# COMPARATIVE ANALYSIS OF DEEP LEARNING ALGORITHMS – YOLO AND MASK R-CNN AND IMPLEMENTING OBJECT DETECTION

by

| | |
|---|---|
| N Vedanjali | 19BEE1159 |
| Mahima Rajesh | 19BEE1188 |
| Pappula Rajasri | 19BEE1118 |

A project report submitted to

**Dr. ARUL R**

**SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING**

in partial fulfilment of the requirements for the course of

**EEE1007 – NEURAL NETWORK AND FUZZY CONTROL**

in

**B.Tech. ELECTRICAL AND ELECTRONICS ENGINEERING**

**VIT CHENNAI**

**Vandalur – Kelambakkam Road**

**Chennai – 600127**

**MAY 2021**

## INDEX

# 1.BONAFIDE CERTIFICATE

Certified that this project report entitled “**COMPARATIVE ANALYSIS OF DEEP LEARNING ALGORITHMS – YOLO AND Mask R – CNN AND IMPLEMENTING OBJECT DETECTION**” is a bonafide work of **N VEDANJALI** (19BEE1159), **MAHIMA RAJESH** (19BEE1188) and **PAPPULA RAJASRI** (19BEE1118) who carried out the Project work under my supervision and guidance.

**Dr. ARUL R**

**Associate Professor**

**School of Electrical and Electronics Engineering (SELECT),**

**VIT University, Chennai**

**Chennai – 600127.**

## 2.ABSTRACT

21st century has seen a huge rise in a number of technological advancements in the field of AI, ML and DL with respect to numerous arenas especially with regards to Computer Vision. Object detection is a computer vision technique that allows us to identify and locate objects in an image or video.

Considering the prevailing conditions of COVID’19, criminal cases, etc. this technique would be extremely useful in easing our task by allowing us to train the program to detect and identify object or people based on the datasets we have provided.

We have carried out a comparative study of the 2 popularly known algorithms for object detection: YOLO (You Only Look Once) and Mask R-CNN (Regions with Convolutional Neural Networks).

# 3.ACKNOWLEDGEMENT

We express our thanks to our Program Chair **Dr. R Chendur Kumaran** for his support throughout the course of this project.

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. Arul R** , School of Electrical and Electronics Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

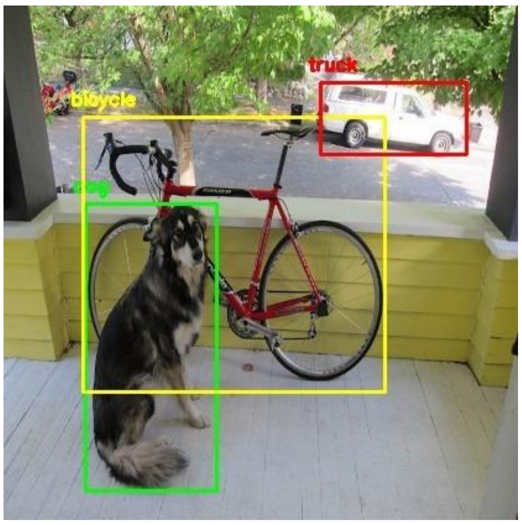
**N VEDANJALI** **MAHIMA RAJESH** **PAPPULA RAJASRI**

# 4.ALGORITHMS USED AND IMPLEMENTATION

## 1) **YOLO –**You Only Look Once

Object detection is one of the classical problems in computer vision where you work to recognize *what* and *where* — specifically what objects are inside a given image and also where they are in the image. The problem of object detection is more complex than classification, which also can recognize objects but doesn't indicate where the object is located in the image. In addition, classification doesn't work on images containing more than one object.

YOLO uses a totally different approach. YOLO is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.



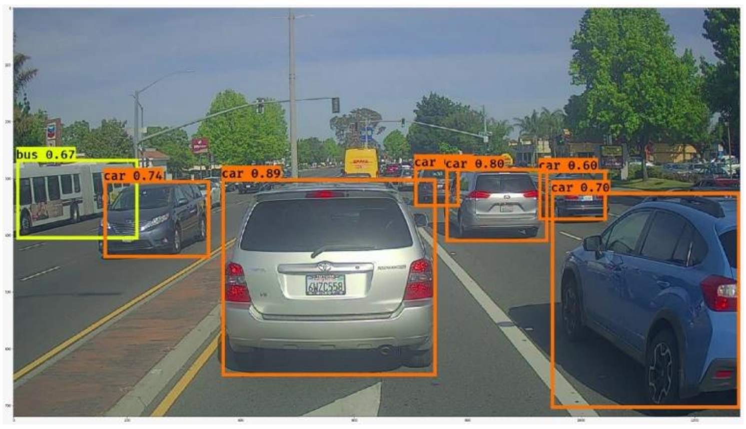
YOLO is popular because it achieves high accuracy while also being able to run in real-time. The algorithm "only looks once" at the image in the sense that it requires only one forward propagation pass through the neural network to make predictions. After non-max suppression (which makes sure the object detection algorithm only detects each object once), it then outputs recognized objects together with the bounding boxes.

With YOLO, a single CNN simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance.

**This model has a number of benefits over other object detection methods**:

- ✓ YOLO is extremely fast

- ✓ YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance.
- ✓ YOLO learns generalizable representations of object s so that when trained on natural images and tested on artwork, the algorithm outperforms other top detection methods.



## Main code for implementation of YOLOv3 for object detection

```
import numpy as np

import argparse

import cv2 as cv

import subprocess

import time

import os

from yolo_utils import infer_image, show_image

FLAGS = []

if _name_ == '_main_':

        parser = argparse.ArgumentParser()

        parser.add_argument('-m', '--model-path',

                type=str,
                default='./yolov3-coco/',

                help='The directory where the model weights and \

                    configuration files are.')
```

**#Add path to files which cotaints weights**

```
        parser.add_argument('-w', '--weights',

                type=str,

                default='C:\\Users\\girish.sai\\OneDrive - Drilling Info\\Desktop\\YOLOv3-Object-Detection-with-OpenCV\\yolov3-coco\\yolov3.weights',

                help='Path to the file which contains the weights \

                        for YOLOv3.')

        parser.add_argument('-cfg', '--config',

                type=str,

                default='C:\\Users\\girish.sai\\OneDrive - Drilling Info\\Desktop\\YOLOv3-Object-Detection-with-OpenCV\\yolov3-coco\\yolov3.cfg',

                help='Path to the configuration file for the YOLOv3 model.')

        parser.add_argument('-i', '--image-path',

                type=str,

                help='The path to the image file')

        parser.add_argument('-v', '--video-path',

                type=str,

                help='The path to the video file')

        parser.add_argument('-vo', '--video-output-path',

                type=str,

      default='./output.avi',

                help='The path of the output video file')

        parser.add_argument('-l', '--labels',

                type=str,
                default='./yolov3-coco/coco-labels',

                help='Path to the file having the \labels in a new-line seperated way.')

        parser.add_argument('-c', '--confidence',

                type=float,

                default=0.5,

                help='The model will reject boundaries which has a \probabiity less than the
                confidence value. \default: 0.5')

        parser.add_argument('-th', '--threshold',

                type=float,

                default=0.3,

                help='The threshold to use when applying the \Non-Max Suppresion')

        parser.add_argument('--download-model',

                type=bool,

                default=False,

                help='Set to True, if the model weights and configurations \are not present
on your local machine.')

        parser.add_argument('-t', '--show-time',

                type=bool,

                default=False,

                help='Show the time taken to infer each image.')

        FLAGS, unparsed = parser.parse_known_args()
```

**# Download the YOLOv3 models if needed**

```
        if FLAGS.download_model:

                subprocess.call(['./yolov3-coco/get_model.sh'])
```

**# Get the labels**

```
        labels = open(FLAGS.labels).read().strip().split('\n')
```

```
# Intializing colors to represent each label uniquely
        colors = np.random.randint(0, 255, size=(len(labels), 3), dtype='uint8')
# Load the weights and configutation to form the pretrained YOLOv3 model
        net = cv.dnn.readNetFromDarknet(FLAGS.config, FLAGS.weights)
# Get the output layer names of the model
        layer_names = net.getLayerNames()
        layer_names = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]
# If both image and video files are given then raise error
        if FLAGS.image_path is None and FLAGS.video_path is None:
          print ('Neither path to an image or path to video provided')
          print ('Starting Inference on Webcam')
        # Do inference with given image
        if FLAGS.image_path:
                # Read the image
                try:
                        img = cv.imread(FLAGS.image_path)
                        height, width = img.shape[:2]
                except:
                        raise 'Image cannot be loaded!\n\
                   Please check the path provided!'
                finally:
                        img, _, _, _, _ = infer_image(net, layer_names, height, width, img, 
colors, labels, FLAGS)
                        show_image(img)
        elif FLAGS.video_path:
                # Read the video
                Try:
                        vid = cv.VideoCapture(FLAGS.video_path)
                        height, width = None, None
                        writer = None
                except:
                        raise 'Video cannot be loaded!\n\
                   Please check the path provided!'
                finally:
                        while True:
                                grabbed, frame = vid.read()
                           # Checking if the complete video is read
                                if not grabbed:
                                        break
                                if width is None or height is None:
                                        height, width = frame.shape[:2]
                                frame, _, _, _, _ = infer_image(net, layer_names, height, width,
frame, colors, labels, FLAGS)
                                if writer is None:
                                        # Initialize the video writer
                                        fourcc = cv.VideoWriter_fourcc(*"MJPG")
                                        writer = cv.VideoWriter(FLAGS.video_output_path,
fourcc, 30,
                                        (frame.shape[1], frame.shape[0]), True)


                                writer.write(frame)
                        print ("[INFO] Cleaning up...")
                        writer.release()
                vid.release()
else:
    # Infer real-time on webcam
    count = 0
    vid = cv.VideoCapture(0)
    while True:
                _, frame = vid.read()
                height, width = frame.shape[:2]
                if count == 0:
                    frame, boxes, confidences, classids, idxs = infer_image(net,
                    layer_names, \height, width, frame, colors, labels, FLAGS)
                    count += 1
                else:
                    frame, boxes, confidences, classids, idxs = infer_image(net,
                    layer_names, \height, width, frame, colors, labels, FLAGS,
                    boxes, confidences, classids, idxs, infer=False)
                    count = (count + 1) % 6
                cv.imshow('webcam', frame)
                if cv.waitKey(1) & 0xFF == ord('q'):
                    break
    vid.release()
    cv.destroyAllWindows()
```

## 2) **MASK R-CNN:** Mask regions with convolution neural networks

The Region-Based Convolutional Neural Network, or R-CNN, is a family of convolutional neural network models designed for object detection, developed by Ross Girshick, et al.

There are perhaps four main variations of the approach, resulting in the current pinnacle called Mask R-CNN. The salient aspects of each variation can be summarized as follows:

- **R-CNN**: Bounding boxes are proposed by the "*selective search*" algorithm, each of which is stretched and features are extracted via a deep convolutional neural network, such as AlexNet, before a final set of object classifications are made with linear SVMs.
- **Fast R-CNN**: Simplified design with a single model, bounding boxes are still specified as input, but a region-of-interest pooling layer is used after the deep CNN to consolidate regions and the model predicts both class labels and regions of interest directly.
- **Faster R-CNN**: Addition of a Region Proposal Network that interprets features extracted from the deep CNN and learns to propose regions-of-interest directly.
- **Mask R-CNN**: Extension of Faster R-CNN that adds an output model for predicting a mask for each detected object.

Mask R-CNN is basically an extension of Faster R-CNN. Faster R-CNN is widely used for object detection tasks. For a given image, it returns the class label and bounding box coordinates for each object in the image.

Mask R-CNN has an additional branch for predicting segmentation masks on each Region of Interest (ROI) in a pixel-to pixel manner.

Faster R-CNN is not designed for pixel-to-pixel alignment between network inputs and outputs.
**Faster R-CNN has two outputs**

- For each candidate object, a class label and a bounding-box offset;

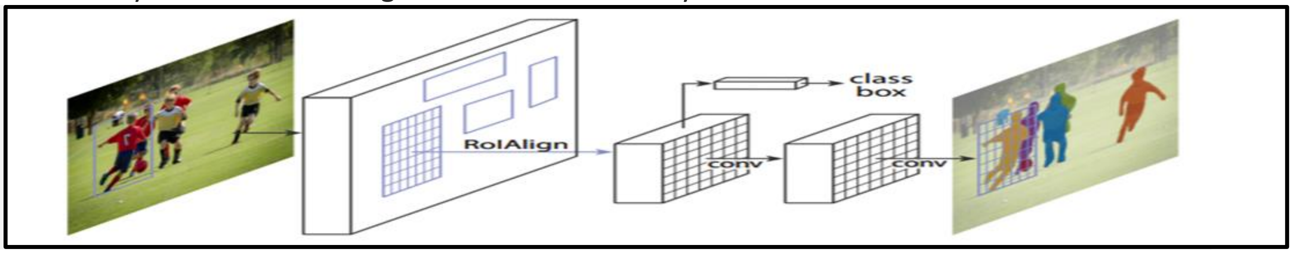
**Mask R-CNN has three outputs**

- For each candidate object, a class label and a bounding-box offset;
- Third output is the object mask

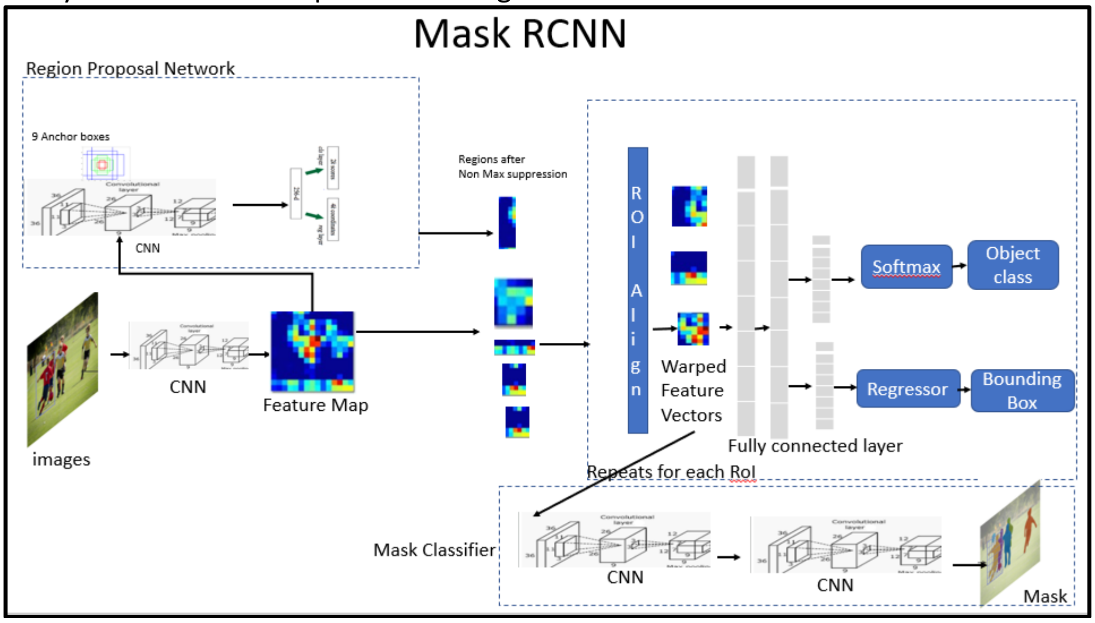
*How does Mask R-CNN work?*

**Mask R-CNN model is divided into two parts**

- Region proposal network (RPN) to proposes candidate object bounding boxes.
- Binary mask classifier to generate mask for every class



- Image is run through the CNN to generate the feature maps.
- Region Proposal Network (RPN) uses a CNN to generate the multiple Region of Interest (ROI) using a lightweight binary classifier. It does this using 9 anchors boxes over the image. The classifier returns object/no-object scores. Non-Max suppression is applied to Anchors with high objectness score.
- The ROI Align network outputs multiple bounding boxes rather than a single definite one and warp them into a fixed dimension.
- Warped features are then fed into fully connected layers to make classification using softmax and boundary box prediction is further refined using the regression model.
- Warped features are also fed into Mask classifier, which consists of two CNN's to output a binary mask for each ROI. Mask Classifier allows the network to generate masks for every class without competition among classes.



## Main code for implementation of MASK R-CNN for Object Detection:

```
import cv2

import numpy as np

import os

import sys

from mrcnn import utils

from mrcnn import model as modellib

ROOT_DIR = os.path.abspath("./")

   # Directory to save logs and trained model

MODEL_DIR = os.path.join(ROOT_DIR, "logs")

# Import COCO config

sys.path.append(os.path.join(ROOT_DIR,"samples/coco/"))

import coco

   # Local path to trained weights file

COCO_MODEL_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")

   # Download COCO trained weights from Releases if needed

if not os.path.exists(COCO_MODEL_PATH):

   utils.download_trained_weights(COCO_MODEL_PATH)

class InferenceConfig(coco.CocoConfig):

   GPU_COUNT = 1

   IMAGES_PER_GPU = 1

config = InferenceConfig()

config.display()

model = modellib.MaskRCNN(

   mode="inference", model_dir=MODEL_DIR, config=config)
```

```
model.load_weights(COCO_MODEL_PATH, by_name=True)

class_names = [

    'BG', 'person', 'bicycle', 'car', 'motorcycle', 'airplane',

    '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', 'couch', 'potted plant', 'bed',

    'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote',

    'keyboard', 'cell phone', 'microwave', 'oven', 'toaster',

    'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors',

    'teddy bear', 'hair drier', 'toothbrush'

]

#generate random colours for mask

def random_colors(N):

    np.random.seed(1)

    colors = [tuple(255 * np.random.rand(3)) for _ in range(N)]

    return colors

colors = random_colors(len(class_names))

class_dict = {

    name: color for name, color in zip(class_names, colors)}
```

**“”” apply mask to image”””**

```
def apply_mask(image, mask, color, alpha=0.5):
    """apply mask to image"""
    for n, c in enumerate(color):
        image[:, :, n] = np.where(
            mask == 1,
            image[:, :, n] * (1 - alpha) + alpha * c,
            image[:, :, n]
        )
    return image
def display_instances(image, boxes, masks, ids, names, scores):
```

**“””take the image and results and apply the mask, box, and Label “””**

```
    n_instances = boxes.shape[0]
    if not n_instances:
        print('NO INSTANCES TO DISPLAY')
    else:
        assert boxes.shape[0] == masks.shape[-1] == ids.shape[0]
    for i in range(n_instances):
        if not np.any(boxes[i]):
            continue
        y1, x1, y2, x2 = boxes[i]
        label = names[ids[i]]
        color = class_dict[label]
        score = scores[i] if scores is not None else None
        caption = '{} {:.2f}'.format(label, score) if score else label
        mask = masks[:, :, i]
```

```
            image = apply_mask(image, mask, color)
            image = cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
            image = cv2.putText(
                image, caption, (x1, y1), cv2.FONT_HERSHEY_COMPLEX, 0.7, color, 2 )
        return image


    if _name_ == '_main_':
        “””test everything”””
        capture = cv2.VideoCapture(0)
        # these 2 lines can be removed if you dont have a 1080p camera.
        capture.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
        capture.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
        while True:
            ret, frame = capture.read()
            results = model.detect([frame], verbose=0)
            r = results[0]
            frame = display_instances(
                frame, r['rois'], r['masks'], r['class_ids'], class_names, r['scores']
            )
            cv2.imshow('frame', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        capture.release()
        cv2.destroyAllWindows()
```

# 5.ANALYSIS OF THE TWO ALGORITHMS

Object Detection is a common Computer Vision problem which deals with identifying and locating object of certain classes in the image. Interpreting the object localization can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which contains the object (called segmentation).

The algorithms we have used produce a list of object categories present in the image along with an axis-aligned bounding box indicating the position and scale of every instance of each object category. From an ocean full of object detection algorithms, we have chosen the top 2 algorithms while keeping in mind parameters like efficiency, accuracy, etc. The complete analysis and insight of each of the 2 algorithms are given below:

- **You Only Look Once (YOLO) Algorithm:**

Some of the best Object Detection algorithms solved some problems mentioned in the but fail on the most important one — Speed for real-time object detection. YOLO is one such algorithm which gives a much better performance on all the parameters we discussed along with a high fps for real-time usage. It is an algorithm based on regression, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in one run of the Algorithm. We have executed and implemented the YOLO algorithm using tools like Python, OpenCV, TensorFlow and Keras, most of which are libraries and packages of Python.

Through this algorithm, we ultimately aim to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors:

- Center of the box (bx, by)
- Width (bw)
- Height (bh)
- Value c corresponding to the class of an object

Along with that we predict a real number pc, which is the probability that there is an object in the bounding box.

YOLO doesn’t search for interested regions in the input image that could contain an object, instead it splits the image into cells, typically 19x19 grid. Each cell is then responsible for predicting K bounding boxes.

**Algorithm YOLOv3** ()

Step 1: Start

Step 2: Set the video capture mode on along with the threshold values for confidence and NMS (non-maximum suppression)

Step 3: Train the model with the “coco” dataset

Step 4: For (every object in each frame extracted) {get x and y coordinates}

Step 5: append box

Step 6: End

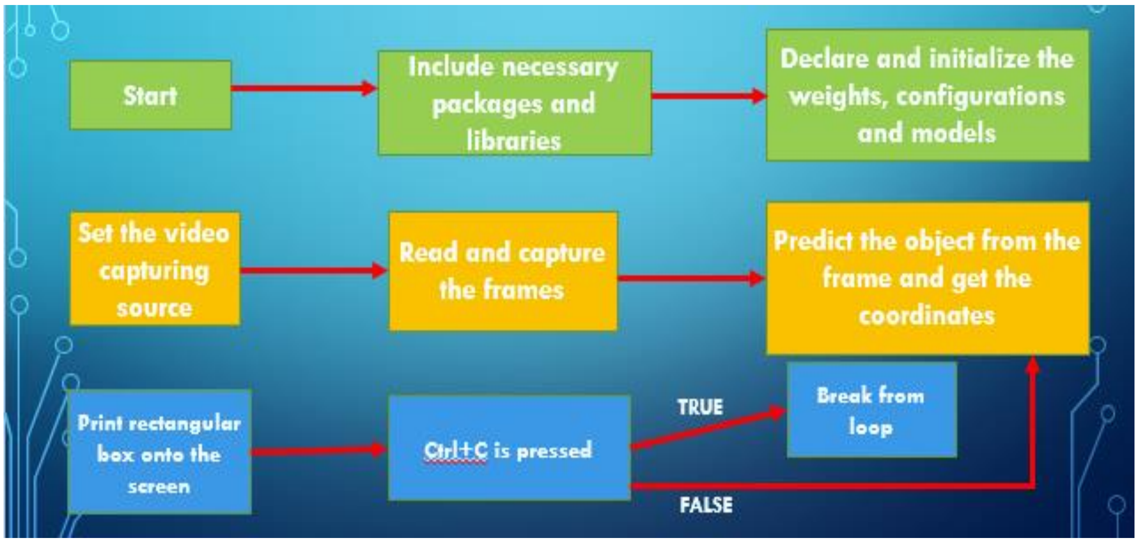

Start
Include necessary packages and libraries
Declare and initialize the weights, configurations and models
Set the video capturing source
Read and capture the frames
Predict the object from the frame and get the coordinates
Print rectangular box onto the screen
Ctrl+C is pressed
TRUE
Break from loop
FALSE

**Fig 1. Workflow of YOLO**

The above-mentioned algorithm and workflow give a small insight into the actual working of the YOLO algorithm and explain in detail the step by step extraction and training process for the algorithm to work.

- **Training and Datasets used**

Installing the appropriate datasets, parsing through the annotation file, developing the dataset object, testing the dataset objects, and then training the model to detect the objects forms a major part of the initial stages of developing the algorithm.

We have made use of *COCO dataset*. This dataset has around 80 labels which include: People, Bicycles, Cars and trucks, Airplanes, Stop signs and fire hydrants, Animals, including cats, dogs, birds, horses, cows, and sheep, to name a few, Kitchen and dining objects, such as wine glasses, cups, forks, knives, spoons, etc.

- **MASK R-CNN (Region based Convolutional Neural Network) Algorithm**

Region Based Convolutional Neural Networks (R-CNN) are a family of machine learning models for computer vision and specifically object detection. The original goal of R-CNN was to take an input image and produce a set of bounding boxes as output, where each bounding box contains an object and also the category (e.g. car or pedestrian) of the object.

**Algorithm for Mask R—CNN()**

```
{
Step 1: Take an image as input and extracts around 2000 region proposals from the image
Step 2: Each region proposal is then warped(reshaped) to a fixed size to be passed on as an input to a CNN.
Step 3: The CNN extracts a fixed-length feature vector for each region proposal
Step 4: These features are used to classify region proposals using category-specific linear SVM
Step 5: The bounding boxes are refined using bounding box regression so that the object is properly captured by the box.
}
```

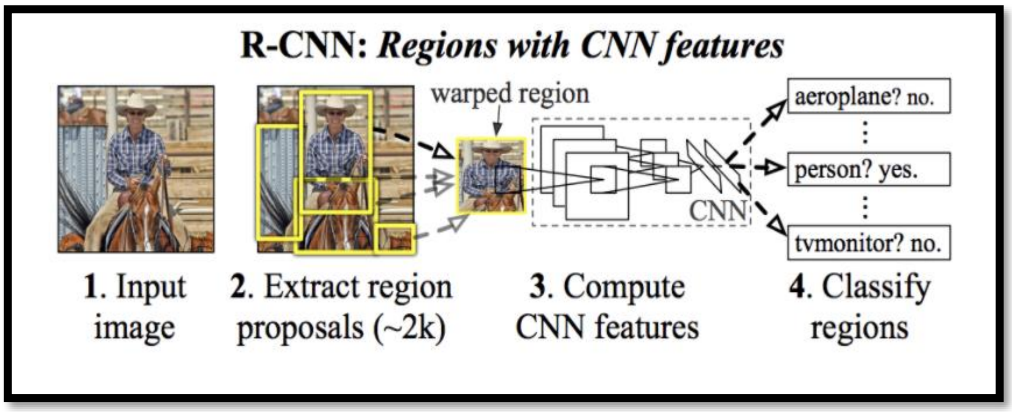


**Fig 3. Workflow of R-CNN Algorithm**

While implementing this algorithm, we made use of region proposals which are the bounding boxes that may contain an object.

These are represented by a tuple of 4 numbers(x,y,h,w). The (x,y) are the coordinates of the centre of the bounding box and (h,w) are the height and width of the bounding box respectively. These region proposals are calculated by an algorithm called selective search. For an image, approximately 2000 region proposals are extracted.

## ➢ ***<u>Training and Datasets Used</u>***

To train the CNN for feature extraction, an architecture like VGG-16 was initialized with the pre-trained weights from image-net data. The output layer having 1000 classes is chopped off. So when a region proposal image (warped to size 224x224) is passed to the network we get a 4096-dimensional feature vector as shown in the above image. We have made use of the coco dataset which trains the model with the following objects:

‘BG’, ‘person’, ‘bicycle’, ‘car’, ‘motorcycle’, ‘airplane’, ‘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’, ‘couch’, ‘potted plant’, ‘bed’, ‘dining table’, ‘toilet’, ‘tv’, ‘laptop’, ‘mouse’, ‘remote’, ‘keyboard’, ‘cell phone’, ‘microwave’, ‘oven’, ‘toaster’, ‘sink’, ‘refrigerator’, ‘book’, ‘clock’, ‘vase’, ‘scissors’, ‘teddy bear’, ‘hair drier', 'toothbrush'

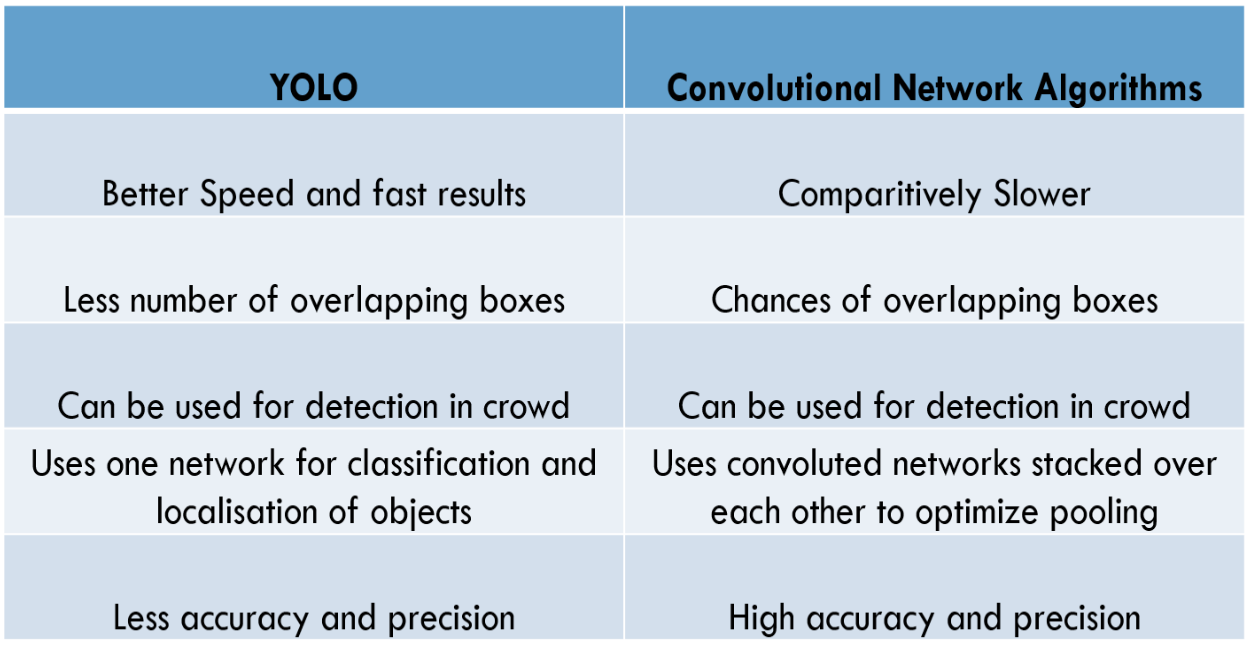
## ➢ **Observations from the two Algorithms**

Based on our analysis and implementation, we have come to conclude that YOLO (You Only Look Once) system, an open-source method of object detection that can recognize objects in images and videos swiftly whereas algorithms like R-CNN run a convolutional network on input image only one time and computes a feature map.

Convolutional Algorithms are a better option as we are able to run it on a video and the exactness trade-off is very modest. However, if exactness is not too much of disquiet but you want to go super quick, YOLO will be the best way to move forward. First of all, a visual thoughtfulness of swiftness Vs precision trade-off would differentiate them well.

- MASK RCNN offers a region of interest for doing convolution while YOLO does detection and classification at the same time.
- YOLO appears to be a cleaner way of doing object detection since it’s fully end-to-end training. The MASK RCNN offers end-to-end training as well, but the steps involved are more.

We have summed up our observations in the table given below:



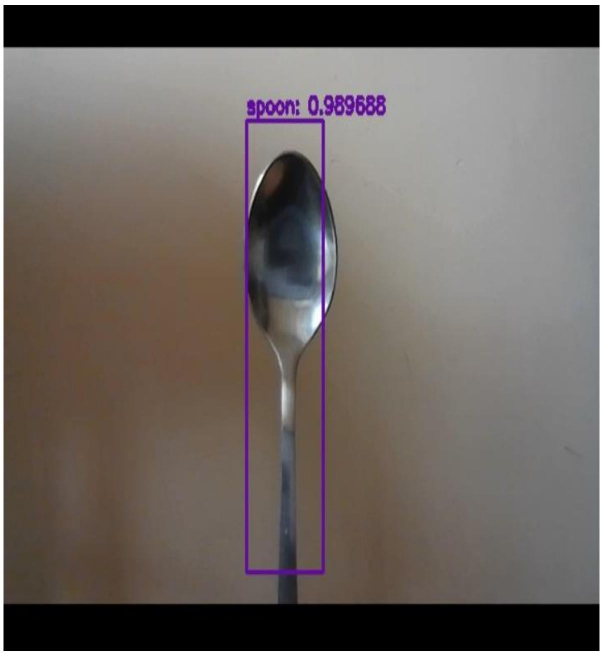
| YOLO | Convolutional Network Algorithms |
|---|---|
| Better Speed and fast results | Comparitively Slower |
| Less number of overlapping boxes | Chances of overlapping boxes |
| Can be used for detection in crowd | Can be used for detection in crowd |
| Uses one network for classification and localisation of objects | Uses convoluted networks stacked over each other to optimize pooling |
| Less accuracy and precision | High accuracy and precision |

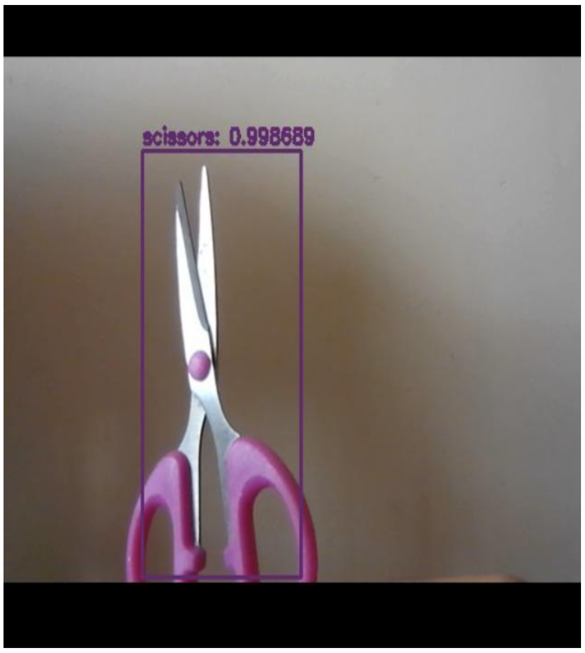
The comparison between speed and accuracy of different object detection models:

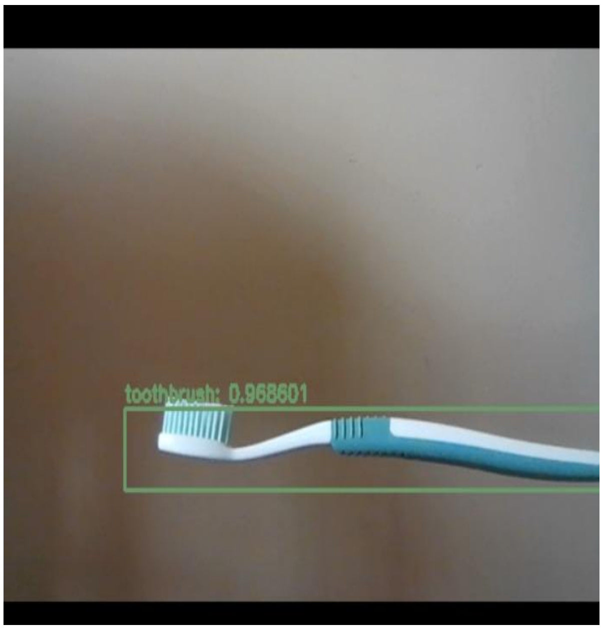
| Algorithm | Speed (FPS) | Accuracy (mAP) |
|---|---|---|
| **YOLO** | 45 | 63.4% |
| **R-CNN** | 7 | 73.2% |

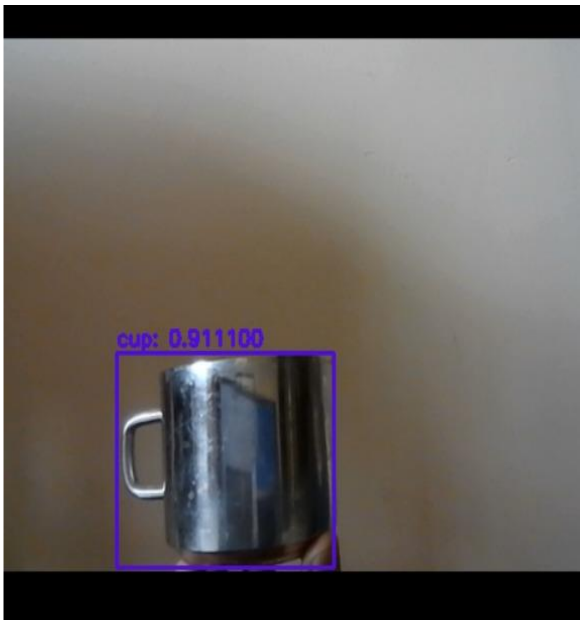
# 6. RESULTS AND OUTPUTS

**IMPLEMENTATION OF YOLOv3:**

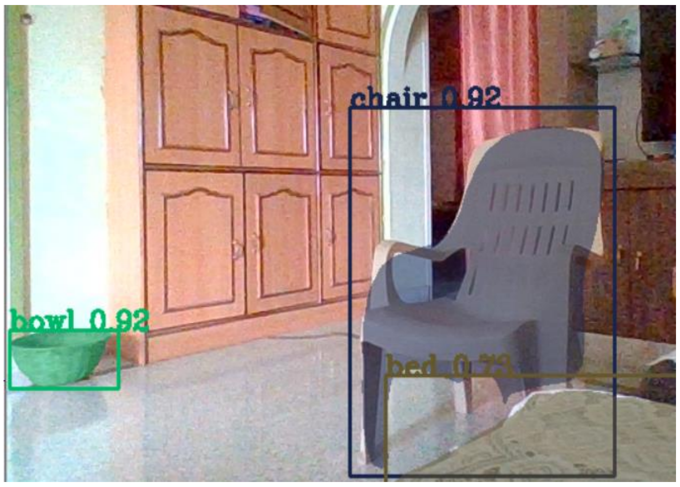


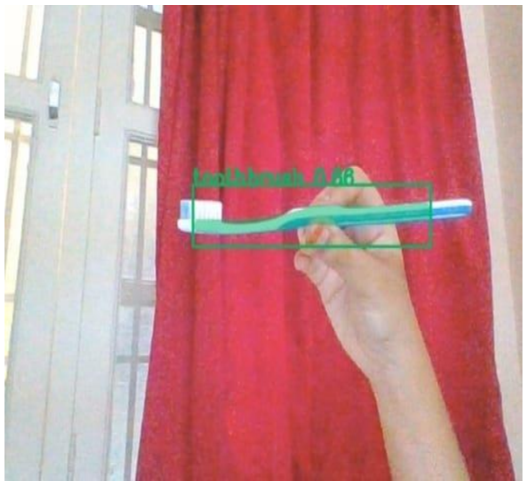






**IMPLEMENTATION OF MASK RCNN:**





# 7.CONCLUSION

Object detection is a key ability for most computer and robot vision system. Although great progress has been observed in the last years one of which we developed is object and mask detection and some existing techniques are now part of many consumer electronics or have been integrated in assistant driving technologies, we are still far from achieving human-level performance, in particular in terms of open-world learning.

It should be noted that object detection has not been used much in many areas where it could be of great help especially during this pandemic. After a proper analysis of both the algorithms YOLO and RCNN in various aspects we concluded that RCNN is the best algorithm.

In a Convolutional Neural Network dimensional reduction is achieved using a sliding window with a size less than that of the input matrix unlike YOLO. When compared with YOLO's algorithm RCNN provides a proper end to end training too which makes it more efficient. CNN's are really effective for image classification as the concept of dimensionality reduction suits the huge number of parameters in an image. Hence RCNN is the best algorithm for object detection.

# 8. FUTURE SCOPE

Although the provided analyses and methodologies are good and constitute a set of useful tools to guarantee the real-time requirements of the object detection, there are some improvements that can still be made. In this section we will survey some of the provided results which can be improved or modified further. This section also briefly describes some interesting research topics, which are worth investigating further. In the following, we outline problems that we believe have not been addressed, or addressed only partially, and may be interesting in relevant research directions.

- The use of new sensing modalities, in particular depth and thermal cameras, has seen some development in the last years [e.g., Fehr and Burkhardt (2008) ]. However, the methods used for processing visual images are also used for thermal images, and to a lesser degree for depth images. Using depth images is easy to segment the objects, but general methods for detecting specific classes has not been proposed, and probably higher resolution depth images are required.

- It seems that depth and thermal cameras alone are not enough for object detection, at least with their current resolution, but further advances can be expected as the sensing technology improves.

- The project can even be extended in identifying the different types of objects as per requirement. The code has been designed for identifying the objects alone but can be used and modified for detecting other safety objects like gloves, masks etc. considering the current pandemic situation.

- Also we can use the object detection in heavy trafic areas or at by-pass road where there is chance of accidents. Using a fast object detection the information can be notified to police stations or hospitals automatically.

# 9.REFERENCES

[1] Geethapriya. S, N. Duraimurugan, S.P. Chokkalingam “Real-Time Object Detection with Yolo” *International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-8, Issue-3S, February 2019.*

[2] Naman Mittal, Akarsh Vaidya, Sherya Kapoor “Object Detection and Classification Using Yolo” *International Journal of Scientific Research & Engineering Trends Volume 5, Issue 2*, Mar-Apr-2019, ISSN (Online): 2395-566X.

[3] W. Yang and Z. Jiachun, "Real-time face detection based on YOLO," *2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII)*, 2018, pp. 221-224, doi: 10.1109/ICKII.2018.8569109.

[4] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137-1149, 1 June 2017, doi: 10.1109/TPAMI.2016.2577031.

[5] https://iopscience.iop.org/article/10.1088/1742-6596/1004/1/012029/meta

[6] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980-2988, doi: 10.1109/ICCV.2017.322.