On Shelf Cold Drinks Detection Using Detectron2 👁️

This Notebook goes through the implementation of the state of art custom object detection model (detectron2) for detecting cold drinks (focused on `Fanta`) on shelves in Markets and Super stores.

In other words, using computer vision to detect products(fanta and other cold drinks in our case) in pictures of shelves.
As shown in image below 👇

Detectron-2 IN ACTION

Problem Definition

This is purely a classification Problem and we have to classify cold drinks on the shelves. In our Case we have about 200 images of store shelves and we assigned the products in these images 11 classes. Here are the classes we wanna deal with:

1. Fanta 1.5L Bottle
2. Fanta 500mL Bottle
3. Fanta Can
4. Fanta Carton-Can
5. Fanta Carton-Bottles
6. Coca Cola Bottle
7. Sprite Bottle
8. 7up Bottle
9. Marinda Bottle
10. Dew Bottle
11. Pepsi Bottle

It can be seen that I have special focus on `Fanta` as it was the specific category assigned to me for project. But also we need to identify other major products too.

Why It is Useful?

There are a lot of applications of the project and few are discussed below:

Auditing Product Placement — Planogram Compliance:

Customers make important buying decisions at store shelves. CPG companies invest in tools and studies to create planograms that are part of their ideal store strategy. Auditing shelf management using Image Recognition digitizes store checks and plays an essential role in understanding the shelf conditions and how they affect the sales of their core SKUs. Object Detection, using a deep neural network, detects SKUs within images of shelves and classifies them at manufacturer, brand or SKU level. A Detectron-2 can be trained to compare and identify gaps between actual shelf image and reference planogram. Using this Detectron-2 model , the auditor can get real-time feedback on their handheld device to take corrective actions while they are still in store. Leveraging object detection in retail can make store checks productive and empower your sales representative or auditor to take corrective actions and fix issues.

Detecting Empty Shelf:

A study conducted by IHL group puts the total loss of sales due to out of stock at nearly $1 trillion. The study found that over 20% of Amazon's North American Retail revenue can be attributed to consumers who first tried to buy the product at a local store but found it out-of-stock. Empty shelves were encountered by 32% of the shoppers in the study and thus, retailers and CPG firms need to ensure the shelf is always stocked with the correct product. Retailers can leverage Image Recognition and Object Detection using fixed cameras in their store to alert the store staff whenever a shelf is found to be empty. The store staff can get an instant alert on their phone along with the accurate description of the SKU missing from the shelf and the location of the shelf.

Assessing Competition:

Detectron-2 can also be trained to identify competitive activity in-store and spot category trends.Identifying competitors and tracking their in-store activity can help a CPG brand to grow in its category by identifying potential opportunities.

Real Time Inventory Assessment:

Detectron-2 can be trained to do this job for us as it is the biggest problem to maintain the inventory levels. Using this real time estimate of the products on the shelves can be made and this can help optimizing the inventory management.

What defines Success for me?

I have a target to achieve 90%+ accuracy in detecting the allocated/assigned products. This is the least criteria but by optimizing the whole process again and again I have to achieve 95%+

accuracy in detection of each product belonging to my categories. At the end of each loop of this whole notebook work process, I will evaluate the performance and repeat it untill the desired goal.

Major Tools and Techs used in Project

Some of the major tools and packages used in this Project are following:

1. Python
2. Detectron2
3. Numpy
4. Pandas
5. PyTorch
6. TorchVision
7. OpenCV

Lets Start The Implementation

The cells next from here will implement the project. Expand the cells to see the implementation.

This notebook is completly reuseble for any related project so any body can follow this notebook for his/her own project with his/her own Data Set.

Downloading/Installing the dependencies

Detectron2 has following requirements and required to install/download in order to make it work.

1. Linux or macOS with Python ≥ 3.6 (already available in collab)
2. PyTorch ≥ 1.4 and torchvision that matches the PyTorch installation.
3. OpenCV (Its optional and also already available)

Note: Detectron-2 needs GPU to run and fortunately collab provides us with it.

```
#installing Pytorch and torchvision from source
!pip install -U torch torchvision
!pip install git+https://github.com/facebookresearch/fvcore.git
```

```
Requirement already up-to-date: torch in /usr/local/lib/python3.6/dist-packag
Requirement already up-to-date: torchvision in /usr/local/lib/python3.6/dist-
Requirement already satisfied, skipping upgrade: numpy in /usr/local/lib/pyth
Requirement already satisfied, skipping upgrade: future in /usr/local/lib/pyt
Requirement already satisfied, skipping upgrade: pillow>=4.1.1 in /usr/local/
Collecting git+https://github.com/facebookresearch/fvcore.git
  Cloning https://github.com/facebookresearch/fvcore.git to /tmp/pip-req-buil
  Running command git clone -q https://github.com/facebookresearch/fvcore.git
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-package
Collecting yacs>=0.1.6
  Downloading https://files.pythonhosted.org/packages/38/4f/fe9a4d472aa867878
Collecting pyyaml>=5.1
  Downloading https://files.pythonhosted.org/packages/64/c2/b80047c7ac2478f95
     |████████████████████████████████| 276kB 8.4MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
Collecting portalocker
  Downloading https://files.pythonhosted.org/packages/89/a6/3814b7107e0788040
Requirement already satisfied: termcolor>=1.1 in /usr/local/lib/python3.6/dis
Requirement already satisfied: Pillow in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: tabulate in /usr/local/lib/python3.6/dist-pack
Building wheels for collected packages: fvcore, pyyaml
  Building wheel for fvcore (setup.py) ... done
  Created wheel for fvcore: filename=fvcore-0.1.2-cp36-none-any.whl size=4785
  Stored in directory: /tmp/pip-ephem-wheel-cache-wsy_jw39/wheels/48/53/79/3c
```

```
# Checking availability of GPU and required OS
import torch, torchvision
print("Yesss !!! GPU is Enabled." if torch.cuda.is_available() else "Opss! GPU is
print("\nFollowing are the details of OS:\n")
!gcc --version
```

```
Yesss !!! GPU is Enabled.

Following are the details of OS:

gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0
Copyright (C) 2017 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

```
#checking python and opencv version (to confirm that they are already installed)
import cv2
!python --version
print(f"OpenCV Version : {cv2.__version__}")
```

```
Python 3.6.9
OpenCV Version : 4.1.2
```

Installing Detectron2

After successful installation of dependencies for detectron2, now we are able to install detectron2. There are 2 ways to install Detectron2:

1. Install Pre-Built Detectron2 (Available on Linux only)
2. Build Detectron2 from Source

We are going to follow the second method as it is more convenient.

Note: We must have to restart runtime after successful instalation of detectron2 in order to use it.

```
# cloning repo for source and other modules
!git clone https://github.com/facebookresearch/detectron2 detectron2_repo
```

```
Cloning into 'detectron2_repo'...
remote: Enumerating objects: 12, done.
remote: Counting objects: 100% (12/12), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 7084 (delta 0), reused 2 (delta 0), pack-reused 7072
Receiving objects: 100% (7084/7084), 3.28 MiB | 1.74 MiB/s, done.
Resolving deltas: 100% (5020/5020), done.
```

```
# installing from repo (making build)
!pip install -e detectron2_repo
```

```
Obtaining file:///content/detectron2_repo
Requirement already satisfied: termcolor>=1.1 in /usr/local/lib/python3.6/dis
Collecting Pillow>=7.1
  Downloading https://files.pythonhosted.org/packages/30/bf/92385b4262178ca22
     |████████████████████████████████| 2.2MB 4.8MB/s
Requirement already satisfied: yacs>=0.1.6 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: tabulate in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-pa
Collecting mock
  Downloading https://files.pythonhosted.org/packages/cd/74/d72daf8dff5b6566d
Requirement already satisfied: tqdm>4.29.0 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: tensorboard in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: fvcore>=0.1.1 in /usr/local/lib/python3.6/dist
Requirement already satisfied: pycocotools>=2.0.2 in /usr/local/lib/python3.6
Requirement already satisfied: future in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: pydot in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: PyYAML in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-
```

Some Basic Imports

Its standard to import all the required packages and modules at the start.

```
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/di
```

```
# import some common libraries
import numpy as np
import pandas as pd
import os, json, cv2, random
from google.colab.patches import cv2_imshow
from cv2 import VideoWriter, VideoWriter_fourcc, imread, resize
import glob

# import some common detectron2 utilities
from detectron2.engine import DefaultTrainer
from detectron2.engine import DefaultPredictor
from detectron2.config import get_cfg
from detectron2.utils.visualizer import Visualizer
from detectron2.data import MetadataCatalog, DatasetCatalog
from detectron2.data.datasets import register_coco_instances

# imports for coco evaluation
from detectron2.evaluation import COCOEvaluator, inference_on_dataset
from detectron2.data import build_detection_test_loader
```

```
Requirement already satisfied: pyyaml<5.1,>=0.4.0 in /usr/local/lib/python3
```

Creating Custom Functions

Our complete work flow is as following:

1. Registering Our data with COCO
2. Visualizing registered dataset
3. Training Model on our dataset
4. Making Predictor from trained model
5. Evaluating Predictions on validation set

6. Making and Visualising predictions on model's unseen images (test set)

As all the processes above are iterative and we have to use them again and again so the best aproach is to **define a custom function** for each process in the work flow and use it again and again.

```
## Function for registering data with COCO
def register_dataset(dataset,annotations,images_dir):
  """
  This function takes images and their annotations in coco format, register dataset
  dataset: string, name of dataset you want to register with coco.
  annotations: string, path of Json file for annotations in coco format.
  images_dir: string, path of directory where images are present.
  """
  register_coco_instances(dataset, {}, annotations, images_dir)
  metadata = MetadataCatalog.get(dataset)
  data_dict = DatasetCatalog.get(dataset)
  return metadata,data_dict


## Function for visualising the registered coco dataset
def visualize_annotated_imgs(dataset_dict,num_samples,metadata):
  """
  This functions takes coco's registered dataset's dict, matadata displayed the ann
  dataset_dict: dict, from coco's registered dataset.
  metadata: metadata from coco's dataset.
  num_samples: int, number of images to display
  """
  for d in random.sample(dataset_dict, num_samples):
    img = cv2.imread(d["file_name"])
    visualizer = Visualizer(img[:, :, ::-1], metadata=metadata, scale=0.5)
    vis = visualizer.draw_dataset_dict(d)
    cv2_imshow(vis.get_image()[:, :, ::-1])


## Function for training Model
def train_model(model_path,train_data,num_classes,model_weights,lr,iters,num_worke
  """
  This functions takes coco's registered dataset, model path, model weights and tra
  model_path: str, path of model to be used.
  train_data: str, coco registered dataset name for training.
  num_calsses: int, number of classes.
  model_weights: str, path of model weights for used model.
  lr: float, learning rate.
  iters: int, number of iterations.
  num_workers: int, number of CPUs to load the data. As colab has 2 CPU's so by def
  ims_per_batch: int, number of images per batch. As colab has one GPU so it will h
  batch_size: int, how many parts of an image are interesting, how many of these a
  test_data= str, coco registered dataset name for validation/testing.
  """
  cfg = get_cfg()
  cfg.merge_from_file(model_path)
  cfg.DATASETS.TRAIN = (train_data,)
  if test_data==None:
    cfg.DATASETS.TEST = ()  # no metrics implemented for this dataset
```

```
    else:
      cfg.DATASETS.TEST = (test_data,)
      cfg.TEST.EVAL_PERIOD = 100
    cfg.DATALOADER.NUM_WORKERS = num_workers
    cfg.MODEL.WEIGHTS = model_weights
    cfg.SOLVER.IMS_PER_BATCH = ims_per_batch
    cfg.SOLVER.BASE_LR = lr
    cfg.SOLVER.MAX_ITER = iters
    cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = batch_size_per_img
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = num_classes
    os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
    trainer = DefaultTrainer(cfg)
    trainer.resume_or_load(resume=False)
    trainer.train()
    return trainer


  ## Function for loading trained model
  def load_trained_model(model_path,weights,num_classes,threshold=0.85):
    """
    This function takes already trained model and return a predictor to use on test/
    model_path: str, path of model to be used in training.
    weights: str, path of trained model in pkl format (used by pytorch).
    num_calsses: int, number of classes.
    threshold: float, confidence level of model for making predictions.
    """
    cfg = get_cfg()
    cfg.merge_from_file(model_path)
    cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = threshold
    cfg.MODEL.WEIGHTS = weights
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = num_classes
    predictor = DefaultPredictor(cfg)
    return predictor



  # Function for visualizing predictions
  def visualize_predictions(predictor,imgs_path,metadata,single_img=False,num_imgs=5
    """
    This function takes predictor and test images and visualise the predictions on i
    predictor: trained model.
    imgs_path: str, path of directory for test images or path of single image in cas
    metadata: dataset matadata on which model was trained.
    num_imgs: num of images to display.

    """
    if single_img:
        im=cv2.imread(imgs_path)
        outputs=predictor(im)
        v = Visualizer(im[:, :,::-1], metadata, scale=scale)
        out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
        im=out.get_image()[:, :, ::-1]
        cv2_imshow(im)

    else:
        for img in os.listdir(imgs path)[:num imgs]:
```

```
        im=cv2.imread(os.path.join(imgs_path,img))
        outputs=predictor(im)
        v = Visualizer(im[:, :,::-1], metadata, scale=scale)
        out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
        im=out.get_image()[:, :, ::-1]
        cv2_imshow(im)


## Function for coco evaluation
def coco_evaluator(val_data,trainer):
  """
  This function takes validation/test data and trainer with which model was trained
  val_data: registered coco dataset (val/test).
  trainer: with which model was trained.
  """
  cfg = get_cfg()
  evaluator = COCOEvaluator(val_data, cfg, False, output_dir="./output/")
  val_loader = build_detection_test_loader(cfg, val_data)
  print(inference_on_dataset(trainer.model, val_loader, evaluator))


#function for converting predictions into dataFrame
def get_predictions_dataframe(imgs_path,predictor,metadata,num_imgs):
  """
  This Functions takes images_path, the predictor (trained model), metadata for da
  and returns back the dataframe containing the predicted instances informations o

  imgs_path: path of images directory.
  predictor: trained model to make predictions.
  num_imgs: number of images to make predictions
  """
  imgs_path=imgs_path
  num_imgs=num_imgs
  metadata=metadata
  data=[]
  for img in os.listdir(imgs_path)[:num_imgs]:
      im=cv2.imread(os.path.join(imgs_path,img))
      outputs=predictor(im)
      boxes = [i.cpu().detach().numpy().astype('float').tolist() for i in outputs[
      classes=[i.cpu().detach().numpy().astype('int').tolist() for i in outputs["i
      scores =[i.cpu().detach().numpy().astype('float32').tolist() for i in output
      height=outputs["instances"].image_size[0]
      width=outputs["instances"].image_size[1]
      name=img
      for i in range(len(boxes)):
        box = boxes[i]
        x1=box[0]
        y1=box[1]
        x2=box[2]
        y2=box[3]
        score=scores[i]
        class_id=classes[i]
        pred_class=metadata.thing_classes[class_id]
        data.append({"Folder Path":imgs_path,
                     "Img Name":name,
                     "Height":height
```

```
                      "height":height,
                      "Width":width,
                      "X1":f"{x1:.2f}",
                      "Y1":f"{y1:.2f}",
                      "X2":f"{x2:.2f}",
                      "Y2":f"{y2:.2f}",
                      "Predicted Class":pred_class,
                      "class_id":class_id,
                      "Confidence (%)":f"{(score*100):.2f}"})
    df=pd.DataFrame(data)
    return df


  # Function for making predictions on video frames
  def predict_on_frames(video_path,predictor,target_dir,metadata):
    """
    This Functions takes a video and trained model and split it into frame by frame 
    by drawing predictions on it and return image file paths to again convert them i

    video_path: str, path of video to be loaded.
    predictor: trained model.
    target_dir: desired path to save splited images.
    metadata: metadata of data on which model is trained.
    """


    video_path = video_path
    target_path = target_dir
    capture = cv2.VideoCapture(video_path)
    frame_count = 0
    while True:
        ret, frame = capture.read()
        if not ret:
            break
        frame_count += 1
        results = predictor(frame)

        v = Visualizer(frame[:, :, ::-1],
                       metadata=metadata,
                       scale=0.8
                       )
        v = v.draw_instance_predictions(results["instances"].to("cpu"))

        name = '{0}.jpg'.format(frame_count)
        name = os.path.join(target_path, name)
        cv2.imwrite(name, v.get_image()[:, :, ::-1])
    # Get all image file paths to a list.
    images = list(glob.iglob(os.path.join(target_path, '*.jpg')))
    # Sort the images by name index.
    images = sorted(images, key=lambda x: float(os.path.split(x)[1][:-3]))
    return images


  # Function for combining predicted video frames together
  def images_to_video(target_path, images, fps=30, size=None,is_color=True, format="
      """
```

```
    This Function takes splited frames of a video and combine them back into video

    target_path: path to save output video.
    images: list of images to be converted into video.
    """
    fourcc = VideoWriter_fourcc(*format)
    vid = None
    for image in images:
        if not os.path.exists(image):
            raise FileNotFoundError(image)
        img = imread(image)
        if vid is None:
            if size is None:
                size = img.shape[1], img.shape[0]
            vid = VideoWriter(target_path, fourcc, float(fps), size, is_color)
        if size[0] != img.shape[1] and size[1] != img.shape[0]:
            img = resize(img, size)
        vid.write(img)
    vid.release()
    return vid
    print("Video Saved Successfuly.")
```

Getting Data Ready

This is the main process (also called as back-bone) of any machine/deep learning problem. So, we have to get our data so that we can use it and train models on it.
Here is the complete workflow for this process:

- Data Gathering
- Data Preprocessing (Annotating Images)
- Data Loading
- Registering Data Set

1. Data Gathering

DataSet in our case is the images of beverages/cold drinks on shelves of marts. So, we gathered our data by visiting multiple stores in the city and taking images from them. We have about 200 images and we have already split them into Train, Test and Validations sets by 80%,10% and 10% respectively. Almost all of the images are real world and taken by myself. Only a few images (around 20) are taken from the internet but also they depict real world beverages on shelf.

2. Image Annotations

There are multiple tools available for annotating images and also there are several techniques to annotate objects in an image. We have used `COCO Annotator` for annotating our images. This is the best tool as it directly converts annotations into COCO format (which is the required format for Detectron-2). More information about COCO Annotator can be found here : https://github.com/jsbroks/coco-annotator.

Secondly, the technique that we used for annotation is Binding Boxes. Where we draw rectangle boxes around the objects to tell the model what they are and where they are and which category they belong to.

3. Data Loading

Here in colab we have three ways to load our data into our work place. They are following:

1. Upload data set during runtime.
2. Mount Google Drive.
3. Download Data into colab workplace during runtime.

Option-1 is strongly not recommended by colab as each time run time will disconnect, we have to upload data again and again.

Option-2 seems great, as we can mount our G-Drive and access our storage to use our dataset. I have used this option in my experiments and work during model training.

Option-3 is good for someone else to use this notebook it is easy to download data using colab into workplace during run time. Server is very fast and data can be downloaded in few minutes. But again the problem is that it will be deleted after each time the runtime will disconnect.

Here I am going to use the Option-2 as it is very good and almost standard way of loading data.

Mounting Google Drive

The cell below will mount our `google drive` with collab so that we can easily access our drive data.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

4. Registering Data Set

This is the most important and compulsory part. Here we need to register our dataset to COCO Format. We will utilise our predefined custom function to make it easy and reuseable. Simply, we have to provide path for our dataset folder/directory, path for annotations file, and desired name for dataset.

```
annotations_path="/content/drive/My Drive/Bevarages_Dset/annotations/train.json"
images_dir_path="/content/drive/My Drive/Bevarages_Dset/train"
fanta_metadata,fanta_dict=register_dataset(dataset="fanta",annotations=annotations_
fanta_metadata
```

```
Category ids in annotations are not in [1, #categories]! We'll apply a mappin
Metadata(evaluator_type='coco', image_root='/content/drive/My Drive/Bevarages
```

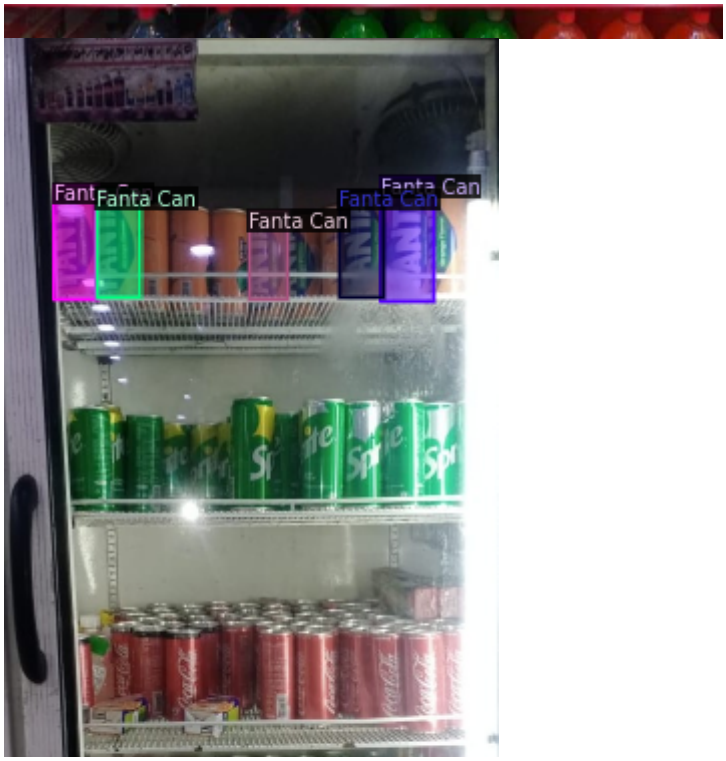
Visualising Registered Data

Let's see how our registered data look like. It is actually set of images with their annotations.

```
visualize_annotated_imgs(dataset_dict=fanta_dict,num_samples=5,metadata=fanta_meta
```


Sprite Bottle
Sprite Bottle
Fanta 1.5L
Fanta 1.5L
Cappy
Cappy
Coca Cola Bottle
Coca Cola Bottle
Coca Cola Bottle
Coca Cola Bottle


Sprite
Fanta 1.5L



Training The Model

This is the core part of modeling. Let's train our model using pre defined custom function and setting up right parameters. This step is very experimenting. Set different parameters and observe the results. After a long duration of experiments and studies I have choosen some best parameters and a good model to train.

Here are some important parameters that we need to understand.

`model_path` : path of model to be used. can be obtained from detectron2_repo in workplace. I am using Segmentation model `mask_rcnn_R_101_C4_3x` for training on my datset which I have found after some experiments. I tried many other models like `retinanet_R_101_FPN_3x`, `faster_rcnn_R_101_FPN_3x`, `faster_rcnn_R_50_FPN_1x`, `mask_rcnn_R_50_C4_3x`, `mask_rcnn_R_50_FPN_3x.` and `mask_rcnn_X_101_32x8d_FPN_3x`. But the best model i found was `mask_rcnn_R_101_C4_3x`.

`train_data` : coco registered dataset name for training.

`num_calsses` : number of classes in dataset.

`model_weights` : path of model weights for used model. Can be obtained from model file used in model path.

`learning rate (lr)` : learning rate for model (to optimize the loss, Detectron2 is using SGD Optimizer by default). The orignal trained (big model) model was trained on 8 GPU's and with 0.1 learning rate and as we have only 2 GPU's then its better to divide 0.1 with 8 and then find some related lr.

`iters` : number of iterations we wish our model to make on dataset and these can be obtained by some experiments. we have to choose the best iters so that model learns a lot and also not overfit.

`num_workers` : number of CPUs to load the data. As colab has 2 CPU's so by default its 2 in our predefined function.

`ims_per_batch` : number of images per batch. As colab has one GPU so it will be 2 by default.

`batch_size` : how many parts of an image are interesting, how many of these are we going to find?. By default its 128.

`test_data` = coco registered dataset name for validation/testing. This is optional.

Note : If we have already trained the model then we can skip this step and move to next step for loading trained model. (One of my trained model can be found in dataset directory and can be loaded).

Select Model and weights

model_path: /content/detectron2_repo/configs/COCO-InstanceSegmentation/mask_rcnn_R_101_C4_3

model_weights: " detectron2://ImageNetPretrained/MSRA/R-101.pkl "

Set Hyper Parameters

num_classes: 11

lr: 0.0125

iters: 6000

```
#training
trainer=train_model(model_path=model_path,train_data="fanta",
                    num_classes=num_classes,
                    model_weights=model_weights,
                    lr=lr,
                    iters=iters)
```

```
[09/24 11:29:21 d2.engine.defaults]: Model:
GeneralizedRCNN(
  (backbone): ResNet(
    (stem): BasicStem(
      (conv1): Conv2d(
        3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False
        (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
      )
    )
    (res2): Sequential(
      (0): BottleneckBlock(
        (shortcut): Conv2d(
          64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
          (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
        )
        (conv1): Conv2d(
          64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
        (conv2): Conv2d(
          64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
        (conv3): Conv2d(
          64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
          (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
        )
      )
      (1): BottleneckBlock(
        (conv1): Conv2d(
          256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
        (conv2): Conv2d(
          64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
        (conv3): Conv2d(
          64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
          (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
        )
      )
      (2): BottleneckBlock(
        (conv1): Conv2d(
          256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
        (conv2): Conv2d(
          64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=Fal
          (norm): FrozenBatchNorm2d(num_features=64, eps=1e-05)
        )
        (conv3): Conv2d(
          64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
          (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
        )
      )
    )
    (res3): Sequential(
      (0): BottleneckBlock(
        (shortcut): Conv2d(
          256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False
```

```
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
    (conv1): Conv2d(
      256, 128, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
  (1): BottleneckBlock(
    (conv1): Conv2d(
      512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
  (2): BottleneckBlock(
    (conv1): Conv2d(
      512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
  (3): BottleneckBlock(
    (conv1): Conv2d(
      512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv2): Conv2d(
      128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=128, eps=1e-05)
    )
    (conv3): Conv2d(
      128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=512, eps=1e-05)
    )
  )
)
(res4): Sequential(
  (0): BottleneckBlock(
    (shortcut): Conv2d(
      512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
```

```
      512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
    (conv1): Conv2d(
      512, 256, kernel_size=(1, 1), stride=(2, 2), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (1): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (2): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (3): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (4): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
```

```
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (5): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (6): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (7): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (8): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
```

```
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (9): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (10): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (11): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (12): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (13): BottleneckBlock(
```

```
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (14): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (15): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (16): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv3): Conv2d(
      256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=1024, eps=1e-05)
    )
  )
  (17): BottleneckBlock(
    (conv1): Conv2d(
      1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False
      (norm): FrozenBatchNorm2d(num_features=256, eps=1e-05)
    )
    (conv2): Conv2d(
      256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=F
```