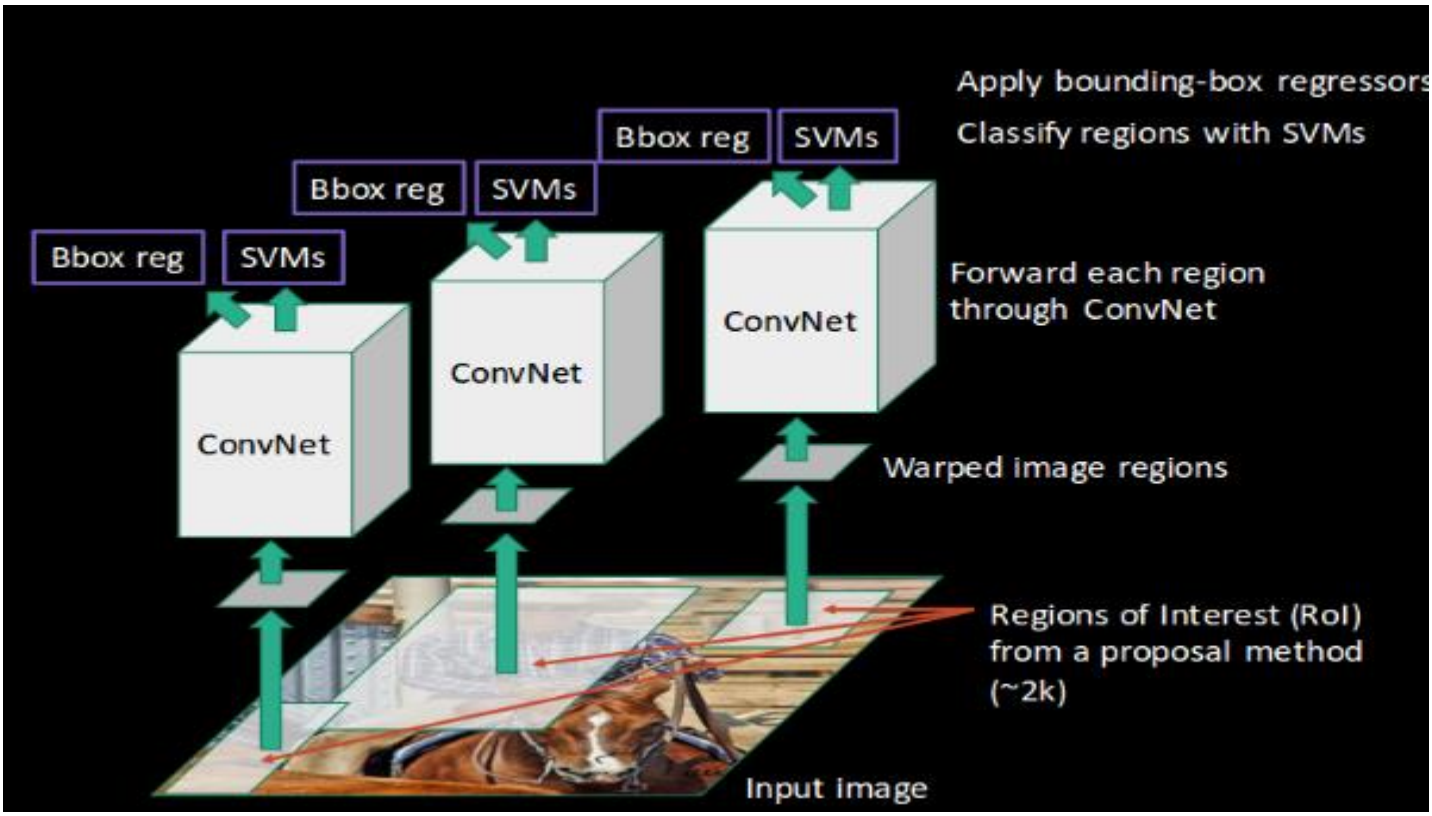
National University of Computer & Emerging Sciences, Karachi
Artificial Intelligence-School of Computing
Fall 2024, Lab Manual – 8

Course Code (AI4002)	**Course: Computer Vision Lab**
Instructor(s):	**Sohail Ahmed**

Objectives:

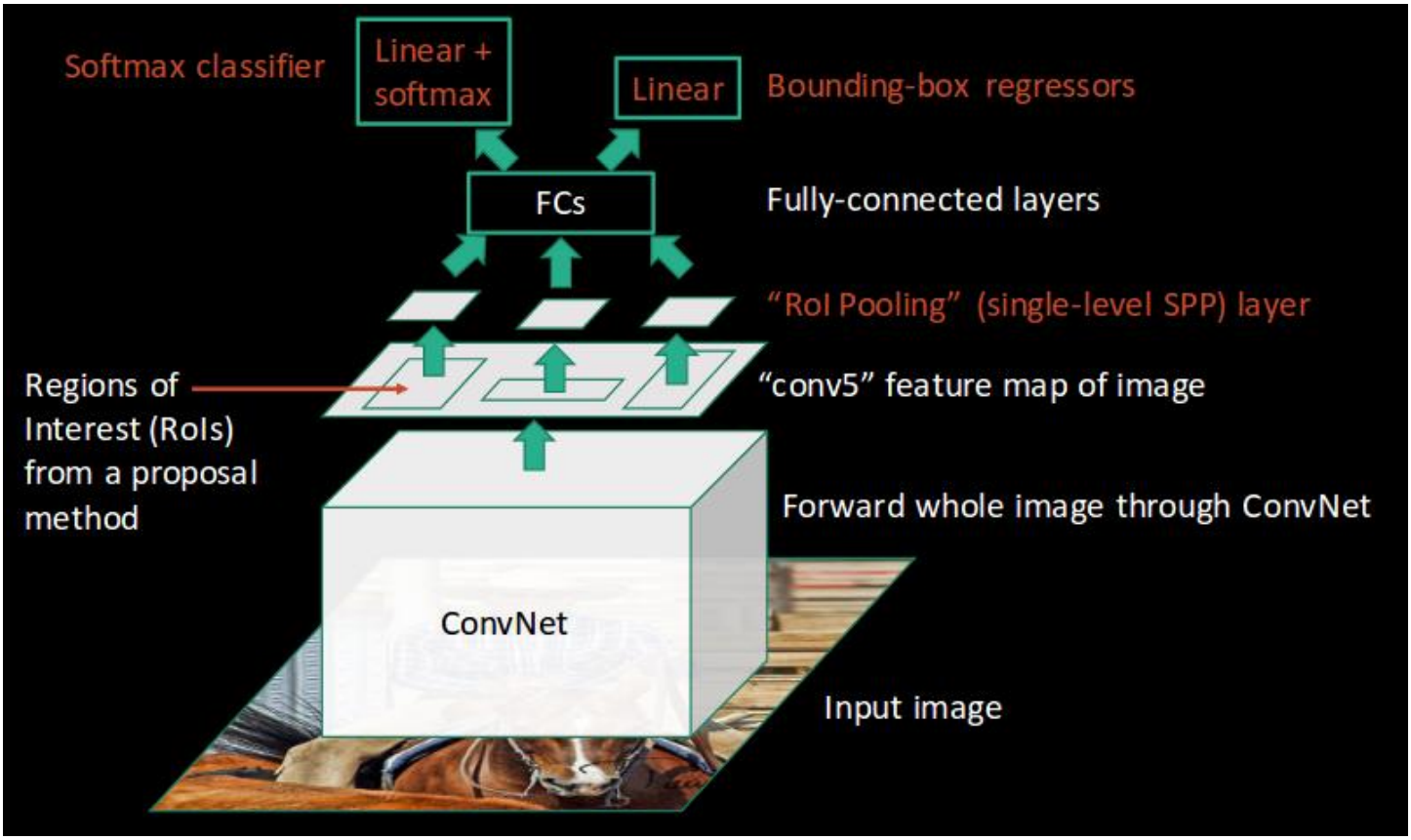
- RCNN
- YOLO

R-CNN extracts a bunch of regions from the given image using selective search, and then checks if any of these boxes contains an object. We first extract these regions, and for each region, CNN is used to extract specific features. Finally, these features are then used to detect objects. Unfortunately, R-CNN becomes rather slow due to these multiple steps involved in the process.

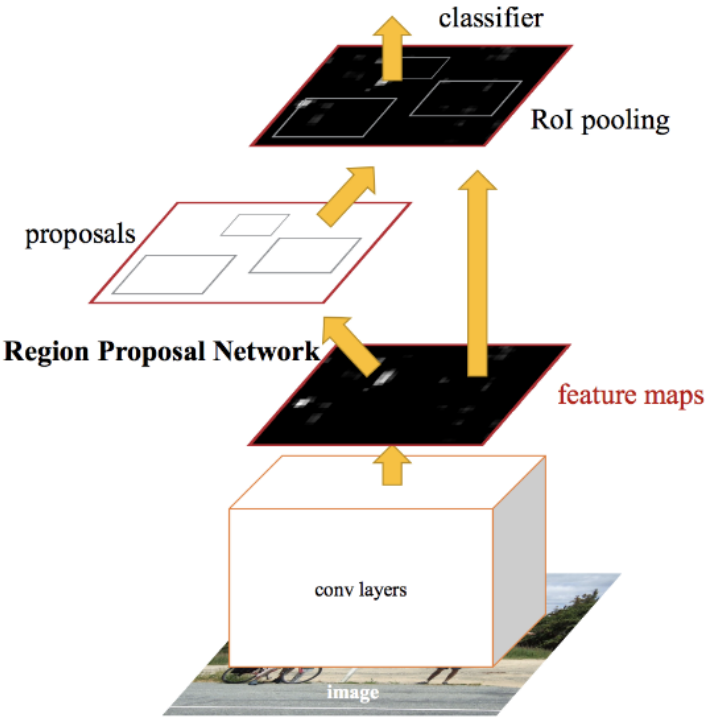


Fast R-CNN, on the other hand, passes the entire image to ConvNet which generates regions of interest (instead of passing the extracted regions from the image). Also, instead of using three different models (as we saw in R-CNN), it uses a single model which extracts features from the regions, classifies them into different classes, and returns the bounding boxes.

All these steps are done simultaneously, thus making it execute faster as compared to R-CNN. Fast R-CNN is, however, not fast enough when applied on a large dataset as it also uses selective search for extracting the regions.



Faster R-CNN fixes the problem of selective search by replacing it with Region Proposal Network (RPN). We first extract feature maps from the input image using ConvNet and then pass those maps through a RPN which returns object proposals. Finally, these maps are classified and the bounding boxes are predicted.



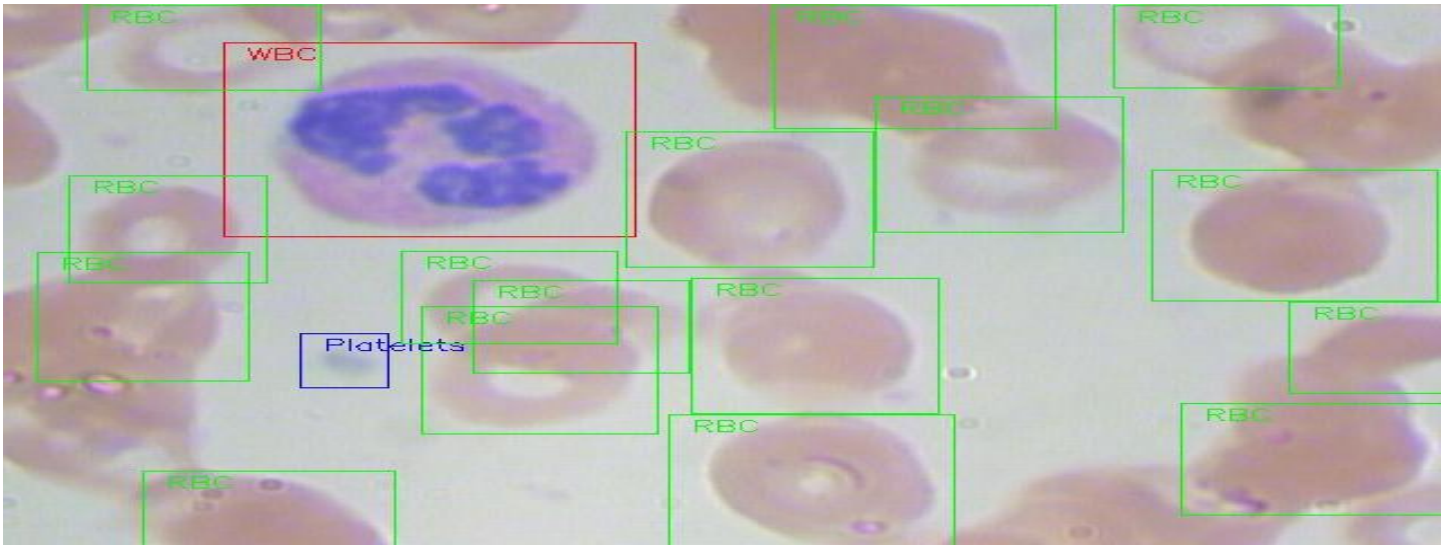
The steps followed by a Faster R-CNN algorithm to detect objects in an image:

1. Take an input image and pass it to the ConvNet which returns feature maps for the image
2. Apply Region Proposal Network (RPN) on these feature maps and get object proposals
3. Apply ROI pooling layer to bring down all the proposals to the same size
4. Finally, pass these proposals to a fully connected layer in order to classify any predict the bounding boxes for the image

Algorithm	Features	Prediction time / image	Limitations
CNN	Divides the image into multiple regions and then classifies each region into various classes.	–	Needs a lot of regions to predict accurately and hence high computation time.
R-CNN	Uses selective search to generate regions. Extracts around 2000 regions from each image.	40-50 seconds	High computation time as each region is passed to the CNN separately. Also, it uses three different models for making predictions.
Fast R-CNN	Each image is passed only once to the CNN and feature maps are extracted. Selective search is used on these maps to generate predictions. Combines all the three models used in R-CNN together.	2 seconds	Selective search is slow and hence computation time is still high.
Faster R-CNN	Replaces the selective search method with region proposal network (RPN) which makes the algorithm much faster.	0.2 seconds	Object proposal takes time and as there are different systems working one after the other, the performance of systems depends on how the previous system has performed.

Understanding the Problem Statement

We will be working on a healthcare related dataset and the aim here is to solve a Blood Cell Detection problem. Our task is to detect all the Red Blood Cells (RBCs), White Blood Cells (WBCs), and Platelets in each image taken via microscopic image readings. Below is a sample of what our final predictions should look like:



The reason for choosing this dataset is that the density of RBCs, WBCs and Platelets in our blood stream provides a lot of information about the immune system and hemoglobin. This can help us potentially identify whether a person is healthy or not, and if any discrepancy is found in their blood, actions can be taken quickly to diagnose that.

Link to BCCD Dataset

Data Exploration

It's always a good idea to first explore the data we have. This helps us not only unearth hidden patterns, but gain a valuable overall insight into what we are working with. The three files I have created out of the entire dataset are:

1. train_images: Images that we will be using to train the model. We have the classes and the actual bounding boxes for each class in this folder.
2. test_images: Images in this folder will be used to make predictions using the trained model. This set is missing the classes and the bounding boxes for these classes.
3. train.csv: Contains the name, class and bounding box coordinates for each image. There can be multiple rows for one image as a single image can have more than one object.

Let's read the .csv file (you can create your own .csv file from the original dataset if you feel like experimenting) and print out the first few rows. We'll need to first import the below libraries for this:

```
# importing required libraries
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import patches
# read the csv file using read_csv function of pandas
train = pd.read_csv('train.csv')
train.head()
```

	image_names	cell_type	xmin	xmax	ymin	ymax
0	1.jpg	RBC	68	165	154	249
1	1.jpg	RBC	1	66	145	260
2	1.jpg	RBC	207	334	160	270
3	1.jpg	RBC	435	540	347	437
4	1.jpg	RBC	535	639	356	464

There are 6 columns in the train file. Let's understand what each column represents:

- image_names: contains the name of the image
- cell_type: denotes the type of the cell
- xmin: x-coordinate of the bottom left part of the image
- xmax: x-coordinate of the top right part of the image
- ymin: y-coordinate of the bottom left part of the image
- ymax: y-coordinate of the top right part of the image

```
# reading single image using imread function of matplotlib
image = plt.imread('images/1.jpg')
plt.imshow(image)
```

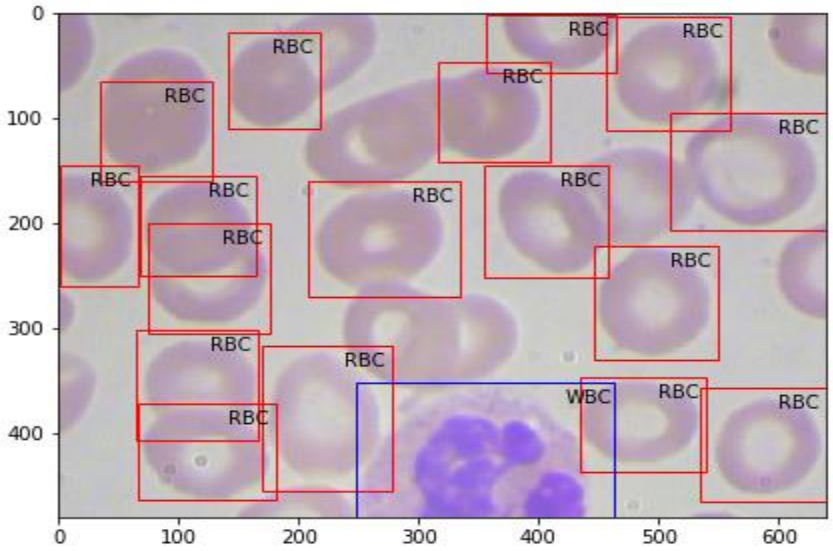
```
# Number of unique training images
train['image_names'].nunique()

#So, we have 254 training images.

# Number of classes
train['cell_type'].value_counts()
```

```
RBC          2909
WBC           262
Platelets     252
Name: cell type, dtype: int64
```

We have three different classes of cells, i.e., RBC, WBC and Platelets. Finally, let's look at how an image with detected objects will look like:

```
fig = plt.figure()

#add axes to the image
ax = fig.add_axes([0,0,1,1])

# read and plot the image
image = plt.imread('images/1.jpg')
plt.imshow(image)

# iterating over the image for different objects
for _,row in train[train.image_names == "1.jpg"].iterrows():
    xmin = row.xmin
    xmax = row.xmax
    ymin = row.ymin
    ymax = row.ymax

    width = xmax - xmin
    height = ymax - ymin

    # assign different color to different classes of objects
    if row.cell_type == 'RBC':
        edgecolor = 'r'
        ax.annotate('RBC', xy=(xmax-40,ymin+20))
    elif row.cell_type == 'WBC':
```

```
    edgecolor = 'b'
    ax.annotate('WBC', xy=(xmax-40,ymin+20))
elif row.cell_type == 'Platelets':
    edgecolor = 'g'
    ax.annotate('Platelets', xy=(xmax-40,ymin+20))

# add bounding boxes to the image
rect = patches.Rectangle((xmin,ymin), width, height, edgecolor = edgecolor, facecolor = 'none')

ax.add_patch(rect)
```



This is what a training example looks like. We have the different classes and their corresponding bounding boxes. Let's now train our model on these images. We will be using the keras_frcnn library to train our model as well as to get predictions on the test images.

Implementing Faster R-CNN

Clone this repository: git clone https://github.com/kbardool/keras-frcnn.git
Move the train_images and test_images folder, as well as the train.csv file, to the cloned repository. In order to train the model on a new dataset, the format of the input should be:
filepath,x1,y1,x2,y2,class_name
where,

- filepath is the path of the training image
- x1 is the xmin coordinate for bounding box
- y1 is the ymin coordinate for bounding box
- x2 is the xmax coordinate for bounding box
- y2 is the ymax coordinate for bounding box
- class_name is the name of the class in that bounding box

We need to convert the .csv format into a .txt file which will have the same format as described above. Make a new dataframe, fill all the values as per the format into that dataframe, and then save it as a .txt file.

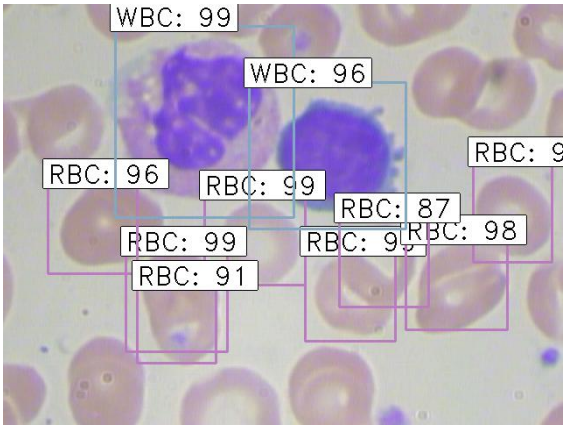
```
data = pd.DataFrame()
data['format'] = train['image_names']

# as the images are in train_images folder, add train_images before the image name
for i in range(data.shape[0]):
    data['format'][i] = 'train_images/' + data['format'][i]

# add xmin, ymin, xmax, ymax and class as per the format required
for i in range(data.shape[0]):
    data['format'][i] = data['format'][i] + ',' + str(train['xmin'][i]) + ',' + str(train['ymin'][i]) + ',' + str(train['xmax'][i]) + ',' + str(train['ymax'][i]) + ',' + train['cell_type'][i]

data.to_csv('annotate.txt', header=None, index=None, sep=' ')
```

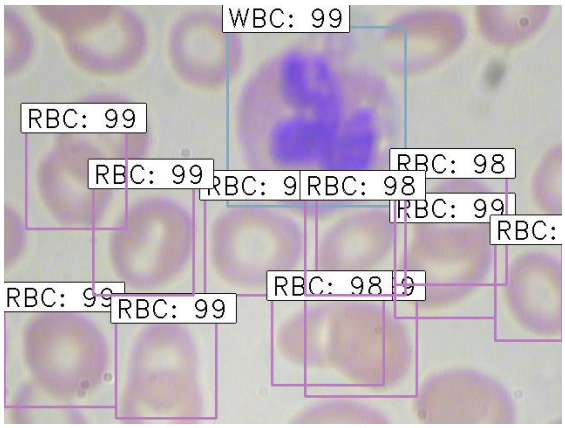
Train our model! We will be using the train_frcnn.py file to train the model.
cd keras-frcnn
python train_frcnn.py -o simple -p annotate.txt

It's prediction time! Keras_frcnn makes the predictions for the new images and saves them in a new folder. We just have to make two changes in the test_frcnn.py file to save the images:
Remove the comment from the last line of this file:

cv2.imwrite('./results_imgs/{}.png'.format(idx),img)
Add comments on the second last and third last line of this file:
cv2.imshow('img', img)
cv2.waitKey(0)

Let's make the predictions for the new images:
python test_frcnn.py -p test_images





Lab Task:

Task: Object Detection Using Regional Convolutional Neural Networks (R-CNN)

(Don't use the mentioned dataset use another one)

Objective:

Implement object detection using a Regional Convolutional Neural Network (R-CNN) on a dataset of your choice. R-CNN is a classic object detection method that combines region proposal, feature extraction, and object classification to identify and localize objects within images.

Object detection has become an increasingly popular field in computer vision, with YOLO (You Only Look Once) being one of the most widely used algorithms. In this blog post, we will explore how to use YOLO and a webcam to get started with a real-time object detection system.

YOLO was developed by Joseph Redmon and his team at the University of Washington and has become one of the most popular object detection algorithms used in computer vision applications

Unlike traditional object detection algorithms that require multiple passes over an image, YOLO processes the entire image in a single pass, making it much faster and more efficient.

YOLO has been used in a variety of applications, including self-driving cars, security systems, and image and video analysis.

YOLO has been implemented in several deep learning frameworks, including Darknet, TensorFlow, and PyTorch. The original implementation of YOLO was done using the Darknet framework, which was developed by Joseph Redmon.

1. We'll capture frames from the webcam using OpenCV. This can be done using the VideoCapture function in OpenCV.

```
import cv2
cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

while True:
    ret, img= cap.read()
    cv2.imshow('Webcam', img)
    if cv2.waitKey(1) == ord('q'):
        break
```

```
cap.release()
cv2.destroyAllWindows()
```

2. We install the ultralytics library that makes working with YOLO very easy and hassle-free.
   $ pip install ultralytics

3. The YOLO model is loaded using the ultralytics library and specifies the location of the YOLO weights file in the yolo-Weights/yolov8n.pt.
   ```
   from ultralytics import YOLO
   model = YOLO("yolo-Weights/yolov8n.pt")
   ```
4. We instantiate a classNames variable containing a list of object classes that the YOLO model is trained to detect.

```
classNames = ["person", "bicycle", "car", "motorbike", "aeroplane", "bus", "train", "truck", "boat",
        "traffic light", "fire hydrant", "stop sign", "parking meter", "bench", "bird", "cat",
        "dog", "horse", "sheep", "cow", "elephant", "bear", "zebra", "giraffe", "backpack", "umbrella",
        "handbag", "tie", "suitcase", "frisbee", "skis", "snowboard", "sports ball", "kite", "baseball bat",
        "baseball glove", "skateboard", "surfboard", "tennis racket", "bottle", "wine glass", "cup",
        "fork", "knife", "spoon", "bowl", "banana", "apple", "sandwich", "orange", "broccoli",
        "carrot", "hot dog", "pizza", "donut", "cake", "chair", "sofa", "pottedplant", "bed",
        "diningtable", "toilet", "tvmonitor", "laptop", "mouse", "remote", "keyboard", "cell phone",
        "microwave", "oven", "toaster", "sink", "refrigerator", "book", "clock", "vase", "scissors",
        "teddy bear", "hair drier", "toothbrush"
        ]
```

5. The while loop starts and it reads each frame from the webcam using cap.read(). Then it passes the frame to the YOLO model for object detection. The results of object detection are stored in the ‘results’ variable.

```
import cv2

cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

while True:
  ret, img= cap.read()
  results = model(img, stream=True)

  cv2.imshow('Webcam', frame)

  if cv2.waitKey(1) == ord('q'):
     break

cap.release()
cv2.destroyAllWindows()
```

6. For each result, the code extracts the bounding box coordinates of the detected object and draws a rectangle around it using cv2.rectangle(). It also prints the confidence score and class name of the detected object on the console.

Complete Code:

In [14]:
```
while True:
    success, img = cap.read()
    results = model(img, stream=True)

    # coordinates
    for r in results:
        boxes = r.boxes

        for box in boxes:
            # bounding box
            x1, y1, x2, y2 = box.xyxy[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2) # convert to int values

            # put box in cam
            cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 255), 3)

            # confidence
            confidence = math.ceil((box.conf[0]*100))/100
            print("Confidence --->",confidence)

            # class name
            cls = int(box.cls[0])
            print("Class name -->", classNames[cls])

            # object details
            org = [x1, y1]
            font = cv2.FONT_HERSHEY_SIMPLEX
            fontScale = 1
            color = (255, 0, 0)
            thickness = 2

            cv2.putText(img, classNames[cls], org, font, fontScale, color, thickness)

    cv2.imshow('Webcam', img)
    if cv2.waitKey(1) == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

```
0: 480x640 1 person, 187.5ms
Speed: 5.0ms preprocess, 187.5ms inference, 0.0ms postprocess per image at shape (1, 3, 480, 640)

Confidence ---> 0.92
```

Lab Task:

1. **Use Yolo any version using live web camera detect your lab assets.**