75, -0.05, 87.35, 20.7
2]
Detected apple with confidence 0.5 at location [4.02, -0.2, 38.91, 18.34]
Detected apple with confidence 0.974 at location [157.46, 80.21, 193.55, 11
3.96]
Detected apple with confidence 0.854 at location [111.81, 1.86, 142.32, 30.
8]
Detected apple with confidence 0.844 at location [-0.12, 12.65, 21.08, 44.5
1]
Detected apple with confidence 0.923 at location [231.57, 34.33, 255.33, 55.
34]
Detected apple with confidence 0.66 at location [136.07, 41.72, 168.13, 72.3
3]
Detected apple with confidence 0.865 at location [-0.32, 13.47, 31.47, 48.0
6]
Detected apple with confidence 0.977 at location [106.07, -0.1, 139.91, 28.8
7]
Detected apple with confidence 0.833 at location [209.92, 11.64, 235.86, 35.
65]
Detected apple with confidence 0.982 at location [53.1, 2.07, 90.22, 35.32]
Detected apple with confidence 0.639 at location [86.62, 15.5, 118.0, 42.09]
Detected apple with confidence 0.839 at location [291.45, 9.15, 307.88, 37.7
6]
Detected apple with confidence 0.723 at location [32.97, 47.16, 71.6, 84.8]
Detected apple with confidence 0.983 at location [109.66, 27.9, 142.71, 59.4
8]
Detected apple with confidence 0.977 at location [115.45, 84.13, 154.43, 12
```

```
1.86]
Detected apple with confidence 0.607 at location [114.26, -0.03, 147.42, 25.
03]
Detected apple with confidence 0.952 at location [120.08, 88.23, 154.77, 12
2.67]
```

Out[58]:
```
<matplotlib.image.AxesImage at 0x7f42edcb6a00>
```



Almonds

In [60]:
```
from transformers import DetrImageProcessor

feature_extractor = DetrImageProcessor.from_pretrained("facebook/detr-resnet
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almond
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almonds/
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
batch = next(iter(train_dataloader))
batch.keys()
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:780: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
```

Out[60]:
```
dict_keys(['pixel_values', 'pixel_mask', 'labels'])
```

In [61]:
```
from pytorch_lightning import Trainer
model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)

max_steps = 1000

trainer = Trainer(max_steps=max_steps, gradient_clip_val=0.1)
trainer.fit(model)
```

```
Some weights of DetrForObjectDetection were not initialized from the model c
heckpoint at facebook/detr-resnet-50 and are newly initialized because the s
hapes did not match:
- class_labels_classifier.weight: found shape torch.Size([92, 256]) in the c
heckpoint and torch.Size([2, 256]) in the model instantiated
- class_labels_classifier.bias: found shape torch.Size([92]) in the checkpoi
nt and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                   | Params
-------------------------------------------------
0 | model | DetrForObjectDetection | 41.5 M
-------------------------------------------------
18.0 M    Trainable params
23.5 M    Non-trainable params
41.5 M    Total params
166.007   Total estimated model params size (MB)
Sanity Checking: 0it [00:00, ?it/s]
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/traine
r/connectors/data_connector.py:430: PossibleUserWarning: The dataloader, val
_dataloader, does not have many workers which may be a bottleneck. Consider
increasing the value of the `num_workers` argument` (try 4 which is the numb
er of cpus on this machine) in the `DataLoader` init to improve performance.
  rank_zero_warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/traine
r/connectors/data_connector.py:430: PossibleUserWarning: The dataloader, tra
in_dataloader, does not have many workers which may be a bottleneck. Conside
r increasing the value of the `num_workers` argument` (try 4 which is the nu
mber of cpus on this machine) in the `DataLoader` init to improve performanc
e.
  rank_zero_warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/loops/
fit_loop.py:280: PossibleUserWarning: The number of training batches (35) is
smaller than the logging interval Trainer(log_every_n_steps=50). Set a lower
value for log_every_n_steps if you want to see logs for the training epoch.
  rank_zero_warn(
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
```

```
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
`Trainer.fit` stopped: `max_steps=1000` reached.
```

In [62]:
```
torch.save(model.state_dict(), "models/detr_almonds_v1.pt")
```

In [63]:
```
from transformers import AutoImageProcessor, AutoModelForObjectDetection
import torch
from PIL import Image
import requests
import numpy as np
import cv2

val_id = np.random.randint(0, len(val_dataset))

file_name = val_dataset.coco.loadImgs(
                                    val_dataset[val_id][1]["image_id"].item(


image = Image.open(os.path.join("datasets/acfr-fruit-dataset/almonds/coco/va
                            file_name
                    ))
image_processor = AutoImageProcessor.from_pretrained("facebook/detr-resnet-5

cats = train_dataset.coco.cats
id2label = {k: v['name'] for k,v in cats.items()}

val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
pixel_values, pixel_mask, labels = list(iter(val_dataloader))[val_id].values

outputs = model(pixel_values=pixel_values, pixel_mask=pixel_mask)


# convert outputs (bounding boxes and class logits) to COCO API
target_sizes = torch.tensor([image.size[::-1]])
results = image_processor.post_process_object_detection(outputs, threshold=0
    0
]

image = np.array(image)
image = image[:,:,::-1].copy()
for score, label, box in zip(results["scores"], results["labels"], results["
    box = [round(i, 2) for i in box.tolist()]
    print(
        f"Detected {id2label[label.item()]} with confidence "
        f"{round(score.item(), 3)} at location {box}"
    )
    cv2.rectangle(image, (int(box[0]), int(box[1])), (int(box[2]), int(box[
```

```
for idx, row in dataset["almonds"]["global_df"].loc[dataset["almonds"]["glob
    cv2.rectangle(image, (int(row.x), int(row.y)), (int(row.x) + int(row.dx)

plt.imshow(image)
```

```
Could not find image processor class in the image processor config or the mo
del config. Loading based on pattern matching with the model's feature extra
ctor configuration.
```

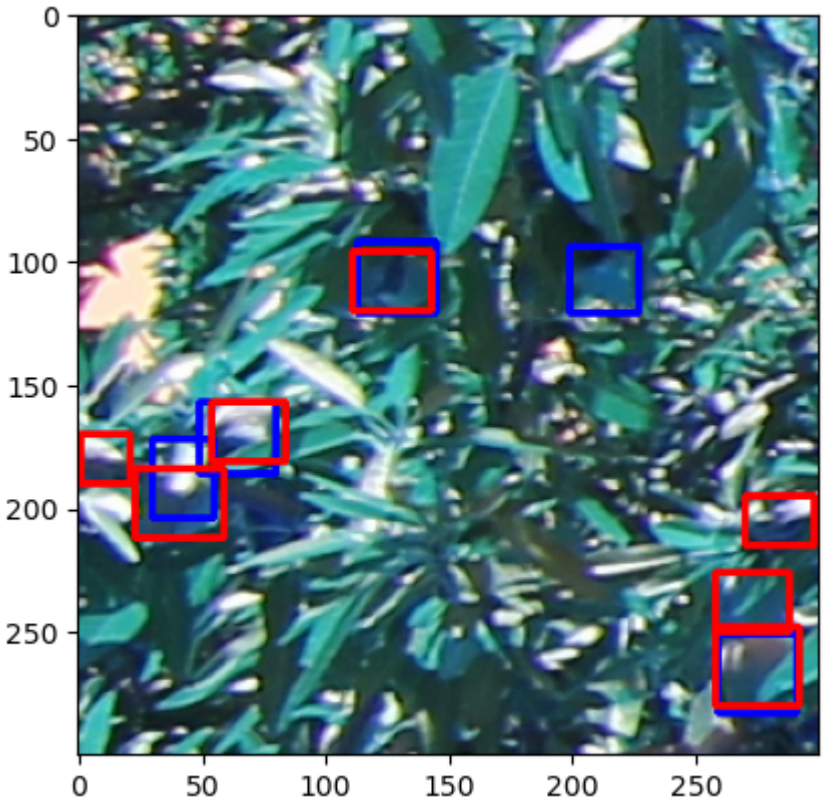
```
DetrObjectDetectionOutput(loss=None, loss_dict=None, logits=tensor([[[-1.181
9,  1.4676],
         [-3.1141,  1.7593],
         [-1.0492,  1.0742],
         [-3.2896,  1.4333],
         [-3.0169,  0.9654],
         [-2.8712,  1.4123],
         [-2.5567,  1.4691],
         [-2.4318,  2.1766],
         [-2.7787,  1.4407],
         [-3.0860,  1.9875],
         [-2.8570,  1.7201],
         [ 0.2617, -0.4945],
         [-2.6143,  2.3524],
         [-2.4785,  2.2694],
         [-2.9263,  1.3080],
         [-2.3970,  2.4236],
         [-3.3014,  1.9944],
         [-3.1632,  1.8737],
         [-2.4271,  1.9106],
         [-2.8477,  1.8154],
         [ 1.1873, -1.6236],
         [-2.4795,  2.0186],
         [-3.8380,  2.9723],
         [-1.5974,  1.3463],
         [ 0.0809, -1.2773],
         [-2.8013,  2.0916],
         [-2.3349,  2.0616],
         [-1.5445,  1.1221],
         [-3.1585,  0.9438],
         [-3.0991,  2.5305],
         [-1.5264,  1.5266],
         [-2.1218,  0.3747],
         [-2.3498,  1.8750],
         [-1.6497,  1.0880],
         [-2.8019,  2.1723],
         [-0.6835, -0.5806],
         [-2.9981,  2.6517],
         [-2.6513,  2.2803],
         [-2.2340,  1.5501],
         [-2.5403,  0.5973],
         [-1.7492,  1.0344],
         [-1.7469,  0.7071],
         [-2.4142,  0.3341],
         [-2.2151,  1.5504],
         [-0.6802,  0.5506],
         [-2.1192,  1.0924],
         [-2.5777,  1.9277],
         [-2.8873,  1.4831],
         [-1.3775,  1.2589],
         [-1.0083, -0.3493],
         [-2.4470,  0.5877],
         [-2.9822,  1.4766],
         [-1.8049,  0.2021],
         [ 0.5701, -2.1073],
         [-1.6221,  1.6674],
         [-2.0935,  1.6395],
         [-2.1081,  1.9518],
         [-3.1638,  1.6319],
         [-2.8748,  1.5948],
         [-4.8197,  3.4939],
         [-3.1150,  1.6149],
         [-2.6049,  2.0212],
         [-2.2795,  1.5933],
```

```
                [-2.9386,  2.2383],
                [-3.7083,  2.7002],
                [-0.6806,  0.1860],
                [-3.2895,  2.3156],
                [-2.6316,  2.2672],
                [-1.9342,  0.8883],
                [-3.0773,  2.8594],
                [-3.7995,  3.0735],
                [-2.9932,  2.0283],
                [-2.4300,  1.8525],
                [-2.6650,  1.9257],
                [-2.7923,  1.4652],
                [-3.0269,  2.2420],
                [-0.2218, -0.2974],
                [-0.3656,  0.4080],
                [-2.5463,  0.4564],
                [-3.6109,  2.8712],
                [-1.2959,  0.6535],
                [-2.2911,  1.1860],
                [-2.6811,  2.5171],
                [-2.9431,  0.7937],
                [-1.8063,  1.6432],
                [-1.7065,  0.9621],
                [-1.6812,  1.5663],
                [-2.5186,  1.3805],
                [-2.5442,  1.0217],
                [-2.4565,  0.9894],
                [-2.1680,  0.3917],
                [-3.3935,  2.5402],
                [-2.1811,  0.9929],
                [-2.7692,  0.9678],
                [-2.7115,  1.7488],
                [-2.4402,  0.9868],
                [-2.9921,  2.5193],
                [-1.5776, -0.4886],
                [-2.9667,  2.4357],
                [-3.7134,  2.1443]]], grad_fn=<ViewBackward0>), pred_boxes=tensor
        ([[[0.0398, 0.6131, 0.0756, 0.0869],
                [0.5757, 0.3479, 0.0864, 0.1093],
                [0.1486, 0.6831, 0.1051, 0.0970],
                [0.6348, 0.1884, 0.0864, 0.1038],
                [0.4185, 0.1553, 0.0921, 0.1048],
                [0.5391, 0.7919, 0.0995, 0.1101],
                [0.4938, 0.5520, 0.0950, 0.0855],
                [0.2263, 0.5833, 0.0968, 0.0859],
                [0.2323, 0.0925, 0.1203, 0.0753],
                [0.2341, 0.0517, 0.1149, 0.0805],
                [0.7237, 0.8410, 0.0909, 0.0812],
                [0.1419, 0.6283, 0.0834, 0.1052],
                [0.8966, 0.5622, 0.0984, 0.0982],
                [0.8634, 0.5449, 0.0879, 0.0863],
                [0.4962, 0.8482, 0.0882, 0.0845],
                [0.1283, 0.5279, 0.0913, 0.0919],
                [0.2158, 0.1026, 0.1141, 0.0791],
                [0.8527, 0.1417, 0.0948, 0.1070],
                [0.8126, 0.4002, 0.0963, 0.0851],
                [0.3638, 0.9341, 0.1004, 0.1024],
                [0.9177, 0.8914, 0.1063, 0.1056],
                [0.8586, 0.6172, 0.0910, 0.0907],
                [0.9259, 0.0809, 0.1322, 0.1325],
                [0.9541, 0.9127, 0.0829, 0.1448],
                [0.7125, 0.3585, 0.0940, 0.0903],
                [0.6995, 0.4023, 0.0829, 0.0844],
                [0.9086, 0.3187, 0.0847, 0.0917],
```

```
[0.2620, 0.6775, 0.0991, 0.0870],
[0.4999, 0.1201, 0.0874, 0.1152],
[0.0386, 0.3823, 0.0710, 0.1190],
[0.0383, 0.6935, 0.0737, 0.1074],
[0.4785, 0.8463, 0.0906, 0.1000],
[0.1112, 0.9399, 0.1233, 0.1051],
[0.8969, 0.3170, 0.0932, 0.0938],
[0.9408, 0.6184, 0.1044, 0.1649],
[0.4625, 0.3514, 0.0872, 0.0863],
[0.9064, 0.3318, 0.0858, 0.0949],
[0.2181, 0.5631, 0.1062, 0.0894],
[0.3661, 0.6229, 0.0890, 0.0751],
[0.7511, 0.1708, 0.0878, 0.1006],
[0.8876, 0.2544, 0.0912, 0.1174],
[0.8630, 0.1968, 0.0952, 0.1060],
[0.6454, 0.1900, 0.0880, 0.1067],
[0.2864, 0.6791, 0.1002, 0.0926],
[0.0821, 0.5161, 0.1244, 0.1008],
[0.4037, 0.3543, 0.0955, 0.1041],
[0.8124, 0.3911, 0.0889, 0.0894],
[0.5696, 0.8063, 0.0928, 0.0970],
[0.0426, 0.5191, 0.0811, 0.0933],
[0.3854, 0.3561, 0.1003, 0.0978],
[0.4143, 0.1598, 0.0910, 0.1007],
[0.7541, 0.1755, 0.0902, 0.1032],
[0.3951, 0.9510, 0.1005, 0.0883],
[0.4314, 0.3569, 0.1045, 0.0952],
[0.1582, 0.6229, 0.0912, 0.0940],
[0.2300, 0.6169, 0.1140, 0.1009],
[0.1458, 0.5626, 0.0846, 0.0901],
[0.5732, 0.3509, 0.0863, 0.1128],
[0.5563, 0.3637, 0.0861, 0.1082],
[0.4871, 0.5930, 0.9886, 0.8947],
[0.3998, 0.1643, 0.0859, 0.0966],
[0.2452, 0.5700, 0.1130, 0.0957],
[0.9361, 0.0486, 0.1075, 0.0819],
[0.2370, 0.4035, 0.1045, 0.1038],
[0.8307, 0.0735, 0.1021, 0.1144],
[0.2459, 0.6766, 0.1208, 0.0891],
[0.0489, 0.2249, 0.0911, 0.0999],
[0.0971, 0.5322, 0.0973, 0.0889],
[0.3698, 0.3514, 0.1036, 0.1026],
[0.9560, 0.4602, 0.0821, 0.0907],
[0.8930, 0.1742, 0.1223, 0.1317],
[0.4744, 0.5393, 0.0969, 0.0942],
[0.2744, 0.6731, 0.1040, 0.0924],
[0.8119, 0.3951, 0.0863, 0.0840],
[0.7383, 0.9304, 0.0891, 0.0844],
[0.7776, 0.3671, 0.0805, 0.0856],
[0.2153, 0.5724, 0.1037, 0.0961],
[0.0435, 0.7140, 0.0864, 0.0979],
[0.4143, 0.1598, 0.0919, 0.0980],
[0.9018, 0.1891, 0.1194, 0.1265],
[0.9358, 0.9040, 0.0879, 0.1085],
[0.4322, 0.3553, 0.0985, 0.0947],
[0.9566, 0.4681, 0.0779, 0.0994],
[0.5611, 0.0499, 0.1152, 0.0905],
[0.2254, 0.5573, 0.1032, 0.0825],
[0.8911, 0.9476, 0.1085, 0.0936],
[0.9486, 0.6756, 0.0900, 0.0844],
[0.6494, 0.3009, 0.0924, 0.1160],
[0.3849, 0.9479, 0.1069, 0.0906],
[0.5481, 0.7746, 0.0923, 0.0960],
[0.6608, 0.3061, 0.0998, 0.1232],
```

```
          [0.8780, 0.2397, 0.1051, 0.1375],
          [0.6902, 0.8485, 0.0957, 0.0801],
          [0.6484, 0.2204, 0.0866, 0.0998],
          [0.6856, 0.3868, 0.0910, 0.0915],
          [0.5631, 0.7826, 0.0889, 0.1031],
          [0.2222, 0.3909, 0.1024, 0.0967],
          [0.6922, 0.0486, 0.1230, 0.0883],
          [0.8361, 0.3935, 0.0884, 0.0890],
          [0.7422, 0.1869, 0.0998, 0.1210]]], grad_fn=<SigmoidBackward0>), au
xiliary_outputs=None, last_hidden_state=tensor([[[-0.4560,  1.1163,  0.0187,
...,  0.6876,  1.3836, -1.0809],
         [-0.4567,  0.9419, -0.3144,  ..., -0.0877,  3.0631,  0.1980],
         [-0.5467,  0.0494, -0.3663,  ...,  0.5781,  1.8565, -1.3879],
         ...,
         [-0.0852,  0.1799, -1.1266,  ...,  0.3421,  1.0624,  0.8010],
         [ 0.1842,  1.0075, -0.3700,  ..., -0.4093,  3.2565, -0.0510],
         [-0.1255,  0.9927, -0.6689,  ..., -0.4278,  2.6679,  0.2909]]],
       grad_fn=<NativeLayerNormBackward0>), decoder_hidden_states=None, deco
der_attentions=None, cross_attentions=None, encoder_last_hidden_state=tensor
([[[-0.0417,  0.0074, -0.0211,  ...,  0.1831,  0.0247,  0.1357],
         [-0.0384,  0.0024, -0.0112,  ...,  0.2105,  0.0194,  0.2051],
         [-0.0592, -0.0006, -0.0049,  ..., -0.0409, -0.2044, -0.1848],
         ...,
         [ 0.1551,  0.0172,  0.0089,  ..., -0.1303, -0.1361, -0.2046],
         [ 0.1487,  0.0064,  0.0089,  ..., -0.1227, -0.1681,  0.1484],
         [ 0.1243, -0.0077, -0.0141,  ..., -0.3261, -0.0702,  0.3489]]],
       grad_fn=<NativeLayerNormBackward0>), encoder_hidden_states=None, enco
der_attentions=None)
Detected almond with confidence 0.681 at location [30.06, 172.7, 55.1, 204.2
7]
Detected almond with confidence 0.943 at location [259.38, 251.56, 291.26, 2
83.25]
Detected almond with confidence 0.795 at location [199.67, 94.01, 227.85, 12
1.11]
Detected almond with confidence 0.936 at location [113.75, 92.78, 145.1, 12
1.35]
Detected almond with confidence 0.519 at location [49.04, 157.31, 80.16, 18
6.15]
```

Out[63]: <matplotlib.image.AxesImage at 0x7f42ec245730>



Mangoes

In [64]:
```
from transformers import DetrImageProcessor

feature_extractor = DetrImageProcessor.from_pretrained("facebook/detr-resnet
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoe
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoes/
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
batch = next(iter(train_dataloader))
batch.keys()
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:780: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
```

Out[64]:
```
dict_keys(['pixel_values', 'pixel_mask', 'labels'])
```

In [65]:
```
from pytorch_lightning import Trainer
model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)
```

```
max_steps = 1000

trainer = Trainer(max_steps=max_steps, gradient_clip_val=0.1)
trainer.fit(model)
```

```
Some weights of DetrForObjectDetection were not initialized from the model c
heckpoint at facebook/detr-resnet-50 and are newly initialized because the s
hapes did not match:
- class_labels_classifier.weight: found shape torch.Size([92, 256]) in the c
heckpoint and torch.Size([2, 256]) in the model instantiated
- class_labels_classifier.bias: found shape torch.Size([92]) in the checkpoi
nt and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                   | Params
-------------------------------------------------
0 | model | DetrForObjectDetection | 41.5 M
-------------------------------------------------
18.0 M    Trainable params
23.5 M    Non-trainable params
41.5 M    Total params
166.007   Total estimated model params size (MB)
Sanity Checking: 0it [00:00, ?it/s]
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/traine
r/connectors/data_connector.py:430: PossibleUserWarning: The dataloader, val
_dataloader, does not have many workers which may be a bottleneck. Consider
increasing the value of the `num_workers` argument` (try 4 which is the numb
er of cpus on this machine) in the `DataLoader` init to improve performance.
  rank_zero_warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/traine
r/connectors/data_connector.py:430: PossibleUserWarning: The dataloader, tra
in_dataloader, does not have many workers which may be a bottleneck. Conside
r increasing the value of the `num_workers` argument` (try 4 which is the nu
mber of cpus on this machine) in the `DataLoader` init to improve performanc
e.
  rank_zero_warn(
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/utilit
ies/data.py:77: UserWarning: Trying to infer the `batch_size` from an ambigu
ous collection. The batch size we found is 10. To avoid any miscalculations,
use `self.log(..., batch_size=batch_size)`.
  warning_cache.warn(
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
```

```
`Trainer.fit` stopped: `max_steps=1000` reached.
```

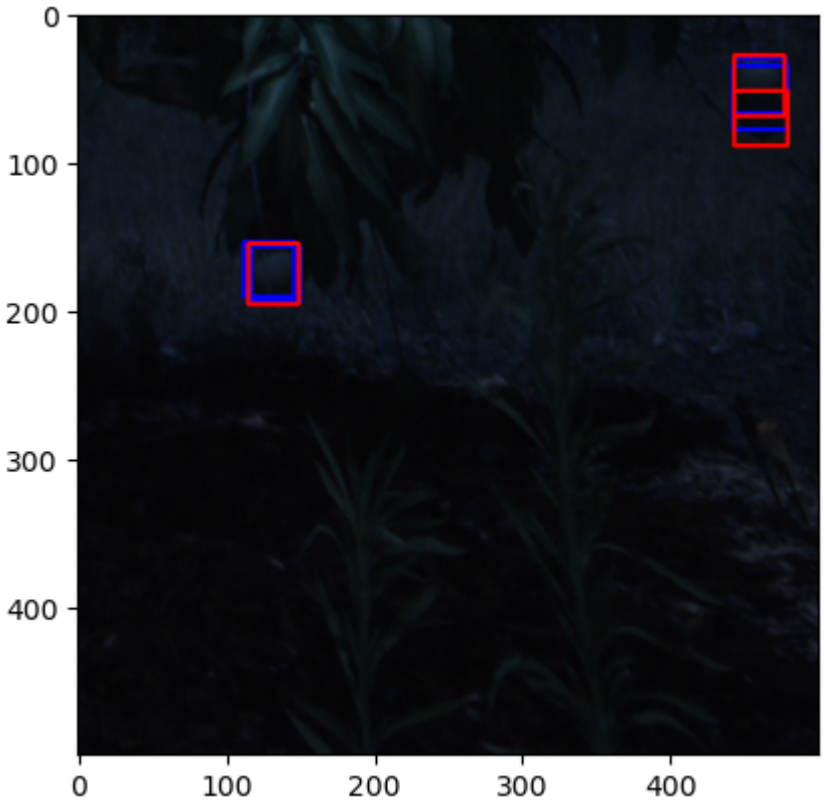
In [66]:
```
torch.save(model.state_dict(), "models/detr_mangoes_v1.pt")
```

In [67]:
```
from transformers import AutoImageProcessor, AutoModelForObjectDetection
import torch
from PIL import Image
import requests
import numpy as np
import cv2

val_id = np.random.randint(0, len(val_dataset))

file_name = val_dataset.coco.loadImgs(
                                    val_dataset[val_id][1]["image_id"].item(


image = Image.open(os.path.join("datasets/acfr-fruit-dataset/mangoes/coco/va
                            file_name
                    ))
image_processor = AutoImageProcessor.from_pretrained("facebook/detr-resnet-5

cats = train_dataset.coco.cats
id2label = {k: v['name'] for k,v in cats.items()}

val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
pixel_values, pixel_mask, labels = list(iter(val_dataloader))[val_id].values

outputs = model(pixel_values=pixel_values, pixel_mask=pixel_mask)


# convert outputs (bounding boxes and class logits) to COCO API
target_sizes = torch.tensor([image.size[::-1]])
results = image_processor.post_process_object_detection(outputs, threshold=0
     0
]

image = np.array(image)
image = image[:,:,::-1].copy()
for score, label, box in zip(results["scores"], results["labels"], results["
     box = [round(i, 2) for i in box.tolist()]
     print(
         f"Detected {id2label[label.item()]} with confidence "
         f"{round(score.item(), 3)} at location {box}"
     )
     cv2.rectangle(image, (int(box[0]), int(box[1])), (int(box[2]), int(box[

for idx, row in dataset["mangoes"]["global_df"].loc[dataset["mangoes"]["glob
    cv2.rectangle(image, (int(row.x), int(row.y)), (int(row.x) + int(row.dx)

plt.imshow(image)
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
Could not find image processor class in the image processor config or the mo
del config. Loading based on pattern matching with the model's feature extra
ctor configuration.
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:780: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
```

```
Detected mango with confidence 0.888 at location [116.58, 157.09, 149.64, 19
4.09]
Detected mango with confidence 0.748 at location [112.64, 154.46, 146.56, 19
1.35]
Detected mango with confidence 0.906 at location [117.11, 157.59, 150.05, 19
5.55]
Detected mango with confidence 0.99 at location [443.02, 30.82, 478.69, 67.3
9]
Detected mango with confidence 0.973 at location [442.85, 35.29, 479.24, 78.
93]
```

Out[67]: `<matplotlib.image.AxesImage at 0x7f42db076460>`



YOLOS

The "You Only Look at One Sequence" model or YOLOS for short, combines the original Vision Transformer (Dosovitskiy et al., 2021) with the object detection capabilities of DETR (Carion et al., 2020), in an effort to create a versatile, adaptable and spatially independent Transformer based object detection model (Fang et al., 2021). This is achieved, by taking the original Vision Transformer as it is and removing the classification token. Instead, 100 randomly initialized learnable Detection Tokens (`[DET]`) are appended to the input patch embeddings for object detection (Fang et al., 2021). These tokens serve the same purpose as the N object queries in DETR and create one hundred different output embeddings. YOLOS Detector Heads then take the Detection Tokens as input and perform both the classification and bounding box regression through one Forward Neural Network with ReLu activations, containing two hidden layers. Because YOLOS internally only works with sequence and essentially create sequence to sequence embeddings, the model can in theory perform any dimensional object detection without the need of knowing the exact spatial structure and geometry (Fang et al., 2021). At the same time, because of its simplicity, the model

can be easily adapted to other state-of-the-art transformer architectures from the NLP space such as BERT (Fang et al., 2021).

In order to Fine-Tune YOLOS on the acfr-fruit-dataset, again, a custom collate function has to be defined as the model does not expect a pixel mask as input because it doesn't support pixel-level segmentation. Other than that, the Training procedure for YOLOS remains the same as for DETR, as both models are trained in the same fashion using the same methods (Fang et al., 2021). The model is initialized using a pre-trained YOLOS checkpoint from Huggingface model hub, which has been pre-trained on ImageNet-1k and then Fine-Tuned on COCO 2017 object detection. It is then trained for 2000 optimisation steps, also following the propositions of Fang et al. For the Almonds and Mangoes, two models are trained instead of one, as training for 2000 steps showed overfitting behaviour, where the validation loss was rising while the train loss was degrading. Therefore, the loss curves plotted to Tensorboard for both checkpoints have been used to define early stopping points which equates to 734 optimization steps for almonds and 1436 for mangoes respectively.

In [4]:
```
# custom collate function to pad images and convert them to the required for
def collate_fn(batch):
    pixel_values = [item[0] for item in batch]
    encoding = feature_extractor.pad(pixel_values, return_tensors="pt")
    labels = [item[1] for item in batch]
    batch = {}
    batch['pixel_values'] = encoding['pixel_values']
    batch['labels'] = labels
    return batch
```

Apples

In [22]:
```
from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os

# define dataloaders and initialize feature extractor
feature_extractor = AutoFeatureExtractor.from_pretrained("hustvl/yolos-small
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples/c
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
```

```
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
```

In [5]:
```
import pytorch_lightning as pl
from transformers import AutoModelForObjectDetection
import torch

# define Yolos VIT as a Lightning Module, so it can be trained with pytorch
class Yolos(pl.LightningModule):

    def __init__(self, lr, weight_decay, trained="hustvl/yolos-small"):
```

```
        super().__init__()
        self.model = AutoModelForObjectDetection.from_pretrained(trained,
                                                                 num_labels=1,
                                                                 ignore_mismatch
        self.lr = lr
        self.weight_decay = weight_decay

    def forward(self, pixel_values, output_attentions=False):
        return self.model(pixel_values=pixel_values, output_attentions=outpu

    def common_step(self, batch): # step that is executed during training an
        pixel_values = batch["pixel_values"] # extract pixel values from bat
        labels = [{k: v.to(self.device) for k, v in t.items()} for t in batc

        outputs = self.model(pixel_values=pixel_values, labels=labels) # run
        return outputs.loss, outputs.loss_dict

    def training_step(self, batch, batch_idx): # train step with custom logg
        loss, loss_dict = self.common_step(batch) # get losses and perform c
        self.log("training_loss", loss)
        for k,v in loss_dict.items():
            self.log("train_" + k, v.item())
        return loss

    def validation_step(self, batch, batch_idx): # validation step with cust
        loss, loss_dict = self.common_step(batch)
        self.log("validation_loss", loss)
        for k,v in loss_dict.items():
          self.log("validation_" + k, v.item())
        return loss

    def configure_optimizers(self): # define Optimizer, AdamW is recommended
        optimizer = torch.optim.AdamW(self.parameters(), lr=self.lr, weight_
        return optimizer

    def train_dataloader(self):
        return train_dataloader

    def val_dataloader(self):
        return val_dataloader
```

In [23]:
```
# Check Shapes
batch = next(iter(train_dataloader))
print(batch.keys())
pixel, target = train_dataset[2]
print("pixel shape", pixel.shape)
print(target)
```

```
dict_keys(['pixel_values', 'labels'])
pixel shape torch.Size([3, 512, 780])
{'size': tensor([512, 780]), 'image_id': tensor([2]), 'class_labels': tensor
([0, 0]), 'boxes': tensor([[0.2646, 0.1559, 0.1201, 0.1832],
        [0.4026, 0.1782, 0.1494, 0.2277]]), 'area': tensor([ 8978.5391, 1374
8.3057]), 'iscrowd': tensor([0, 0]), 'orig_size': tensor([202, 308])}
```

In [54]:
```
# initialize Model
model = Yolos(lr=2.5e-5, weight_decay=1e-4)
outputs = model(pixel_values=batch['pixel_values'])
outputs.logits.shape
```

```
Some weights of YolosForObjectDetection were not initialized from the model
checkpoint at hustvl/yolos-small and are newly initialized because the shape
s did not match:
- class_labels_classifier.layers.2.weight: found shape torch.Size([92, 384])
in the checkpoint and torch.Size([2, 384]) in the model instantiated
- class_labels_classifier.layers.2.bias: found shape torch.Size([92]) in the
checkpoint and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
```

Out[54]:
```
torch.Size([1, 100, 2])
```

In [27]:
```
from pytorch_lightning import Trainer
# Train Model using pytorch lightning
torch.device("cuda")
trainer = Trainer(max_steps=2000, gradient_clip_val=0.1, accumulate_grad_bat
trainer.fit(model=model) # start training
```

```
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                    | Params
-------------------------------------------------
0 | model | YolosForObjectDetection | 30.7 M
-------------------------------------------------
30.7 M    Trainable params
0         Non-trainable params
30.7 M    Total params
122.600   Total estimated model params size (MB)
Sanity Checking: 0it [00:00, ?it/s]
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
`Trainer.fit` stopped: `max_steps=2000` reached.
```

In [28]:
```
torch.save(model.state_dict(), "models/yolos_apples_v9.pt") # save model for
```

In [55]:
```
model.load_state_dict(torch.load("models/yolos_apples_v9.pt")) # load saved
```

Out[55]:
```
<All keys matched successfully>
```

In [88]:
```
from transformers import AutoImageProcessor, AutoModelForObjectDetection
import torch
from PIL import Image
import requests
import numpy as np
import cv2

# Display random sample with predictions and grount truth to check if the mo

val_id = np.random.randint(0, len(val_dataset))

file_name = val_dataset.coco.loadImgs(
                                    val_dataset[val_id][1]["image_id"].item(
```

```
image = Image.open(os.path.join("datasets/acfr-fruit-dataset/apples/coco/val
                                file_name
                    ))
image_processor = AutoImageProcessor.from_pretrained("hustvl/yolos-tiny")

cats = train_dataset.coco.cats
id2label = {k: v['name'] for k,v in cats.items()}


pixel_values = val_dataset[val_id][0].unsqueeze(0)

outputs = model(pixel_values=pixel_values)


# convert outputs (bounding boxes and class logits) to COCO API
target_sizes = torch.tensor([image.size[::-1]])
results = image_processor.post_process_object_detection(outputs, threshold=0
    0
]

image = np.array(image)
image = image[:,:,::-1].copy()
for score, label, box in zip(results["scores"], results["labels"], results["
    box = [round(i, 2) for i in box.tolist()]
    print(
        f"Detected {id2label[label.item()]} with confidence "
        f"{round(score.item(), 3)} at location {box}"
    )
    cv2.rectangle(image, (int(box[0]), int(box[1])), (int(box[2]), int(box[

for idx, row in dataset["apples"]["global_df"].loc[dataset["apples"]["global
    cv2.circle(image, (int(row["c-x"]), int(row["c-y"])), int(row.radius), c

plt.imshow(image)
```

```
Could not find image processor class in the image processor config or the mo
del config. Loading based on pattern matching with the model's feature extra
ctor configuration.
```

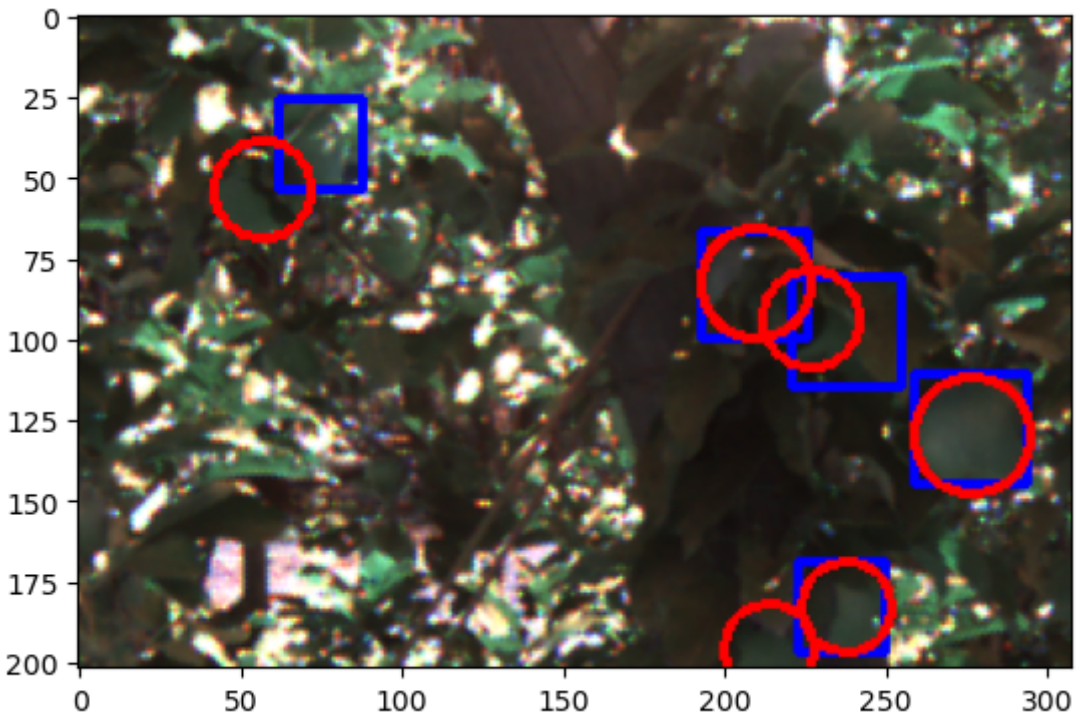
```
YolosObjectDetectionOutput(loss=None, loss_dict=None, logits=tensor([[[-4.57
88,  2.0617],
         [-3.8508,  1.4761],
         [-1.2335, -1.9343],
         [-3.4292,  1.1534],
         [-3.4169,  1.0100],
         [-4.0340,  1.0225],
         [-2.1802, -0.9563],
         [-3.6305,  0.9137],
         [ 1.3971, -4.6544],
         [-4.4231,  3.0229],
         [-3.8521,  1.3301],
         [-3.9636,  1.7187],
         [-3.7593,  1.8635],
         [-3.0155,  0.8553],
         [-4.7436,  1.7102],
         [-4.2922,  1.3717],
         [-4.0870,  1.2082],
         [-4.6371,  1.6775],
         [-3.4885,  1.2046],
         [-4.4516,  2.3954],
         [-3.4768,  1.1695],
         [-5.1585,  1.8812],
         [-3.3357,  1.3112],
         [-5.4039,  2.9642],
         [-1.3846, -1.9272],
         [-4.4956,  2.0685],
         [-4.1236,  2.1797],
         [-4.6194,  2.0136],
         [-4.7092,  1.9141],
         [-3.3894,  1.2455],
         [-3.3569,  1.1214],
         [-3.4329,  1.6017],
         [-3.1185,  0.1401],
         [-4.7326,  2.3146],
         [-3.9160,  1.7209],
         [-4.7309,  3.0007],
         [-4.4591,  1.6461],
         [-4.1821,  1.8762],
         [-4.1644,  1.1239],
         [-4.1266,  2.4334],
         [-3.6950,  1.5368],
         [-3.2686,  1.4055],
         [-3.5014,  1.4201],
         [-1.7376, -1.3768],
         [-2.8144,  0.4644],
         [-2.7023, -0.2912],
         [-3.2588,  0.8229],
         [-3.8927,  1.6010],
         [-3.6113,  1.5084],
         [-2.5324, -0.2540],
         [-3.7670,  0.5775],
         [-5.0875,  2.2792],
         [-3.9936,  1.6239],
         [-4.5861,  1.8041],
         [-3.6151,  0.8794],
         [-4.8762,  1.7230],
         [-2.7269,  0.7425],
         [-5.8057,  2.9964],
         [-4.2735,  1.9256],
         [-3.6668,  1.2580],
         [-3.9230,  1.5420],
         [-4.0059,  1.2749],
         [-2.3398, -0.0828],
```

```
                [-4.7795,  2.4719],
                [-2.5839, -0.1212],
                [-2.9081,  0.4144],
                [-3.8030,  1.5785],
                [-4.5195,  1.4820],
                [-3.6168,  1.7508],
                [-4.3407,  1.6326],
                [-3.0326,  0.5539],
                [-3.4170,  1.5383],
                [-2.8864,  0.5533],
                [-3.1886,  0.7760],
                [-5.4813,  2.9959],
                [-3.7623,  1.7717],
                [-3.7741,  1.1404],
                [-3.4793,  1.4795],
                [-4.5531,  2.4980],
                [-3.9882,  1.0549],
                [-5.0556,  2.3657],
                [-3.3407,  1.1798],
                [-4.7267,  3.0708],
                [-3.4808,  1.6574],
                [-4.0828,  1.8785],
                [-4.4121,  1.6803],
                [-0.4293, -2.7453],
                [-3.2419,  0.4195],
                [-2.5500, -0.0860],
                [-3.4251,  1.0387],
                [-3.5563,  1.3995],
                [-0.6565, -2.1133],
                [-4.6578,  2.0192],
                [-4.6092,  1.9867],
                [-4.1031,  1.3874],
                [-3.2482,  1.5224],
                [-3.0052,  0.8468],
                [-3.8041,  1.6682],
                [-3.4588,  0.5953],
                [-4.9286,  3.3550]]], grad_fn=<ViewBackward0>), pred_boxes=tensor
        ([[[0.2426, 0.7269, 0.0999, 0.1387],
                [0.0234, 0.7811, 0.0464, 0.1286],
                [0.7747, 0.4888, 0.1126, 0.1688],
                [0.8138, 0.8077, 0.1018, 0.1477],
                [0.8451, 0.7649, 0.0878, 0.1260],
                [0.1062, 0.3443, 0.0869, 0.1438],
                [0.7035, 0.0827, 0.1033, 0.1568],
                [0.7162, 0.4911, 0.0894, 0.1465],
                [0.8990, 0.6348, 0.1159, 0.1669],
                [0.4569, 0.1095, 0.1857, 0.2161],
                [0.7075, 0.1048, 0.1035, 0.1552],
                [0.8244, 0.5394, 0.1108, 0.2216],
                [0.9079, 0.6142, 0.0916, 0.1326],
                [0.5761, 0.5191, 0.1223, 0.2233],
                [0.8890, 0.4214, 0.0859, 0.1420],
                [0.3753, 0.2825, 0.0995, 0.1530],
                [0.1123, 0.3631, 0.0764, 0.1378],
                [0.3004, 0.6921, 0.0877, 0.1357],
                [0.8830, 0.8155, 0.0774, 0.1483],
                [0.4852, 0.1221, 0.1240, 0.2037],
                [0.7740, 0.4839, 0.1023, 0.1627],
                [0.3834, 0.6990, 0.0932, 0.1399],
                [0.6900, 0.9483, 0.0902, 0.1005],
                [0.0435, 0.3616, 0.0855, 0.1572],
                [0.2457, 0.2014, 0.0828, 0.1359],
                [0.0268, 0.6307, 0.0532, 0.1338],
                [0.4649, 0.1082, 0.1255, 0.2096],
```

```
[0.6001, 0.0892, 0.1345, 0.1733],
[0.5725, 0.1322, 0.1234, 0.1848],
[0.7471, 0.8956, 0.0924, 0.1385],
[0.7926, 0.5012, 0.1071, 0.1704],
[0.8854, 0.6379, 0.0988, 0.1503],
[0.5174, 0.2878, 0.1026, 0.1628],
[0.0389, 0.6514, 0.0758, 0.1328],
[0.4878, 0.1130, 0.1186, 0.2018],
[0.3306, 0.0318, 0.1231, 0.0643],
[0.2294, 0.5008, 0.0857, 0.1358],
[0.0252, 0.7732, 0.0497, 0.1292],
[0.1212, 0.1588, 0.0816, 0.1397],
[0.4569, 0.0841, 0.1181, 0.1627],
[0.7774, 0.9292, 0.0893, 0.1367],
[0.6914, 0.9513, 0.0865, 0.0948],
[0.6854, 0.4073, 0.1077, 0.1676],
[0.1674, 0.2797, 0.0808, 0.1288],
[0.7906, 0.5041, 0.1083, 0.1693],
[0.3057, 0.3494, 0.0900, 0.1369],
[0.8246, 0.6743, 0.1099, 0.1612],
[0.1470, 0.3151, 0.0931, 0.1363],
[0.2109, 0.2649, 0.1035, 0.1523],
[0.7008, 0.4172, 0.0942, 0.1567],
[0.1060, 0.3671, 0.0774, 0.1391],
[0.9257, 0.3133, 0.0777, 0.1178],
[0.7818, 0.1347, 0.1155, 0.1506],
[0.0328, 0.5724, 0.0658, 0.1369],
[0.5613, 0.6284, 0.0964, 0.1377],
[0.8806, 0.4114, 0.0936, 0.1455],
[0.6860, 0.9481, 0.0840, 0.1009],
[0.0543, 0.0715, 0.1043, 0.1369],
[0.5967, 0.2755, 0.1105, 0.1425],
[0.1697, 0.2868, 0.0775, 0.1294],
[0.2063, 0.1583, 0.0913, 0.1426],
[0.8098, 0.4269, 0.0964, 0.1426],
[0.6046, 0.4047, 0.1018, 0.1512],
[0.9362, 0.4290, 0.1029, 0.2313],
[0.6968, 0.4300, 0.0901, 0.1408],
[0.7989, 0.5301, 0.1116, 0.1855],
[0.7543, 0.4859, 0.0999, 0.1556],
[0.3158, 0.5175, 0.0911, 0.1390],
[0.5710, 0.3752, 0.0851, 0.1221],
[0.8038, 0.4253, 0.0999, 0.1446],
[0.7017, 0.4233, 0.0895, 0.1464],
[0.8764, 0.6329, 0.0944, 0.1450],
[0.7580, 0.8988, 0.0933, 0.1464],
[0.5914, 0.5245, 0.1396, 0.2480],
[0.0354, 0.4951, 0.0699, 0.1838],
[0.2103, 0.2723, 0.0992, 0.1454],
[0.5464, 0.6284, 0.0909, 0.1343],
[0.7160, 0.9001, 0.0850, 0.1397],
[0.3201, 0.0434, 0.1165, 0.0877],
[0.2216, 0.5069, 0.0841, 0.1328],
[0.5518, 0.3491, 0.1469, 0.3176],
[0.4628, 0.1094, 0.1444, 0.2134],
[0.3589, 0.0576, 0.1161, 0.1154],
[0.7112, 0.9318, 0.0934, 0.1324],
[0.1872, 0.8733, 0.1053, 0.1619],
[0.5275, 0.4374, 0.0980, 0.1426],
[0.6809, 0.4152, 0.1080, 0.1621],
[0.5228, 0.2880, 0.1035, 0.1627],
[0.0227, 0.7830, 0.0455, 0.1288],
[0.7386, 0.4840, 0.0943, 0.1474],
[0.2796, 0.2874, 0.0831, 0.1528],
```

```
          [0.7710, 0.9077, 0.0877, 0.1408],
          [0.1001, 0.4678, 0.0724, 0.1187],
          [0.1121, 0.8128, 0.1121, 0.1801],
          [0.5599, 0.3168, 0.1182, 0.1743],
          [0.5716, 0.5082, 0.1055, 0.1858],
          [0.6389, 0.3745, 0.1076, 0.1602],
          [0.2783, 0.3329, 0.0945, 0.1448],
          [0.6044, 0.6078, 0.1018, 0.1561],
          [0.4616, 0.0336, 0.0991, 0.0658]]], grad_fn=<SigmoidBackward0>), au
xiliary_outputs=None, last_hidden_state=tensor([[[ 0.0046, -0.3473,  0.8732,
..., -1.8101, -0.5285, -0.2116],
         [-0.6609, -0.5942,  0.5450,  ...,  0.2801, -1.0389, -1.0751],
         [-0.7305, -0.8830,  0.7319,  ...,  0.7034, -0.1386, -1.3027],
         ...,
         [-0.3601, -0.6965,  0.0193,  ...,  0.3892, -0.4259, -0.5889],
         [-0.3088, -0.7260,  0.2473,  ..., -1.8222, -0.4052,  0.1092],
         [ 0.4998,  0.0846,  1.0905,  ...,  3.8487, -1.2823, -2.0801]]],
       grad_fn=<NativeLayerNormBackward0>), hidden_states=None, attentions=N
one)
Detected apple with confidence 0.668 at location [221.25, 81.7, 255.95, 115.
79]
Detected apple with confidence 0.998 at location [259.04, 111.37, 294.74, 14
5.09]
Detected apple with confidence 0.632 at location [62.94, 26.96, 88.43, 54.4
2]
Detected apple with confidence 0.91 at location [193.09, 67.49, 226.34, 100.
24]
Detected apple with confidence 0.811 at location [223.95, 169.14, 250.96, 19
7.58]
```

Out[88]: <matplotlib.image.AxesImage at 0x7f7311a81b20>



Almonds

In [27]:
```
from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os
feature_extractor = AutoFeatureExtractor.from_pretrained("hustvl/yolos-small
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almond
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almonds/
```

```
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1
```

```
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/feature_extraction_yolos.py:28: FutureWarning: The class YolosFeatureExtra
ctor is deprecated and will be removed in version 5 of Transformers. Please
use YolosImageProcessor instead.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/image_processing_yolos.py:710: FutureWarning: The `max_size` parameter is
deprecated and will be removed in v4.26. Please specify in `size['longest_ed
ge'] instead`.
  warnings.warn(
```

In [28]:
```
from pytorch_lightning import Trainer

# max_steps = 2000
max_steps = 734 # Overfitting Retraining

torch.device("cuda")
model = Yolos(lr=2.5e-5, weight_decay=1e-4)
trainer = Trainer(max_steps=max_steps, gradient_clip_val=0.1, accumulate_gra
trainer.fit(model=model)
```

```
Some weights of YolosForObjectDetection were not initialized from the model
checkpoint at hustvl/yolos-small and are newly initialized because the shape
s did not match:
- class_labels_classifier.layers.2.weight: found shape torch.Size([92, 384])
in the checkpoint and torch.Size([2, 384]) in the model instantiated
- class_labels_classifier.layers.2.bias: found shape torch.Size([92]) in the
checkpoint and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                    | Params
-------------------------------------------------
0 | model | YolosForObjectDetection | 30.7 M
-------------------------------------------------
30.7 M    Trainable params
0         Non-trainable params
30.7 M    Total params
122.600   Total estimated model params size (MB)
Sanity Checking: 0it [00:00, ?it/s]
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
```

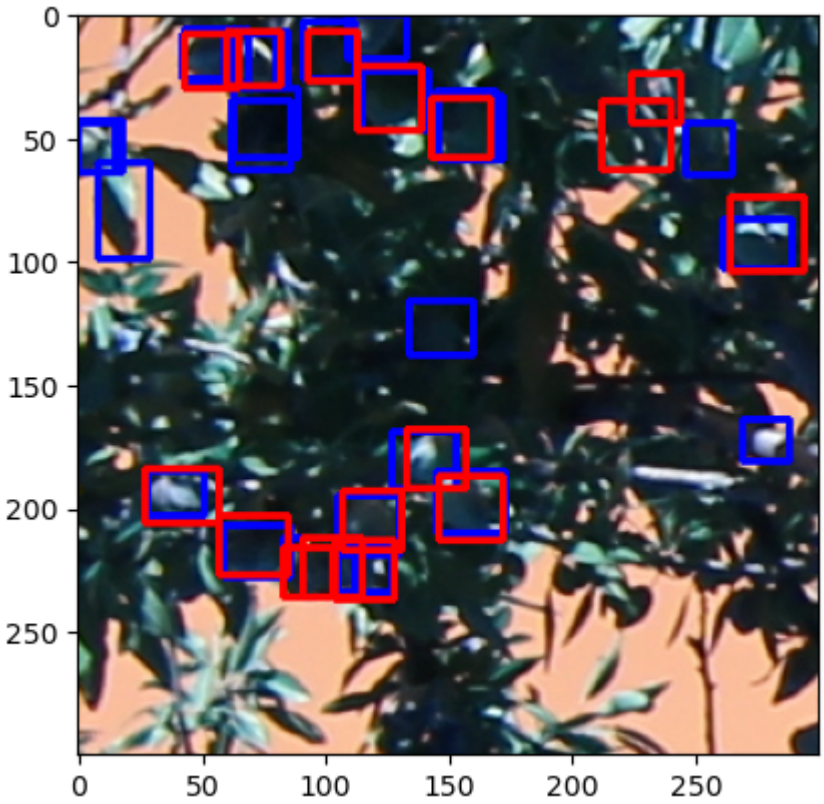
```
`Trainer.fit` stopped: `max_steps=734` reached.
```

In [29]:
```
torch.save(model.state_dict(), "models/yolos_almonds_v2_734.pt")
```

In [33]:
```
from transformers import AutoImageProcessor, AutoModelForObjectDetection
import torch
from PIL import Image
import requests
import numpy as np
import cv2

val_id = np.random.randint(0, len(val_dataset))

file_name = val_dataset.coco.loadImgs(
                                    val_dataset[val_id][1]["image_id"].item(


image = Image.open(os.path.join("datasets/acfr-fruit-dataset/almonds/coco/va
                            file_name
                    ))
image_processor = AutoImageProcessor.from_pretrained("hustvl/yolos-tiny")

cats = train_dataset.coco.cats
id2label = {k: v['name'] for k,v in cats.items()}


pixel_values = val_dataset[val_id][0].unsqueeze(0)

outputs = model(pixel_values=pixel_values)


# convert outputs (bounding boxes and class logits) to COCO API
target_sizes = torch.tensor([image.size[::-1]])
results = image_processor.post_process_object_detection(outputs, threshold=0
     0
]

image = np.array(image)
image = image[:,:,::-1].copy()
for score, label, box in zip(results["scores"], results["labels"], results["
     box = [round(i, 2) for i in box.tolist()]
     print(
         f"Detected {id2label[label.item()]} with confidence "
         f"{round(score.item(), 3)} at location {box}"
     )
     cv2.rectangle(image, (int(box[0]), int(box[1])), (int(box[2]), int(box[

for idx, row in dataset["almonds"]["global_df"].loc[dataset["almonds"]["glob
    cv2.rectangle(image, (int(row.x), int(row.y)), (int(row.x) + int(row.dx)

plt.imshow(image)
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolos/image_processing_yolos.py:824: FutureWarning: The `max_size` parameter is deprecated and will be removed in v4.26. Please specify in `size['longest_edge'] instead`.
  warnings.warn(
Could not find image processor class in the image processor config or the model config. Loading based on pattern matching with the model's feature extractor configuration.
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolos/image_processing_yolos.py:710: FutureWarning: The `max_size` parameter is deprecated and will be removed in v4.26. Please specify in `size['longest_edge'] instead`.
  warnings.warn(
Detected almond with confidence 0.982 at location [83.47, 212.32, 112.66, 236.3]
Detected almond with confidence 0.786 at location [28.91, 185.81, 51.52, 203.79]
Detected almond with confidence 0.979 at location [43.6, 6.97, 69.04, 28.34]
Detected almond with confidence 0.557 at location [246.8, 44.7, 265.72, 65.8]
Detected almond with confidence 0.621 at location [269.43, 164.55, 288.21, 181.25]
Detected almond with confidence 0.996 at location [105.74, 195.32, 130.62, 219.98]
Detected almond with confidence 0.941 at location [105.35, 214.36, 127.04, 234.72]
Detected almond with confidence 0.843 at location [0.5, 43.24, 18.65, 64.42]
Detected almond with confidence 0.78 at location [63.55, 30.81, 89.31, 58.89]
Detected almond with confidence 0.989 at location [127.85, 169.94, 153.67, 192.83]
Detected almond with confidence 0.854 at location [145.71, 31.3, 169.58, 58.03]
Detected almond with confidence 0.954 at location [109.33, 0.52, 133.88, 18.03]
Detected almond with confidence 0.551 at location [62.07, 35.96, 86.3, 63.99]
Detected almond with confidence 0.653 at location [0.36, 44.3, 15.33, 64.29]
Detected almond with confidence 0.542 at location [42.18, 8.04, 62.52, 26.38]
Detected almond with confidence 0.848 at location [8.37, 60.94, 29.32, 99.85]
Detected almond with confidence 0.532 at location [59.35, 8.86, 80.84, 29.99]
Detected almond with confidence 0.997 at location [145.84, 185.74, 173.63, 210.41]
Detected almond with confidence 0.988 at location [261.85, 83.15, 289.05, 103.21]
Detected almond with confidence 0.791 at location [91.28, 3.6, 112.78, 26.5]
Detected almond with confidence 0.991 at location [59.72, 207.99, 86.52, 228.92]
Detected almond with confidence 0.988 at location [115.35, 23.11, 142.54, 47.98]
Detected almond with confidence 0.889 at location [64.4, 7.91, 85.03, 29.08]
Detected almond with confidence 0.922 at location [134.48, 116.26, 160.9, 138.91]
Detected almond with confidence 0.972 at location [146.38, 33.43, 172.4, 59.62]
```

Out[33]:
```
<matplotlib.image.AxesImage at 0x7f4301ce9a60>
```



Mangoes

In [35]:
```
from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os
feature_extractor = AutoFeatureExtractor.from_pretrained("hustvl/yolos-small
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoe
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoes/
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1

from pytorch_lightning import Trainer

# max_steps = 2000
max_steps = 1463 # Overfitting Retraining

torch.device("cuda")
model = Yolos(lr=2.5e-5, weight_decay=1e-4)
trainer = Trainer(max_steps=max_steps, gradient_clip_val=0.1, accumulate_gra
trainer.fit(model=model)
```

```
loading annotations into memory...
Done (t=0.02s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
```

```
Some weights of YolosForObjectDetection were not initialized from the model
checkpoint at hustvl/yolos-small and are newly initialized because the shape
s did not match:
- class_labels_classifier.layers.2.weight: found shape torch.Size([92, 384])
in the checkpoint and torch.Size([2, 384]) in the model instantiated
- class_labels_classifier.layers.2.bias: found shape torch.Size([92]) in the
checkpoint and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name  | Type                      | Params
-------------------------------------------------
0 | model | YolosForObjectDetection | 30.7 M
-------------------------------------------------
30.7 M    Trainable params
0         Non-trainable params
30.7 M    Total params
122.600   Total estimated model params size (MB)
```

```
Sanity Checking: 0it [00:00, ?it/s]
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/traine
r/connectors/data_connector.py:430: PossibleUserWarning: The dataloader, val
_dataloader, does not have many workers which may be a bottleneck. Consider
increasing the value of the `num_workers` argument` (try 4 which is the numb
er of cpus on this machine) in the `DataLoader` init to improve performance.
  rank_zero_warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/pytorch_lightning/traine
r/connectors/data_connector.py:430: PossibleUserWarning: The dataloader, tra
in_dataloader, does not have many workers which may be a bottleneck. Conside
r increasing the value of the `num_workers` argument` (try 4 which is the nu
mber of cpus on this machine) in the `DataLoader` init to improve performanc
e.
  rank_zero_warn(
```

```
Training: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
Validation: 0it [00:00, ?it/s]
```

```
`Trainer.fit` stopped: `max_steps=1463` reached.
```

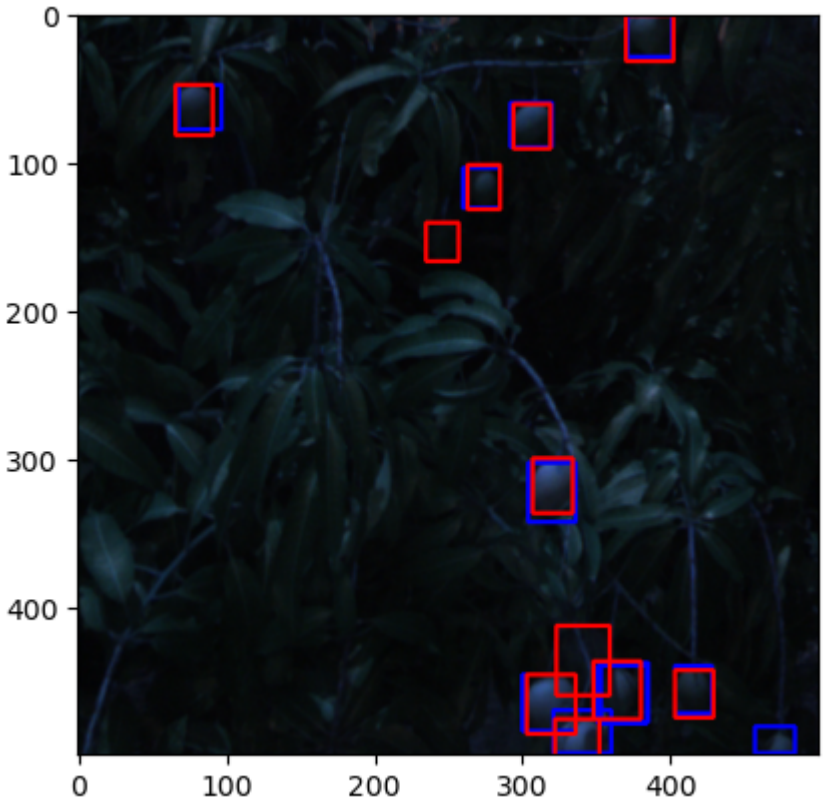
In [36]:
```
torch.save(model.state_dict(), "models/yolos_mangoes_v2_1463.pt")
```

In [38]:
```
from transformers import AutoImageProcessor, AutoModelForObjectDetection
import torch
from PIL import Image
import requests
import numpy as np
import cv2

val_id = np.random.randint(0, len(val_dataset))

file_name = val_dataset.coco.loadImgs(
                                    val_dataset[val_id][1]["image_id"].item(


image = Image.open(os.path.join("datasets/acfr-fruit-dataset/mangoes/coco/va
                            file_name
                    ))
image_processor = AutoImageProcessor.from_pretrained("hustvl/yolos-tiny")
```

```
cats = train_dataset.coco.cats
id2label = {k: v['name'] for k,v in cats.items()}


pixel_values = val_dataset[val_id][0].unsqueeze(0)

outputs = model(pixel_values=pixel_values)


# convert outputs (bounding boxes and class logits) to COCO API
target_sizes = torch.tensor([image.size[::-1]])
results = image_processor.post_process_object_detection(outputs, threshold=0
    0
]

image = np.array(image)
image = image[:,:,::-1].copy()
for score, label, box in zip(results["scores"], results["labels"], results["
    box = [round(i, 2) for i in box.tolist()]
    print(
        f"Detected {id2label[label.item()]} with confidence "
        f"{round(score.item(), 3)} at location {box}"
    )
    cv2.rectangle(image, (int(box[0]), int(box[1])), (int(box[2]), int(box[

for idx, row in dataset["mangoes"]["global_df"].loc[dataset["mangoes"]["glob
    cv2.rectangle(image, (int(row.x), int(row.y)), (int(row.x) + int(row.dx)

plt.imshow(image)
```

```
Could not find image processor class in the image processor config or the mo
del config. Loading based on pattern matching with the model's feature extra
ctor configuration.
Detected mango with confidence 0.873 at location [353.86, 438.43, 385.9, 47
8.63]
Detected mango with confidence 0.551 at location [351.93, 440.94, 384.56, 47
9.43]
Detected mango with confidence 0.984 at location [371.39, 0.38, 401.25, 29.
0]
Detected mango with confidence 0.993 at location [292.0, 60.03, 320.45, 90.0
8]
Detected mango with confidence 0.986 at location [403.18, 440.4, 428.12, 47
2.8]
Detected mango with confidence 0.974 at location [260.64, 104.69, 285.09, 13
1.67]
Detected mango with confidence 0.991 at location [68.58, 48.64, 97.82, 78.1
2]
Detected mango with confidence 0.989 at location [304.46, 303.33, 336.2, 34
3.44]
Detected mango with confidence 0.885 at location [321.71, 470.32, 360.23, 49
9.61]
Detected mango with confidence 0.967 at location [300.66, 446.52, 336.95, 48
4.64]
Detected mango with confidence 0.786 at location [457.8, 481.05, 484.77, 49
9.72]
```

Out[38]: <matplotlib.image.AxesImage at 0x7f42ec21a700>



Evaluation

In the following, all Fine-Tuned checkpoints for both models are evaluated using the COCO Evaluator on their respective validation datasets. The result of each validation run is also saved in the `evaluation_results` directory, for reproducibility. The models are stored in the `models` directory. The evaluation procedure is the same for both YOLOS and DETR respectively.

Yolos

In the following all Fine-Tuned YOLOS checkpoints are evaluated using the COCO Evaluator.

Apples

In order to evaluate a checkpoint, first the dataset, feature extractor and model have to be initialized. Then the Coco evaluator is created by passing it the validation dataset for target data extraction. The model is then set to evaluation mode, disabling gradient calculation and freezing weights. After that, each batch from the validation data loader is passed through the model, generating the corresponding predictions, which are then converted to bounding box coordinates and labels by the featue extractors `post_process` method. The evaluator is then updated with the retrieved predictions. Last but not least, all results are accumulated and a summary is displayed. This procedure is repeated for every model Fine-Tuned checkpoint of YOLOS and DETR on their respective data subsets (Apples, mangoes, almonds).

In [35]:
```
from coco_eval import CocoEvaluator, get_coco_api_from_dataset
from tqdm.notebook import tqdm

from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os

# Initialize Datasets and Feature Extractor
feature_extractor = AutoFeatureExtractor.from_pretrained("hustvl/yolos-small
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples/c
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1

# Initialize Model
model = Yolos(lr=2.5e-5, weight_decay=1e-4)
model.load_state_dict(torch.load("models/yolos_apples_v9.pt")) # load Model


# initialize COCO evaluator with ground truths
base_ds = get_coco_api_from_dataset(val_dataset)
iou_types = ['bbox']
coco_evaluator = CocoEvaluator(base_ds, iou_types)

# set torch device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)
model.eval()

print("Running evaluation...")

for idx, batch in enumerate(tqdm(val_dataloader)):
    # get the inputs
    pixel_values = batch["pixel_values"].to(device)
    labels = [{k: v.to(device) for k, v in t.items()} for t in batch["labels

    # forward pass
    outputs = model.model(pixel_values=pixel_values)

    orig_target_sizes = torch.stack([target["orig_size"] for target in label
    results = feature_extractor.post_process(outputs, orig_target_sizes) # c
    res = {target['image_id'].item(): output for target, output in zip(label
    coco_evaluator.update(res)

coco_evaluator.synchronize_between_processes()
coco_evaluator.accumulate()
coco_evaluator.summarize()
```

```
Running evaluation...
  0%|          | 0/112 [00:00<?, ?it/s]
```

```
Accumulating evaluation results...
DONE (t=0.07s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
473
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.
862
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.
452
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
294
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
528
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.
130
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.
541
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
621
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
543
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
648
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
```

Almonds

In [34]:

```
from coco_eval import CocoEvaluator, get_coco_api_from_dataset
from tqdm.notebook import tqdm


from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os
feature_extractor = AutoFeatureExtractor.from_pretrained("hustvl/yolos-small
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almond
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almonds/
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1

model = Yolos(lr=2.5e-5, weight_decay=1e-4)
model.load_state_dict(torch.load("models/yolos_almonds_v2_734.pt"))


base_ds = get_coco_api_from_dataset(val_dataset)
iou_types = ['bbox']
coco_evaluator = CocoEvaluator(base_ds, iou_types) # initialize evaluator wi

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)
model.eval()

print("Running evaluation...")

for idx, batch in enumerate(tqdm(val_dataloader)):
    # get the inputs
    pixel_values = batch["pixel_values"].to(device)
    labels = [{k: v.to(device) for k, v in t.items()} for t in batch["labels
```

```
    # forward pass
    outputs = model.model(pixel_values=pixel_values)

    orig_target_sizes = torch.stack([target["orig_size"] for target in label
    results = feature_extractor.post_process(outputs, orig_target_sizes) # c
    res = {target['image_id'].item(): output for target, output in zip(label
    coco_evaluator.update(res)

coco_evaluator.synchronize_between_processes()
coco_evaluator.accumulate()
coco_evaluator.summarize()
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/feature_extraction_yolos.py:28: FutureWarning: The class YolosFeatureExtra
ctor is deprecated and will be removed in version 5 of Transformers. Please
use YolosImageProcessor instead.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/image_processing_yolos.py:710: FutureWarning: The `max_size` parameter is
deprecated and will be removed in v4.26. Please specify in `size['longest_ed
ge'] instead`.
  warnings.warn(
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Some weights of YolosForObjectDetection were not initialized from the model
checkpoint at hustvl/yolos-small and are newly initialized because the shape
s did not match:
- class_labels_classifier.layers.2.weight: found shape torch.Size([92, 384])
in the checkpoint and torch.Size([2, 384]) in the model instantiated
- class_labels_classifier.layers.2.bias: found shape torch.Size([92]) in the
checkpoint and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
Running evaluation...
  0%|          | 0/100 [00:00<?, ?it/s]
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/image_processing_yolos.py:824: FutureWarning: The `max_size` parameter is
deprecated and will be removed in v4.26. Please specify in `size['longest_ed
ge'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/image_processing_yolos.py:1167: FutureWarning: `post_process` is deprecate
d and will be removed in v5 of Transformers, please use `post_process_object
_detection`
  warnings.warn(
```

```
Accumulating evaluation results...
DONE (t=0.07s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
309
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.
714
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.
200
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
292
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
454
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.
057
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.
337
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
515
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
506
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
594
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
```

Mangoes

In [39]:

```
from coco_eval import CocoEvaluator, get_coco_api_from_dataset
from tqdm.notebook import tqdm

from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os
feature_extractor = AutoFeatureExtractor.from_pretrained("hustvl/yolos-small
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoe
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoes/
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1

model = Yolos(lr=2.5e-5, weight_decay=1e-4)
model.load_state_dict(torch.load("models/yolos_mangoes_v2_1463.pt"))

base_ds = get_coco_api_from_dataset(val_dataset)
iou_types = ['bbox']
coco_evaluator = CocoEvaluator(base_ds, iou_types) # initialize evaluator wi

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)
model.eval()

print("Running evaluation...")

for idx, batch in enumerate(tqdm(val_dataloader)):
    # get the inputs
    pixel_values = batch["pixel_values"].to(device)
    labels = [{k: v.to(device) for k, v in t.items()} for t in batch["labels

    # forward pass
    outputs = model.model(pixel_values=pixel_values)
```

```
    orig_target_sizes = torch.stack([target["orig_size"] for target in label
    results = feature_extractor.post_process(outputs, orig_target_sizes) # c
    res = {target['image_id'].item(): output for target, output in zip(label
    coco_evaluator.update(res)

coco_evaluator.synchronize_between_processes()
coco_evaluator.accumulate()
coco_evaluator.summarize()
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/feature_extraction_yolos.py:28: FutureWarning: The class YolosFeatureExtra
ctor is deprecated and will be removed in version 5 of Transformers. Please
use YolosImageProcessor instead.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/image_processing_yolos.py:710: FutureWarning: The `max_size` parameter is
deprecated and will be removed in v4.26. Please specify in `size['longest_ed
ge'] instead`.
  warnings.warn(
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Some weights of YolosForObjectDetection were not initialized from the model
checkpoint at hustvl/yolos-small and are newly initialized because the shape
s did not match:
- class_labels_classifier.layers.2.weight: found shape torch.Size([92, 384])
in the checkpoint and torch.Size([2, 384]) in the model instantiated
- class_labels_classifier.layers.2.bias: found shape torch.Size([92]) in the
checkpoint and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
Running evaluation...
  0%|          | 0/250 [00:00<?, ?it/s]
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/image_processing_yolos.py:824: FutureWarning: The `max_size` parameter is
deprecated and will be removed in v4.26. Please specify in `size['longest_ed
ge'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/yolo
s/image_processing_yolos.py:1167: FutureWarning: `post_process` is deprecate
d and will be removed in v5 of Transformers, please use `post_process_object
_detection`
  warnings.warn(
```

```
Accumulating evaluation results...
DONE (t=0.16s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
402
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.
829
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.
326
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
280
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
461
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.
113
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.
472
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
549
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
482
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
583
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
```

Detr

Apples

In [59]:

```
from coco_eval import CocoEvaluator, get_coco_api_from_dataset
from tqdm.notebook import tqdm

from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os
feature_extractor = DetrImageProcessor.from_pretrained("facebook/detr-resnet
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/apples/c
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1

model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)
model.load_state_dict(torch.load("models/detr_apples_v1.pt"))


base_ds = get_coco_api_from_dataset(val_dataset)
iou_types = ['bbox']
coco_evaluator = CocoEvaluator(base_ds, iou_types) # initialize evaluator wi

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)
model.eval()

print("Running evaluation...")

for idx, batch in enumerate(tqdm(val_dataloader)):
    # get the inputs
    pixel_values = batch["pixel_values"].to(device)
```

```
    labels = [{k: v.to(device) for k, v in t.items()} for t in batch["labels
    # forward pass
    outputs = model.model(pixel_values=pixel_values)
    orig_target_sizes = torch.stack([target["orig_size"] for target in label
    results = feature_extractor.post_process(outputs, orig_target_sizes) # c
    res = {target['image_id'].item(): output for target, output in zip(label
    coco_evaluator.update(res)

coco_evaluator.synchronize_between_processes()
coco_evaluator.accumulate()
coco_evaluator.summarize()
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:780: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Some weights of DetrForObjectDetection were not initialized from the model c
heckpoint at facebook/detr-resnet-50 and are newly initialized because the s
hapes did not match:
- class_labels_classifier.weight: found shape torch.Size([92, 256]) in the c
heckpoint and torch.Size([2, 256]) in the model instantiated
- class_labels_classifier.bias: found shape torch.Size([92]) in the checkpoi
nt and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
Running evaluation...
  0%|          | 0/112 [00:00<?, ?it/s]
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:1266: FutureWarning: `post_process` is deprecated
and will be removed in v5 of Transformers, please use `post_process_object_d
etection`
  warnings.warn(
```

```
Accumulating evaluation results...
DONE (t=0.08s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
406
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.
760
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.
386
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
217
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
471
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.
129
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.
480
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
544
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
424
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
586
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
```

Almonds

In [69]:

```
from coco_eval import CocoEvaluator, get_coco_api_from_dataset
from tqdm.notebook import tqdm

from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os
feature_extractor = DetrImageProcessor.from_pretrained("facebook/detr-resnet
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almond
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/almonds/
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1

model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)
model.load_state_dict(torch.load("models/detr_almonds_v1.pt"))


base_ds = get_coco_api_from_dataset(val_dataset)
iou_types = ['bbox']
coco_evaluator = CocoEvaluator(base_ds, iou_types) # initialize evaluator wi

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)
model.eval()

print("Running evaluation...")

for idx, batch in enumerate(tqdm(val_dataloader)):
    # get the inputs
    pixel_values = batch["pixel_values"].to(device)
    labels = [{k: v.to(device) for k, v in t.items()} for t in batch["labels

    # forward pass
```

```
    outputs = model.model(pixel_values=pixel_values)

    orig_target_sizes = torch.stack([target["orig_size"] for target in label
    results = feature_extractor.post_process(outputs, orig_target_sizes) # c
    res = {target['image_id'].item(): output for target, output in zip(label
    coco_evaluator.update(res)

coco_evaluator.synchronize_between_processes()
coco_evaluator.accumulate()
coco_evaluator.summarize()
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:780: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Some weights of DetrForObjectDetection were not initialized from the model c
heckpoint at facebook/detr-resnet-50 and are newly initialized because the s
hapes did not match:
- class_labels_classifier.weight: found shape torch.Size([92, 256]) in the c
heckpoint and torch.Size([2, 256]) in the model instantiated
- class_labels_classifier.bias: found shape torch.Size([92]) in the checkpoi
nt and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
Running evaluation...
  0%|          | 0/100 [00:00<?, ?it/s]
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:1266: FutureWarning: `post_process` is deprecated
and will be removed in v5 of Transformers, please use `post_process_object_d
etection`
  warnings.warn(
```

```
Accumulating evaluation results...
DONE (t=0.07s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
274
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.
640
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.
173
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
260
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
407
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.
052
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.
314
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
446
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
438
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
522
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
```

Mangoes

In [70]:

```
from coco_eval import CocoEvaluator, get_coco_api_from_dataset
from tqdm.notebook import tqdm

from transformers import AutoFeatureExtractor
from torch.utils.data import DataLoader
import os
feature_extractor = DetrImageProcessor.from_pretrained("facebook/detr-resnet
train_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoe
val_dataset = CocoDetection(img_folder="datasets/acfr-fruit-dataset/mangoes/
train_dataloader = DataLoader(train_dataset, collate_fn=collate_fn, batch_si
val_dataloader = DataLoader(val_dataset, collate_fn=collate_fn, batch_size=1

model = Detr(lr=1e-4, lr_backbone=1e-5, weight_decay=1e-4)
model.load_state_dict(torch.load("models/detr_mangoes_v1.pt"))


base_ds = get_coco_api_from_dataset(val_dataset)
iou_types = ['bbox']
coco_evaluator = CocoEvaluator(base_ds, iou_types) # initialize evaluator wi

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)
model.eval()

print("Running evaluation...")

for idx, batch in enumerate(tqdm(val_dataloader)):
    # get the inputs
    pixel_values = batch["pixel_values"].to(device)
    labels = [{k: v.to(device) for k, v in t.items()} for t in batch["labels

    # forward pass
```

```
    outputs = model.model(pixel_values=pixel_values)

    orig_target_sizes = torch.stack([target["orig_size"] for target in label
    results = feature_extractor.post_process(outputs, orig_target_sizes) # c
    res = {target['image_id'].item(): output for target, output in zip(label
    coco_evaluator.update(res)

coco_evaluator.synchronize_between_processes()
coco_evaluator.accumulate()
coco_evaluator.summarize()
```

```
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:780: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!
Some weights of DetrForObjectDetection were not initialized from the model c
heckpoint at facebook/detr-resnet-50 and are newly initialized because the s
hapes did not match:
- class_labels_classifier.weight: found shape torch.Size([92, 256]) in the c
heckpoint and torch.Size([2, 256]) in the model instantiated
- class_labels_classifier.bias: found shape torch.Size([92]) in the checkpoi
nt and torch.Size([2]) in the model instantiated
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
Running evaluation...
  0%|          | 0/250 [00:00<?, ?it/s]
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:886: FutureWarning: The `max_size` parameter is d
eprecated and will be removed in v4.26. Please specify in `size['longest_edg
e'] instead`.
  warnings.warn(
/opt/conda/envs/pytorch/lib/python3.9/site-packages/transformers/models/det
r/image_processing_detr.py:1266: FutureWarning: `post_process` is deprecated
and will be removed in v5 of Transformers, please use `post_process_object_d
etection`
  warnings.warn(
```

```
Accumulating evaluation results...
DONE (t=0.16s).
IoU metric: bbox
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
353
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.
733
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.
289
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
224
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
410
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.
105
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.
432
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.
509
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.
450
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.
539
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -
1.000
```

Results

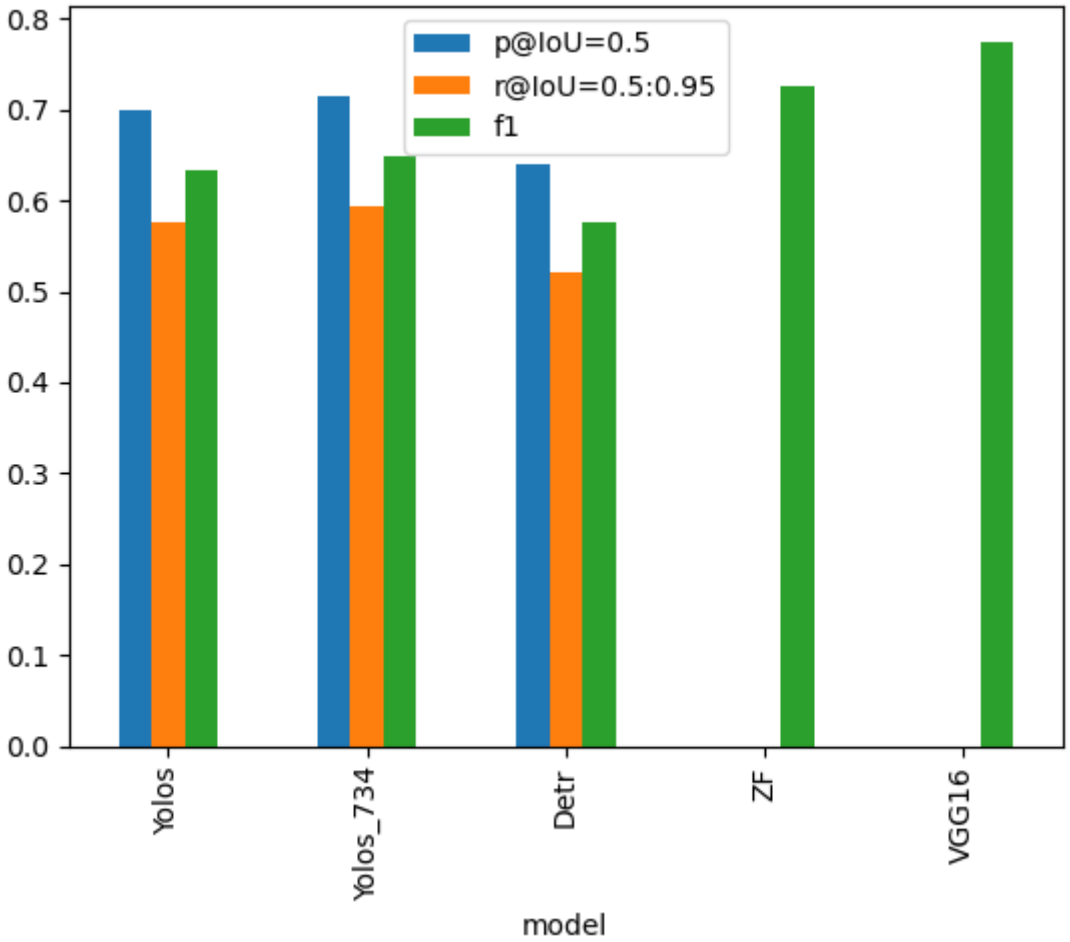
In the following chapter, the results of Experiments on applying YOLOS and DETR on the three object detection tasks are being presented and discussed. As the metric of choice for comparison, the F1-score is used. It is computed from the Intersection-over-Union in the $[0.5, 1]$-Interval for a maximum of 100 Detections and the Recall for the Intersection-over-Union in the $[0.5, 0.95]$-Interval, returned by the COCO evaluator. As a baseline for comparing performance, two Faster-R-CNNs, ZFnet and VGG16 and a pixel-wise CNN which have been applied by the authors of the dataset on the dataset, are used. Each sub-task (Apples, Almonds and Mangoes) is discussed seperately.

The following Figure depicts the results on the Almond sub-task. Both Vision Transformer models showed significantly worse performance than both Faster-R-CNN baselines, achieving around 0.1 less on the F1 score, with the primary bottleneck being the models recall, as it is smaller by over 0.1 compared to precision. Thus, both models have significant problems with being able to detect every fruit on the picture, while the detections in general are quite accurate with precision scores around 0.7. The best scores of all three training checkpoints have been achieved by the YOLOS model with early stopping before overfitting behaviour occurred, with the YOLOS model beating out DETR in general on this sub-task. Unfortunately, Bargoti & Underwood did not release precision and recall scores, such that performance can't be compared more specificely between the baselines and the models. The pixel-wise CNN has also not been tested on the almond task by the authors (Bargoti & Underwood, 2017).

In [19]:
```
# almonds
p = [0.7, 0.714, 0.640]
r = [0.577, 0.594, 0.522]
```

```
models = ["Yolos", "Yolos_734", "Detr"]
df = pd.DataFrame(
    {
        "p@IoU=0.5": p,
        "r@IoU=0.5:0.95": r,
        "model": models
    })
df["f1"] = 2*((df["p@IoU=0.5"] * df["r@IoU=0.5:0.95"]) / (df["p@IoU=0.5"] +
baselines = pd.DataFrame({
    "p@IoU=0.5": [0,0],
    "r@IoU=0.5:0.95": [0,0],
    "f1": [0.726, 0.775],
    "model": ["ZF", "VGG16"]
})
df = pd.concat([df, baselines])
df.set_index(df.model, inplace=True)
df.plot.bar()
```

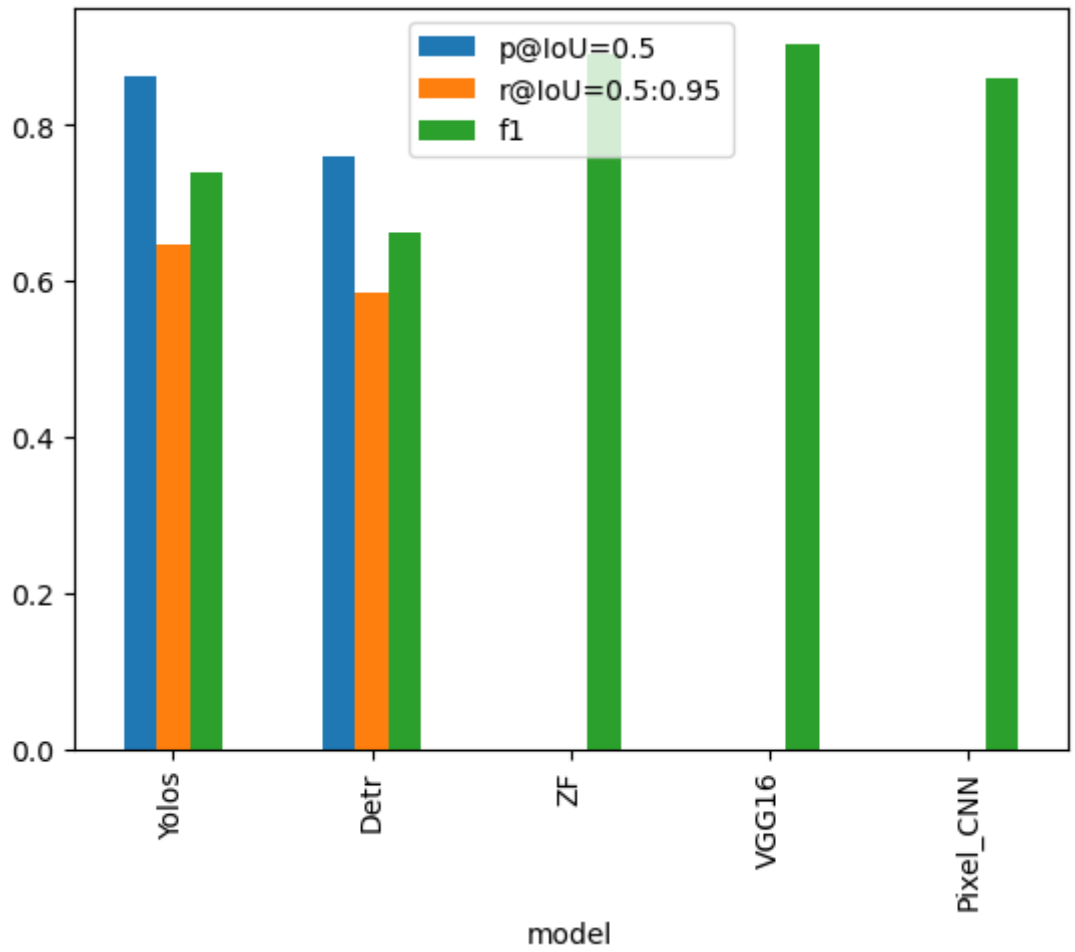
Out[19]: <AxesSubplot: xlabel='model'>



The results on the apple detection sub-task, depicted in the Figure below, show a similar result. Both Vision Transformers are loosing out on overall performance, compared to both Faster-R-CNN derivatives, and the pixel-wise CNN. This time the performance margin between the baselines and the models tested is even larger with performance differing by about 0.15 to 0.2. Again with the biggest bottleneck being the recall of both trained models, showing that again Vision Transformers have problems with detecting all samples of fruit on the image. In this setting YOLOS was again able to outperform DETR.

In [20]:
```
# apples
p = [0.862, 0.760]
r = [0.648, 0.586]
models = ["Yolos", "Detr"]
df = pd.DataFrame(
    {
        "p@IoU=0.5": p,
        "r@IoU=0.5:0.95": r,
        "model": models
    })
df["f1"] = 2*((df["p@IoU=0.5"] * df["r@IoU=0.5:0.95"]) / (df["p@IoU=0.5"] +
baselines = pd.DataFrame({
    "p@IoU=0.5": [0,0, 0],
    "r@IoU=0.5:0.95": [0,0, 0],
    "f1": [0.892, 0.904, 0.861],
    "model": ["ZF", "VGG16", "Pixel_CNN"]
})
df = pd.concat([df, baselines])
df.set_index(df.model, inplace=True)
df.plot.bar()
```

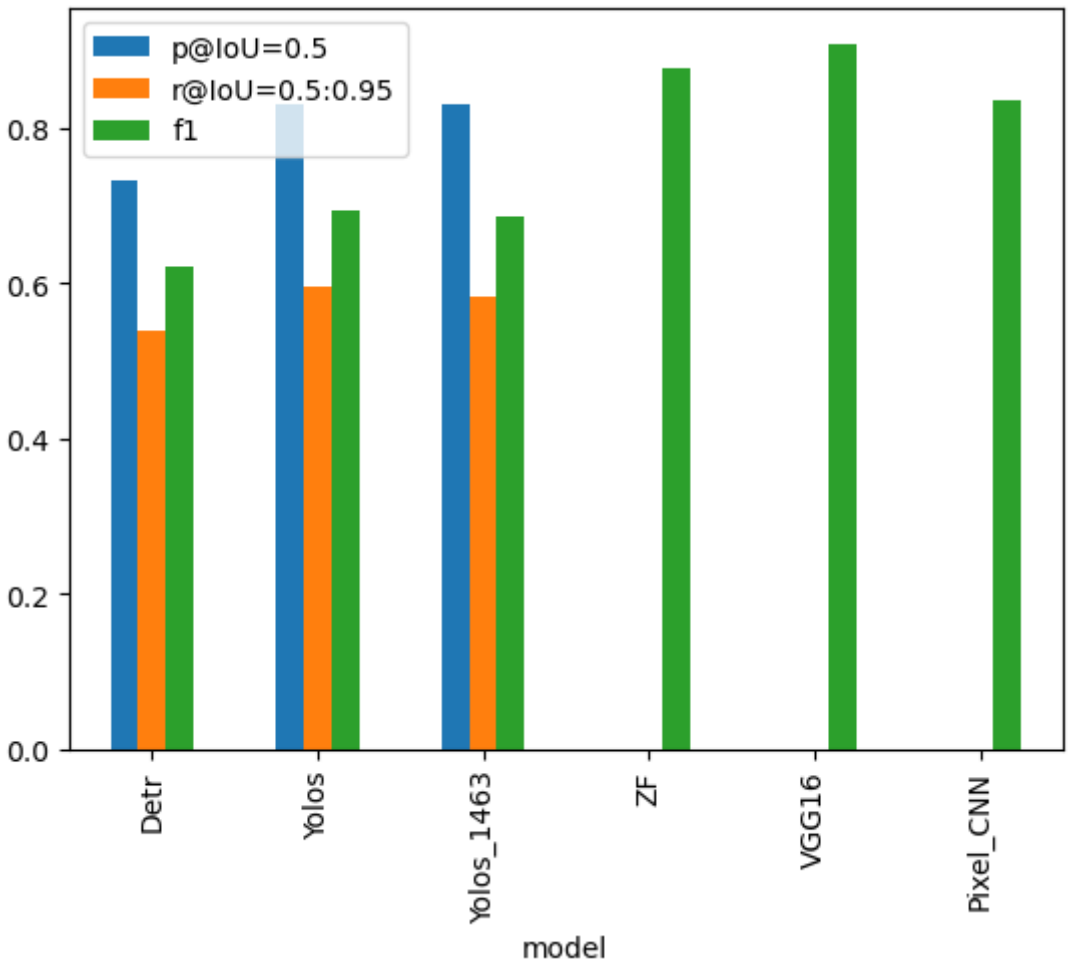
Out[20]:
```
<AxesSubplot: xlabel='model'>
```



For Mangoes, as depicted in the Figure below, the performance pattern observed so far is repeated. All model checkpoints loose out compared to the Faster-R-CNN baselines with a margin about 0.2 on the F1-score. Again all three vision transformer checkpoints and both models have a low recall under 0.6, while achieving competitive performance in terms of precision. This especially holds for both YOLOS checkpoints, where the precision score is in close proximity to the Faster-R-CNN F1-Scores. DETR on the other hand was again the worst performing model across precision, recall and F1.

In [21]:
```
# Mangoes
p = [0.733, 0.831, 0.829]
r = [ 0.539, 0.596, 0.583]
models = ["Detr", "Yolos", "Yolos_1463"]
df = pd.DataFrame(
    {
        "p@IoU=0.5": p,
        "r@IoU=0.5:0.95": r,
        "model": models
    })
df["f1"] = 2*((df["p@IoU=0.5"] * df["r@IoU=0.5:0.95"]) / (df["p@IoU=0.5"] +
baselines = pd.DataFrame({
    "p@IoU=0.5": [0,0, 0],
    "r@IoU=0.5:0.95": [0,0, 0],
    "f1": [0.876, 0.908, 0.836],
    "model": ["ZF", "VGG16", "Pixel_CNN"]
})
df = pd.concat([df, baselines])
df.set_index(df.model, inplace=True)
df.plot.bar()
```

Out[21]:
```
<AxesSubplot: xlabel='model'>
```



As both Vision Transformers were significatnly outperformed by the Faster-R-CNN baselines, the question arises as to why this happens, even though multiple sources were able to show similar or even better performance of Vision Transformers compared to Faster-R-CNNs on popular computer vision benchmarks (Carion et al., 2020; Dosovitskiy et al., 2021; Fang et al., 2021; Kumar & R, 2022; Uparkar et al., 2023). One reason, is that the task of detecting many *small* fruits does not suit Vision Transformers architecture really well. In NLP, it is important to be able to process / attend to each word

in a global context over the whole sequence, which is exactly what Attention and Multi-Head attention achieves. In Multi-Head attention, a token embedding is processed / attended in regard to different aspects, or more formally, in regard to different representation subspaces in parrallel by multiple attention heads. This allows to compute a strong semantic representation of a word or sequence depending on the task. These circumstance remain unchanged with regards to Vision Transformers, with the only difference being the semantics behind the numeric representation. This time, the model does not attend to different representation subspaces of a word, but to different representation subspaces of a patch embedding for the original VIT and YOLOS and for DETR it attends to different extracted features. However, this still happens in a *global* context and not locally. Thus, the result that is computed for every object query or detection token is directly influenced by the whole picture, introducing noise into the computation for small objects, as their activation / attention heat map area is small compared to larger objects. Thus Vision Transformers have been shown to demonstrate significantly better performance on large objects, because of their non-locality, while at the same time achieving lower performances on small objects (Carion et al., 2020). In addition, they only work with fixed learned position specific detection tokens. This means that they can only detect a maximum of N objects near N fixed positions. As fruits, especially almonds are quite small and only fill a very small portion of the image per object, with a lot of them being on the image and often clustered, these circumstances pose a real challenge for the Vision Transformers architecture. Two-Stage detectors such as Faster-R-CNN, as well as Single-Stage detectors like YOLO don't face these challenges, as they don't work on a global context. For both systems, detections are determined using some form of region proposal and some form of image classifier. Faster-R-CNN first predicts bounding boxes and then classify the object. YOLO acts in the same way, just that both steps happen simultaneously on small grid cells over the image (Redmon et al., 2016; Ren et al., 2016). With that, both systems only use the *local* context of the bounding box for image classification and bounding box prediction and can thus extract better features and achieve a higher performance on these datasets during these experiments. However, despite their significant performance loss on this task, they also have some advantages. They are easy to implement, highly adaptable to new architectures and can be trained in a fraction of the time compared to a Faster-R-CNN (Carion et al., 2020; Fang et al., 2021). This allows rapid prototyping and research in this field. In addition, the vision transformer concept is still in very early stages and a lot of potential has already been shown (Carion et al., 2020; Fang et al., 2021; Kumar & R, 2022; Uparkar et al., 2023), thus vision transformers can't yet be written of as unsuitable for this kind of detection task.

However, detection performance is not everything that matters for fruit detection in orchards. The size and real-time performance of the model are equally important, as they have to be able to work on small edge AI systems such as the Nvidia Jetson NX (Lyu et al., 2022). Therefore, the average inference time per frame/image, along with the model size is compared to current state-of-the-art fruit detection systems in orchards in the table below. For YOLOS and DETR the average time per image is computed over the validation dataset. In terms of size, they are quite competitive against the Faster-R-CNN derivatives (VGG16 & ZFnet), as well as YOLO BP, while at the same time being significantly larger than the custom YOLOV5 models for dragon fruit and green citrus

detection. In terms of inference time, both transformers trade blows with the VGG16 network but get outperformed significantly by all YOLO derivatives. This becomes even more clear by looking at the amount of frames per second (fps), the different models were able to process. The YOLOV5 models achieve almost 60fps which is double the frame rate of movies. Even on the small edge AI system, the YOLO system was still able to achieve more than the movie frame rate of 24fps. Both Vision Transformer can't come close to this number with only being able to achieve 6-8fps respectively on a Server GPU. Thus, in order to achieve practical relevance in the field of fruit detection in orchards, Vision Transformers not only have to become better at detecting small objects, they also have to get significantly smaller to keep up with the performance of small YOLO based detectors.

In [1]:
```
import pandas as pd
# Inference Times
df = pd.DataFrame({
    "ms/frame": [116, 148, 19.5, 17, 37, 55, 130, 40, 2500],
    "Parameters (M)": [41.5, 30.7, 5.2, 7.9, 7.9, 60,136, 60, None],
    "model": ["Detr", "Yolos", "YoloV5s-Dragon-Fruit", "Yolov5-CS-Server", "
})
df["fps"] = (1000 / df["ms/frame"]).astype(int)
df.set_index("model", inplace=True)
df
```

Out[1]:

model	ms/frame	Parameters (M)	fps
Detr	116.0	41.5	8
Yolos	148.0	30.7	6
YoloV5s-Dragon-Fruit	19.5	5.2	51
Yolov5-CS-Server	17.0	7.9	58
Yolov5-CS-JetsonNX	37.0	7.9	27
Yolo BP	55.0	60.0	18
VGG16	130.0	136.0	7
ZF	40.0	60.0	25
Pixel-Wise CNN	2500.0	NaN	0

Conclusion

In this paper, two Vision Transformer for object detection have been adapted to the task of fruit detection using the acfr-fruit-dataset. First, the general characteristics and the architecture of Faster-R-CNNs, YOLO based single stage detectors and Vision Transformers have been outlined. Then the used YOLOS and DETR models have been introduced and trained on the down-stream task of fruit detection on the three sub-task of the dataset: Apple, Mango and Almond detection. Then all trained checkpoints are evaluated using the COCO evaluator and performance was compared in terms of the F1 score against ZFnet, VGG16 and Faster-R-CNN. Both Transformers showed inferior performance compared to the established baselines by the authors of the dataset, with recall being the biggest bottleneck across the board. After further analysis of the

models' architecture, it became clear, that the models non locality cripples the detection performance of smaller objects such as fruits, resulting in the performance deficit. In addition to that, vision transformers also were inferior in terms of real-time inference capabilities, loosing out to current state of the art YOLOv5 based fruit detection systems in terms of model size, as well as achievable inference frame rate. They were only able to outperform the VGG16 model in terms of framerate but being smaller than all Faster-R-CNN counterparts. Thus, in order to become relevant for the fruit industry and precision farming methods, additional research is needed. The detection performance for small objects has to be improved, the models have to shrink in size and complexity and achieve faster inference times. Despite their performance losses, they proved to be fast to train with training only taking around 30 minutes for all models trained on an Nvidia Tesla T4 GPU. As they are still quite new, with YOLOS being proposed in late 2021 and fruit detection being an active research area, a lot of potential is still to be gained from applying and adapting the concept of using transformers for computer vision and thus research should continue on improving vision transformers and start to adopt existing concepts for industry relevant tasks such as fruit detection.

Bibliography

- Bargoti, S., & Underwood, J. (2017). Deep Fruit Detection in Orchards (arXiv:1610.03677). arXiv. https://doi.org/10.48550/arXiv.1610.03677
- Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., & Zagoruyko, S. (2020, May 26). End-to-End Object Detection with Transformers. ArXiv.Org. https://arxiv.org/abs/2005.12872v3
- Cuenat, S., & Couturier, R. (2022). Convolutional Neural Network (CNN) vs Vision Transformer (ViT) for Digital Holography (arXiv:2108.09147). arXiv. http://arxiv.org/abs/2108.09147
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (arXiv:2010.11929). arXiv. http://arxiv.org/abs/2010.11929
- Fang, Y., Liao, B., Wang, X., Fang, J., Qi, J., Wu, R., Niu, J., & Liu, W. (2021). You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection (arXiv:2106.00666). arXiv. http://arxiv.org/abs/2106.00666
- Gemtos, T., Fountas, S., Tagarakis, A., & Liakos, V. (2013). Precision Agriculture Application in Fruit Crops: Experience in Handpicked Fruits. Procedia Technology, 8, 324–332. https://doi.org/10.1016/j.protcy.2013.11.043
- Image Processor. (n.d.). Retrieved April 1, 2023, from https://huggingface.co/docs/transformers/main_classes/image_processor
- Koirala, A., Walsh, K. B., Wang, Z., & McCarthy, C. (2019). Deep learning for real-time fruit detection and orchard fruit load estimation: Benchmarking of 'MangoYOLO.' Precision Agriculture, 20(6), 1107–1135. https://doi.org/10.1007/s11119-019-09642-0
- Korstanje, J. (2021, August 31). The F1 score. Medium. https://towardsdatascience.com/the-f1-score-bec2bbc38aa6

- Kumar, T., & R, S. (2022). Vision Transformer based System for Fruit Quality Evaluation [Preprint]. In Review. https://doi.org/10.21203/rs.3.rs-1526586/v1
- Lyu, S., Li, R., Zhao, Y., Li, Z., Fan, R., & Liu, S. (2022). Green Citrus Detection and Counting in Orchards Based on YOLOv5-CS and AI Edge System. Sensors, 22(2), Article 2. https://doi.org/10.3390/s22020576
- Miltenburg, K. (n.d.). Implementing Precision Agriculture in Dutch apple orchards.
- Mirhaji, H., Soleymani, M., Asakereh, A., & Abdanan Mehdizadeh, S. (2021). Fruit detection and load estimation of an orange orchard using the YOLO models through simple approaches in different imaging and illumination conditions. Computers and Electronics in Agriculture, 191, 106533. https://doi.org/10.1016/j.compag.2021.106533
- OECD-FAO Agricultural Outlook. (2021). [Data set]. OECD Publishing. https://doi.org/10.1787/agr-outl-data-en
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection (arXiv:1506.02640). arXiv. http://arxiv.org/abs/1506.02640
- Ren, S., He, K., Girshick, R., & Sun, J. (2016). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (arXiv:1506.01497). arXiv. http://arxiv.org/abs/1506.01497
- Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition (arXiv:1409.1556). arXiv. http://arxiv.org/abs/1409.1556
- Uparkar, O., Bharti, J., Pateriya, R. K., Gupta, R. K., & Sharma, A. (2023). Vision Transformer Outperforms Deep Convolutional Neural Network-based Model in Classifying X-ray Images. Procedia Computer Science, 218, 2338–2349. https://doi.org/10.1016/j.procs.2023.01.209
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need (arXiv:1706.03762). arXiv. https://doi.org/10.48550/arXiv.1706.03762
- Zeiler, M. D., & Fergus, R. (2013). Visualizing and Understanding Convolutional Networks (arXiv:1311.2901). arXiv. http://arxiv.org/abs/1311.2901
- Zhang, B., Wang, R., Zhang, H., Yin, C., Xia, Y., Fu, M., & Fu, W. (2022). Dragon fruit detection in natural orchard environment by integrating lightweight network and attention mechanism. Frontiers in Plant Science, 13, 1040923. https://doi.org/10.3389/fpls.2022.1040923
- Zheng, Z., Xiong, J., Lin, H., Han, Y., Sun, B., Xie, Z., Yang, Z., & Wang, C. (2021). A Method of Green Citrus Detection in Natural Environments Using a Deep Convolutional Neural Network. Frontiers in Plant Science, 12, 705737. https://doi.org/10.3389/fpls.2021.705737