# Implementation of CNN for Real Time Object detection & comparing results of two different algorithms.

**Project Report**

**Submitted By:**

18BEE0201 - Sucheet Dumbre
18BCE0791 - Siddhant Keskar
18BCE0809 - Sudnyesh Talekar
18BCE0811 - Devang Patel

**Submitted To:**

Prof. Chiranji Lal Chowdhary
Associate Professor, SITE, VIT Vellore

**Winter Semester**
**December 2019 - June 2020**

VIT®

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

## Presentation Drive Video Link:

https://drive.google.com/file/d/114dBWUTj39ohCCShbn3HF-qwkoShPJg2/view?usp=sharing

## Synopsis:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics
The goal of R-CNN is to take in an image, and correctly identify where the main objects in the image are.

R-CNN works really well, but is really quite slow for a few simple reasons:

1. It requires a forward pass of the CNN for every single region proposal for every single image
2. It has to train three different models separately.

R-CNN creates these bounding boxes, or region proposals, using a process called Selective Search. At a high level, Selective Search (shown in the image above) looks at the image through windows of different sizes, and for each size tries to group together adjacent pixels by texture, color, or intensity to identify objects.
More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding box, eliminate duplicate detections, and rescore the box based on other objects in the scene.
YOLO is refreshingly simple. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes.YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits
over traditional methods of object detection.

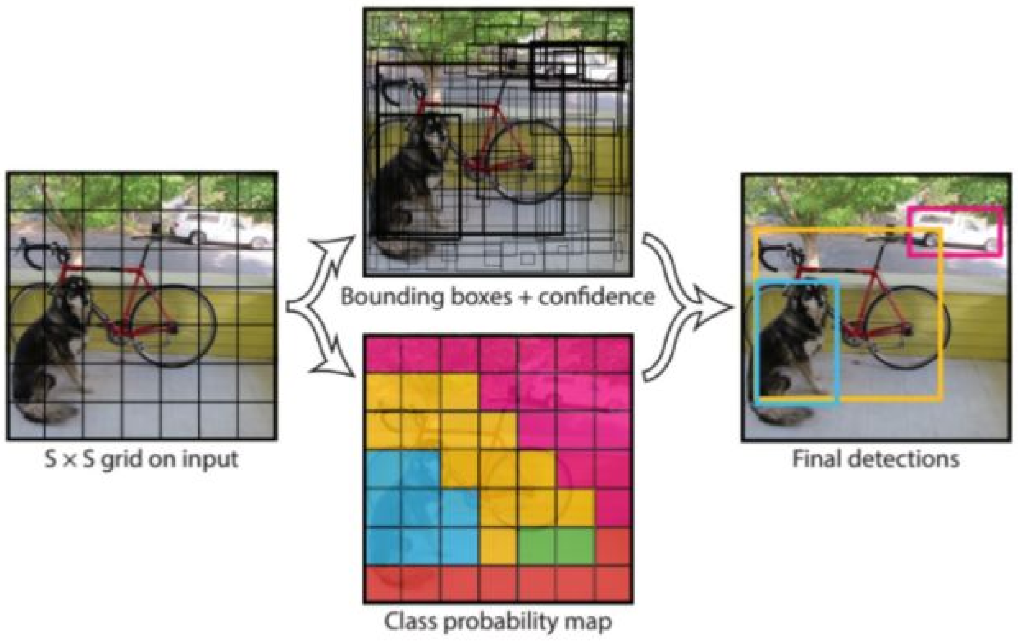


**Fig 1: The Model divides the image into SxS grid and then B bounding Box with C as class Probabilities.**

## Project Report for Review 0 (First)

Initial research regarding real time object detection led to R-CNN  which was used in the past couple of years, After finding the and researching the disadvantage of primitive R-CNN algorithms which are slow and more performance hungry we came across YOLO.
Since then YOLO (You Only Look Once) has been the recent trend in the image processing and deep learning industry. YOLO inspired us  to compare the perform of the primitive R-CNN with the latest YOLO darknet dataset.
A lot of progress has been made in recent years on object detection due to the use of convolutional neural networks (CNNs). Modern object detectors based on these networks - such as Faster R-CNN, R-FCN, Multibox, SSD and YOLO - are now good enough to be deployed in consumer products (e.g., Google Photos, Visual Search) and some have been shown to be fast enough to be run on mobile devices.

Some advantage of YOLO which attracted us:

1. Speed (**45 frames per second** — better than realtime|gives this result on good GPUs)
2. Network understands generalized object representation (This allowed them to train the network on real world images and predictions on artwork was still fairly accurate).
3. faster version (with smaller architecture) — 155 frames per sec but is less accurate.
4. open source: https://pjreddie.com/darknet/yolo/

## Project Report for Review 1 (Second)

As we had discussed about different algorithms like CNN,R-CNN and YOLO that we would be comparing during review-0 in detail, we went ahead to see what parameters can be used  for comparison between these algorithms. We discussed on how these algorithms have the ability to use every single core of a CPU and want even more when there is nothing more that a CPU could provide, a reason because of which cloud platform is used for computing the results and how the cloud is facilitating a lot towards these high computational hungry algorithms.

The accuracy of the algorithm is another parameter which was stressed upon, there exists various algorithms which are made by various different computer scientists but are never implemented because their accuracy level is very low, because of which using it in real time scenario is not possible. But the algorithms we are using are based on Deep learning which means they can be trained by giving inputs and telling them the objects present in a frame. Due to this feature of the modern algorithms it has been possible to use these algorithms in real time scenarios. Though there is a lot of difference in the accuracy of these algorithms, if we are talking at a micro scale.

Time is a factor that has creeped into other parts of this discussion, but it needs to be attended because the actual time taken by these algorithms matters a lot, and it depends on what logic the code is using to identify the object in a frame, YOLO is said to be the fastest object detection algorithm, which is the most used worldwide.

**REVIEW 3**

## Table of Contents

## 1. Introduction:

A lot of progress has been made in recent years on object detection due to the use of convolutional neural networks (CNNs). Modern object detectors based on these networks - such as Faster R-CNN, R-FCN, Multibox, SSD and YOLO - are now good enough to be deployed in consumer products (e.g., Google Photos, Visual Search) and some have been shown to be fast enough to be run on mobile devices.
However, it can be difficult for practitioners to decide what architecture is best suited to their application. Standard accuracy metrics, such as mean average precision (mAP), do not tell the entire story, since for real deployments of computer vision systems, running time and memory usage are also critical. For example, mobile devices often require a small memory footprint, and self driving cars require real time performance.

So, here comes the magic of YOLO Algorithm which is fast but also reliably accurate and also provides good frames per seconds(FPS).

### a. Motivation:

In this paper we seek to compare results between two different algorithms namely CNN and YOLO based on their performance, time complexity, accuracy, memory complexity, FPS. Not only this we also test the time required to train the model. The most important question is not which detector is the best. It may not be possible to answer. The real question is which detector and what configurations give us the best balance of speed and accuracy that your application needed. Image resolution factor is also taken into the account when we required high resolution computation for accurate results.

## b. Contribution:

We can see that YOLO and Faster RCNN both share some similarities. They both use an anchor box based network structure, both use bounding both regression. Things that differ YOLO from Faster RCNN is that it makes classification and bounding box regression at the same time. Judging from the year they were published, it makes sense that YOLO wanted a more elegant way to do regression and classification. YOLO however does have its drawback in object detection. YOLO has difficulty detecting objects that are small and close to each other due to only two anchor boxes in a grid predicting only one class of object. It doesn’t generalize well when objects in the image show rare aspects of ratio. Faster RCNN on the other hand, does detect small objects well since it has nine anchors in a single grid, however it fails to do real-time detection with its two step architecture. We help to compare the performance between the both algorithms and find conclusions  for the best use case.

## c. Organisational report:

This dilemma motivates us to try out both the algorithms and measure and study the performance, accuracy etc. for multiple test scenarios considering the performance both time and memory, with accuracy and also time taken by each model to train in a mobile GPU which will be the practical application of the YOLO in the coming future in the sectors of security, efficiency etc. Considering efficiency, we can train a certain model which can detect whether people are there in a room or not which will help us to save electricity by automatically turning lights off.This will not only be limited to this sector but may find application in each and every sector in the coming future.

## 2. LITERATURE SURVEY:

There have been many attempts at making CNN better for real time workflows like making R-CNN and Faster R-CNN. The R-CNN was used to implement pedestrian Detection and Vehicle detection. R-CNN had many drawbacks, Training R-CNN was a multi-step process, expensive in space and time, as it needed enormous datasets to train which could take days, and even after days of training the object detection would be slow.

| Reference | Methods used | Merits and Demerits |
|---|---|---|
| **https://arxiv.org/pdf/1506.02640v5.pdf** | **YOLO** | Fast. Good for real-time processing.<br><br>This model struggles with small objects that appear in groups, such as flocks of birds. |
| **https://www.researchgate.net/publication/324754316_Understanding_of_Object_Detection_Based_on_CNN_Family_and_YOLO** | **Faster-R CNN** | Gain in speed in training and prediction than other RCNN models.<br><br>It's inefficient for resizing every region proposal to a fixed size before CNN, and it causes image distortion |

Faster R-CNN has its disadvantages. Faster R-CNN was composed of 5 main parts: a deep fully convolutional network, region proposal network, ROI pooling and fully connected networks, bounding box regressor, and classifier. The deep, fully convolutional and region proposal network proposes many object candidates. The ROI pooling layer then normalizes the candidate region proposed by the network. Then, fully connected layers extract useful features to conduct classification and regression. Most of the proposals are not always fit target objects, and the regressor tries to find the best fit. However, the classification is conducted on the proposals rather than on the best fits. In practical use, the size of target objects are various: for example, cars, humans, and license plates. If sizes of target objects are various, many of the proposals are more significant so that they contain small objects as well as large objects. It is not efficient for both classifiers and regressors and produces false positives and false negatives. Then came Mask R-CNN which tried to make Faster R-CNN batter in its way.

Most importantly, Faster R-CNN was not designed for pixel-to-pixel alignment between network inputs and outputs. This is most evident in how ROI pool layer is created, the de facto core operation for attending to instances, performing coarse spatial quantization for feature extraction. Mask R-CNN adopts the same procedure as Faster R-CNN. In the step where ROI pooling does regression and classification, MASK R-CNN creates a binary mask for each candidate region. This process is in contrast to most modern systems, where classification depends on mask predictions. This approach follows the spirit of Fast R-CNN that applies bounding-box classification and regression in parallel, and which also simplify the multi-step process of R-CNN. using CCD cameras simultaneously store images from cameras installed in multiple locations. In such an environment, the use of deep learning requires a method of detecting objects through a single inference engine in a plurality of image. If the inference engine hardware is used for each camera channel, the cost of building a surveillance system increases significantly. A deep-running model with faster and more accurate balance performance than YOLOv3-tiny is necessary. Darknet-53 model reduces the number of layers in the existing YOLOv3 network model to shorten the flops and feature extraction time.Darknet-53 is a feature extractor less than 21 layers. Only a few convolutional layers are added in the primary feature extractor. Finally, we get the 3-d tensor as output, which contains information about box, objectness, and class.

The above algorithms are not thoroughly tested with degrading Images; in other words, their training is done on academic datasets and not tested with randomly captured data sets. The issues that degrading images face are blurring due to instability of camera, poor image quality due to bad weather, overexposure or low resolution. The YOLO neural network integrates the candidate boxes extraction, feature extraction and objects classification methods into a neural network. The YOLO neural network directly extracts candidate boxes from images and objects are detected through the entire image features. YOLO with darknet 53 network is trained for such degenerating photos.

## 3. BACKGROUND OF YOUR PROJECT WORK

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The goal of R-CNN is to take in an image, and correctly identify where the main objects in the image are.
R-CNN works really well, but is really quite slow for a few simple reasons:

1) It requires a forward pass of the CNN for every single region proposal for every single image.
2) It has to train three different models separately.

R-CNN creates these bounding boxes, or region proposals, using a process called Selective Search. At a high level, Selective Search (shown in the image above) looks at the image through windows of different sizes, and for each size tries to group together adjacent pixels by texture, color, or intensity to identify objects.
More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding box, eliminate duplicate detections, and rescore the box based on other objects in the scene.
YOLO is refreshingly simple. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes.YOLO trains on full images and directly optimizes detection

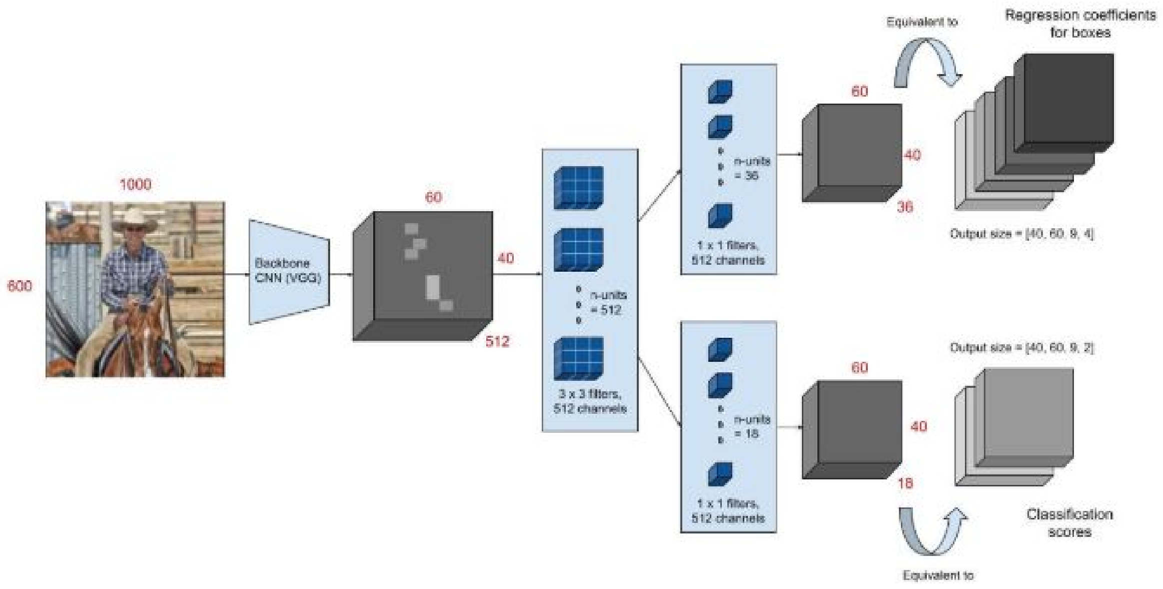
performance. This unified model has several benefits over traditional methods of object detection.

## 4. Proposed Work:

### Faster-RCNN:

The most widely used state of the art version of the R-CNN family — Faster R-CNN was first published in 2015. In the R-CNN family of papers, the evolution between versions was usually in terms of computational efficiency (integrating the different training stages), reduction in test time, and improvement in performance (mAP). These networks usually consist of — a) A region proposal algorithm to generate “bounding boxes” or locations of possible objects in the image; b) A feature generation stage to obtain features of these objects, usually using a CNN; c) A classification layer to predict which class this object belongs to; and d) A regression layer to make the coordinates of the object bounding box more precise.

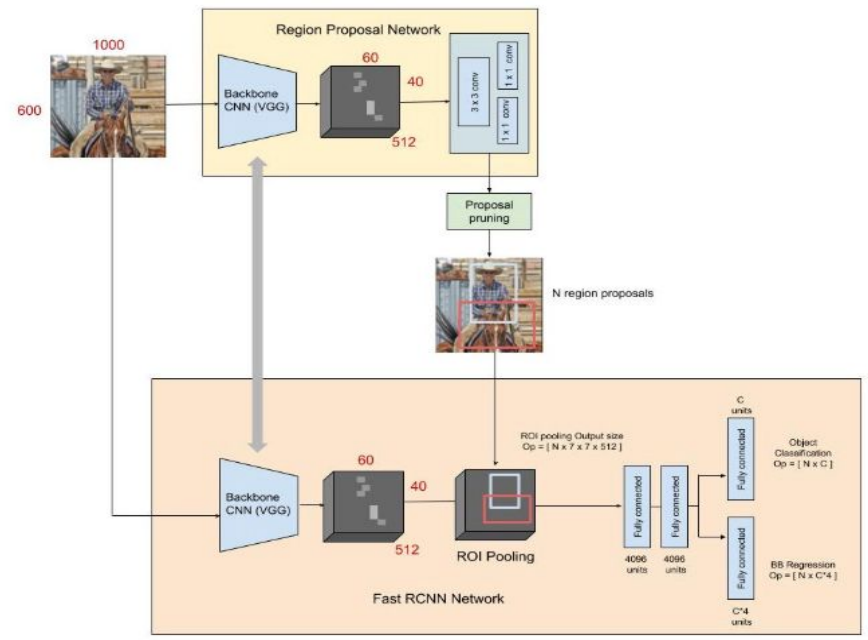
The Faster R-CNN uses another convolutional network (the RPN) to generate the region proposals. This not only brings down the region proposal time from 2s to 10ms per image but also allows the region proposal stage to share layers with the following detection stages, causing an overall improvement in feature representation.


Equivalent to
Regression coefficients for boxes
1000
600
Backbone CNN (VGG)
60
40
512
n-units = 512
3 x 3 filters, 512 channels
n-units = 36
1 x 1 filters, 512 channels
60
40
36
Output size = [40, 60, 9, 4]
n-units = 18
1 x 1 filters, 512 channels
60
40
18
Output size = [40, 60, 9, 2]
Classification scores
Equivalent to

The architecture of the region proposal network

**Architecture of Faster RCNN:**

The region proposal network (RPN) starts with the input image being fed into the backbone convolutional neural network. The input image is first resized such that it's shortest side is 600px.For every point in the output feature map, the network has to learn whether an object is present in the input image at its corresponding location and estimate its size.As the network moves through each pixel in the output feature map, it has to check whether these *k* corresponding anchors spanning the input image actually contain objects, and refine these anchors' coordinates to give bounding boxes as "Object proposals" or regions of interest. First, a 3 x 3 convolution with 512 units is applied to the backbone feature map, to give a 512-d feature map for every location. This is followed by two sibling layers: a 1 x 1 convolution layer with 18 units for object classification, and a 1 x 1 convolution with 36 units for bounding box regression. The 36 units in the regression branch give an output of size (H, W, 36). This output is used to give the 4 regression coefficients of each of the 9 anchors for every point in the backbone feature map (size: H x W). These regression coefficients are used to improve the coordinates of the anchors that contain objects.



The Faster R-CNN architecture consists of the RPN as a region proposal algorithm and the Fast R-CNN as a detector network.
**YOLO (You Only Look Once)**:

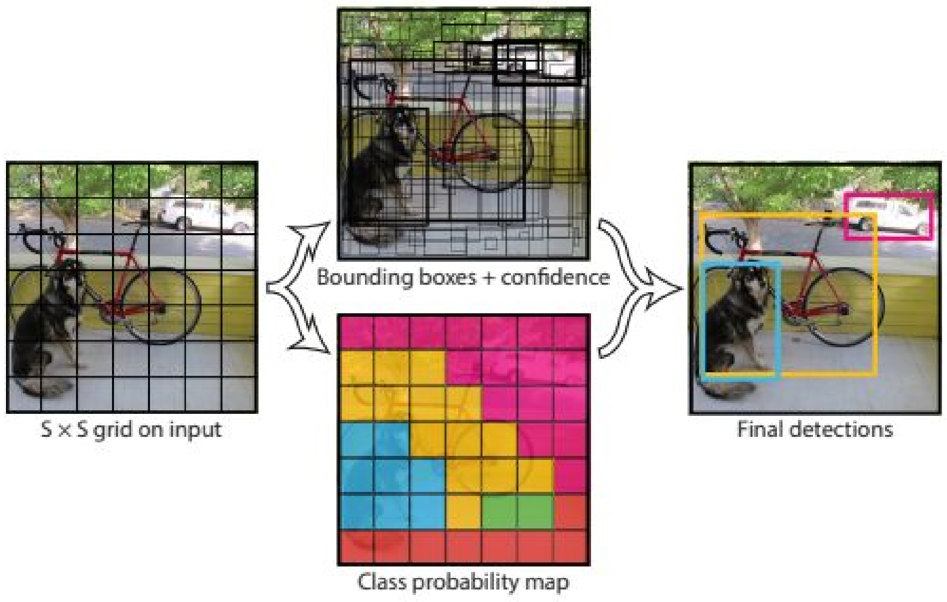
In YOLO we unify the separate components of object detection and extraction into one single neural network. This network uses the entire image for feature extraction and to predict the bounding boxes.  It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image.

This system divides the input image into an S x S grid.If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores show how accurate or how confident the model is that there is an object in the box and also how accurate it thinks the box is that it predicts.Formally we define confidence as Pr(Object) ∗ IOU truth pred. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.

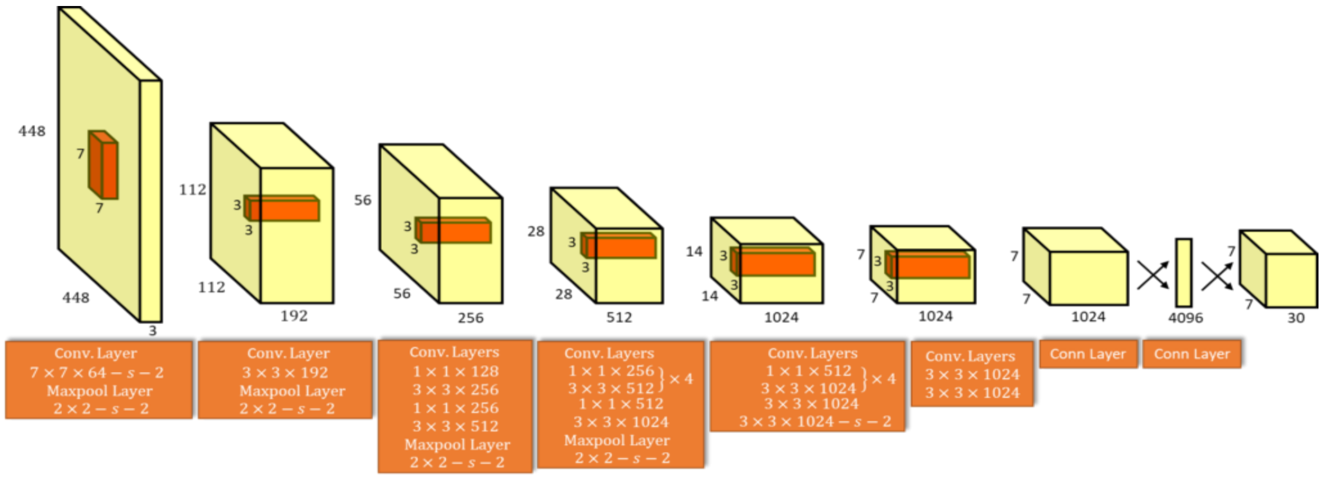
Each grid cell also predicts C conditional class probabilities, Pr(Classi |Object). At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}} = \Pr(\text{Class}_i) * \text{IOU}_{\text{pred}}^{\text{truth}}$$

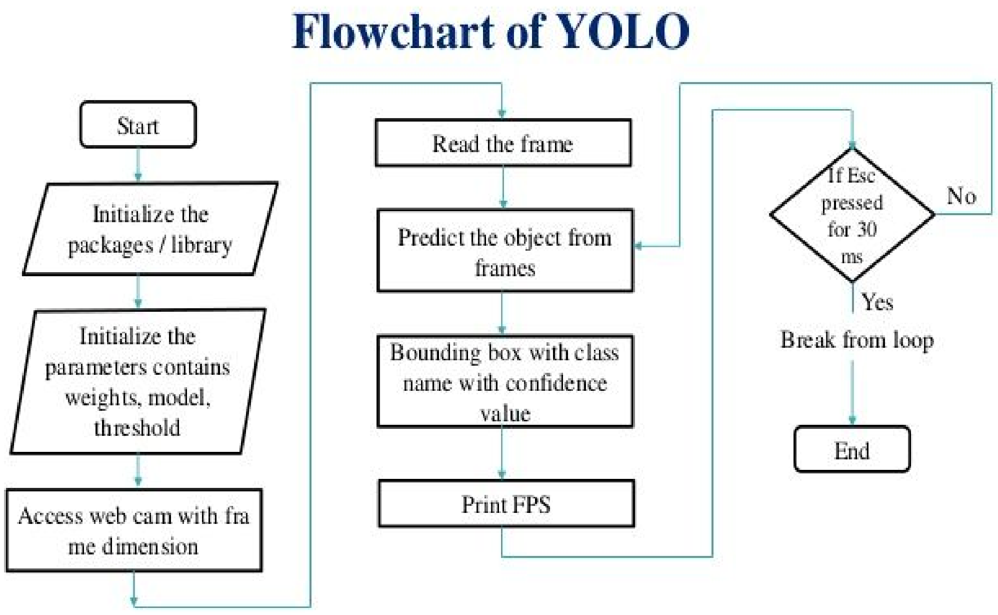
which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.



**Network design of YOLO:**



The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.The network has 24 convolution layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection. The final output of our network is the 7 × 7 × 30 tensor of predictions.



Flowchart of YOLO
Start
Initialize the packages / library
Initialize the parameters contains weights, model, threshold
Access web cam with fra me dimension
Read the frame
Predict the object from frames
Bounding box with class name with confidence value
Print FPS
If Esc pressed for 30 ms
No
Yes
Break from loop
End

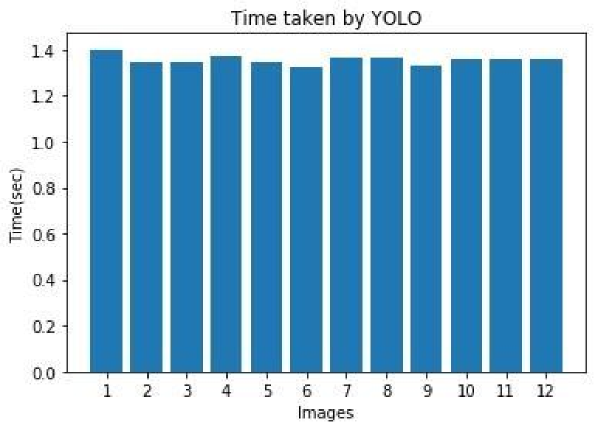
**Training :**

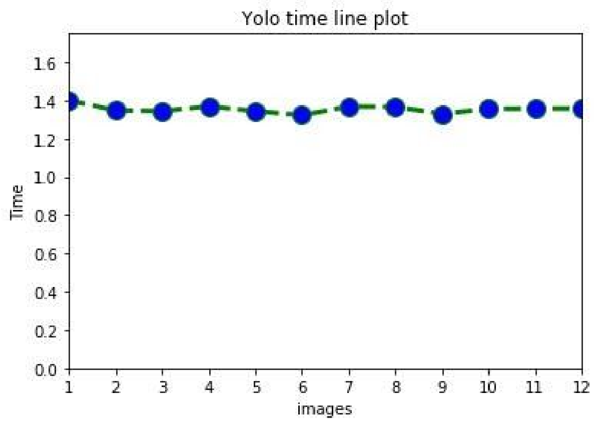
In YOLO, we pretrain our convolutional layers on the ImageNet 1000-class competition dataset. For pretraining we use the first 20 convolutional layers followed by an average-pooling layer and a fully connected layer. Then train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo. We use the Darknet framework for all training and inference.

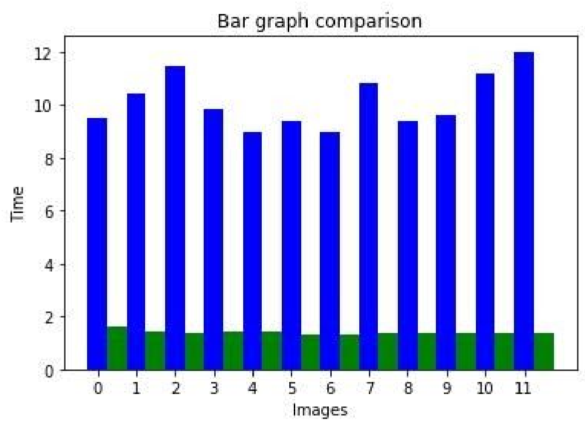
Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1. We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1. We use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation:

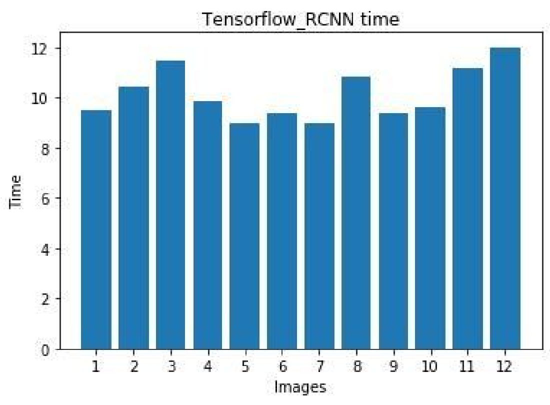
$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases}$$

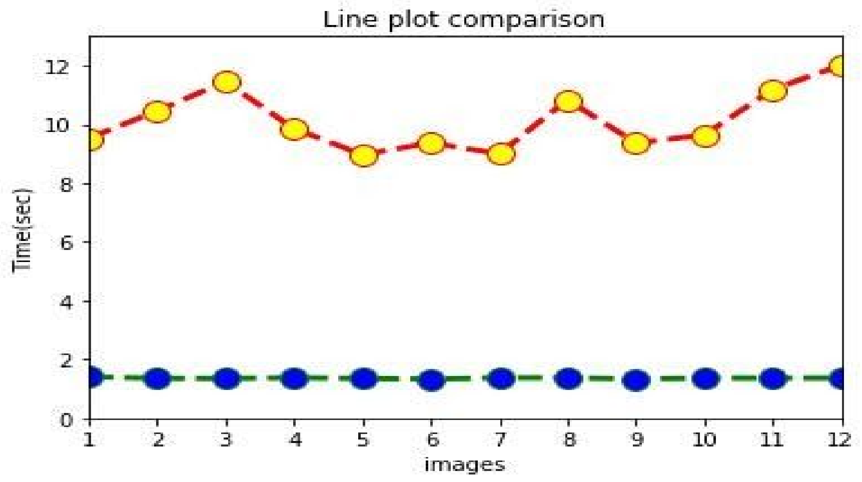
## 5. Evaluation and Results analysis:











It is clearly seen from the graphs that a tensorflow based RCNN object detection algorithm takes more time than the YOLO algorithm.

For recording the time Python's time() module is used.A start.time() a counter is given before the prediction of the model starts in both the algorithms codes.A end.time() counter is given when the object detection is done and the time is recorded.

A for loop is implemented in both the algos to input the 12 different images taken for comparing time efficiency of the algorithms.This time is appended in a list 'ti' for YOLO & 'time' for RCNN and is used to plot the graphs shown above.

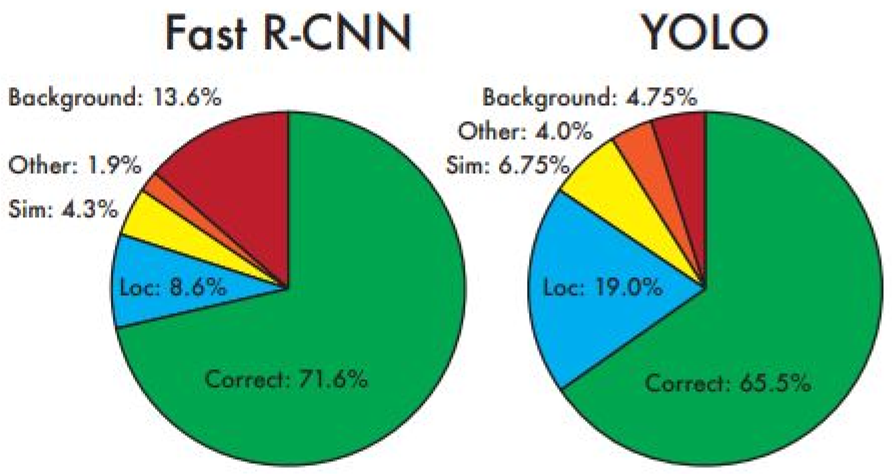
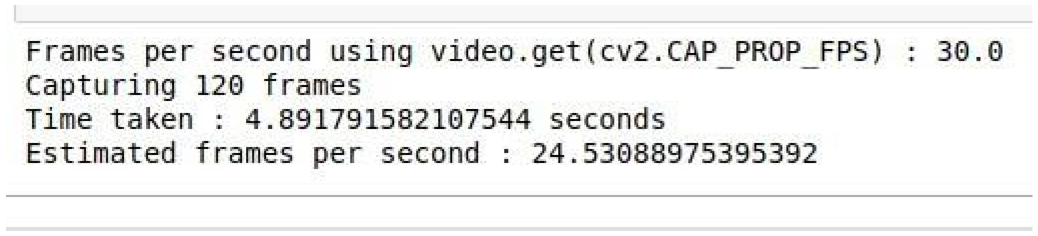


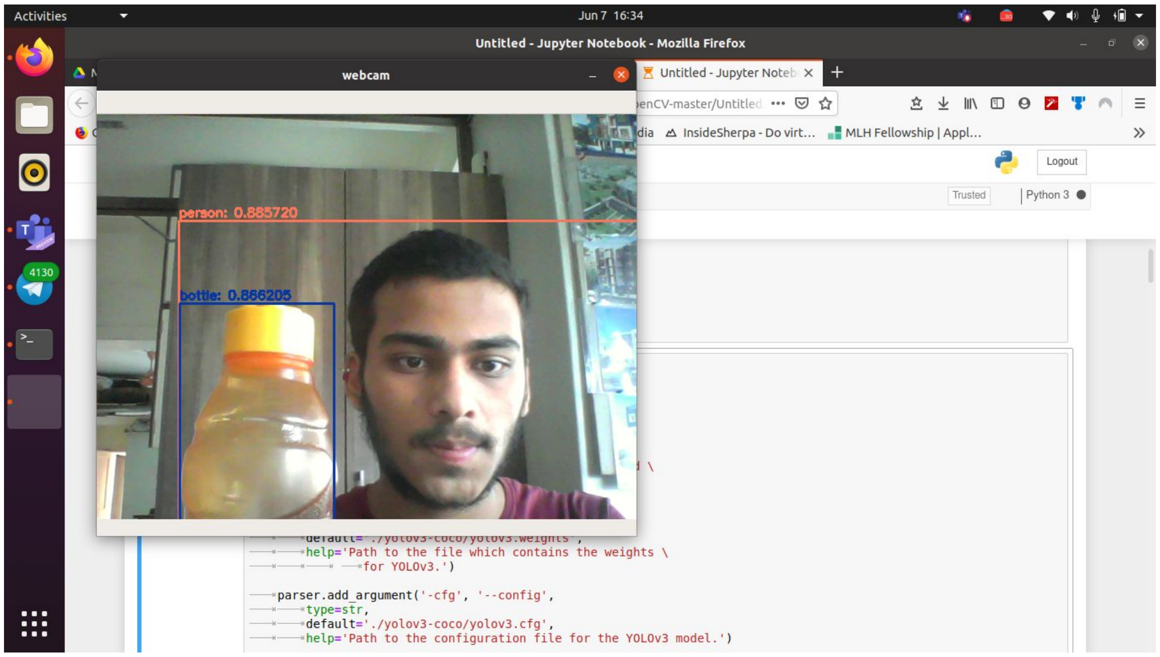
**Figure 4: Error Analysis: Fast R-CNN vs. YOLO** These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

## Evaluating Real-Time object detection in YOLO:

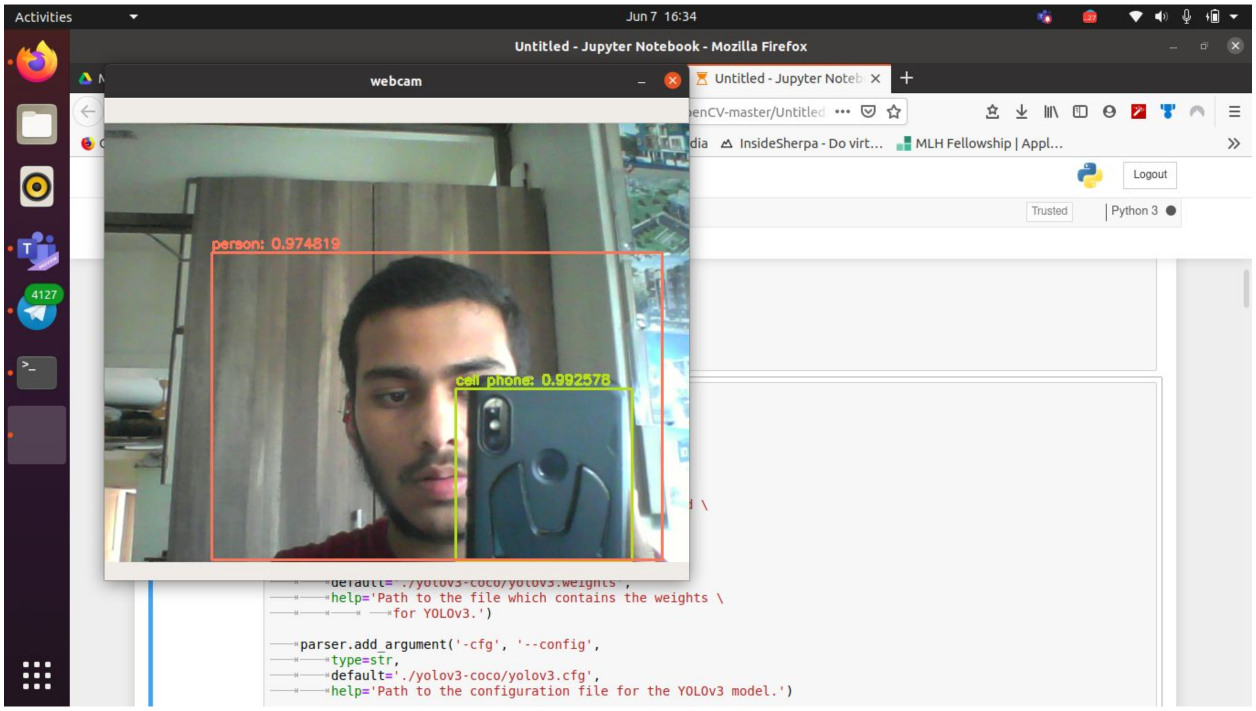


```
Frames per second using video.get(cv2.CAP_PROP_FPS) : 30.0
Capturing 120 frames
Time taken : 4.891791582107544 seconds
Estimated frames per second : 24.53088975395392
```

Shown above is the result of FPS obtained from the Real Object detection of the Yolo algorithm.Time taken is around 1-2 secs to detect objects like ‘person’,’bottle’,’cellphone’.



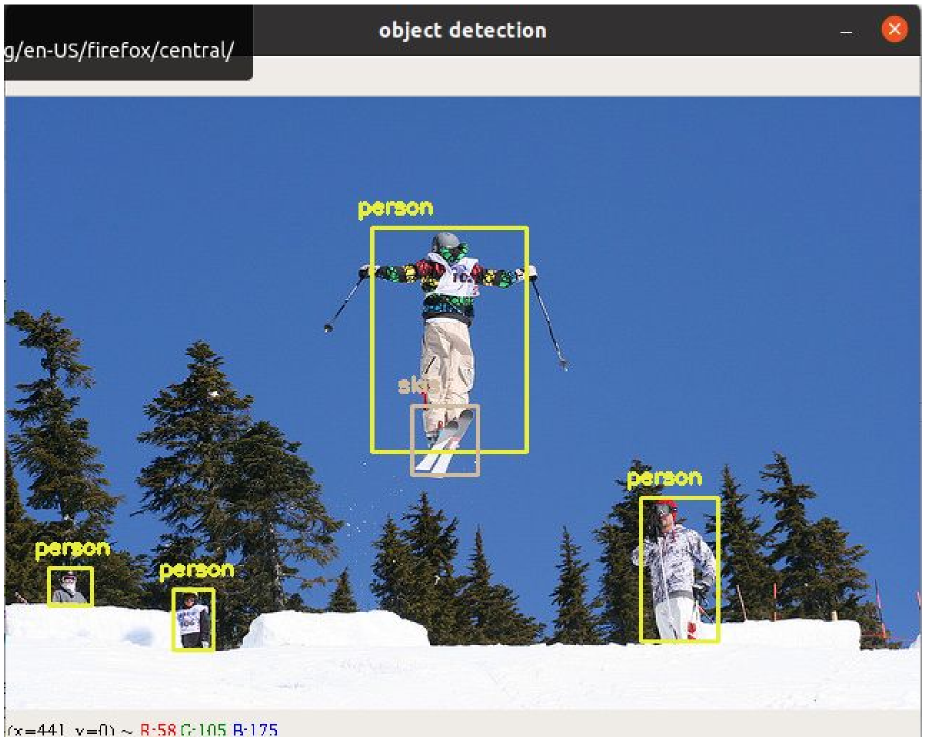
```
        default='./yolov3-coco/yolov3.weights',
        help='Path to the file which contains the weights \
                for YOLOv3.')

    parser.add_argument('-cfg', '--config',
        type=str,
        default='./yolov3-coco/yolov3.cfg',
        help='Path to the configuration file for the YOLOv3 model.')
```

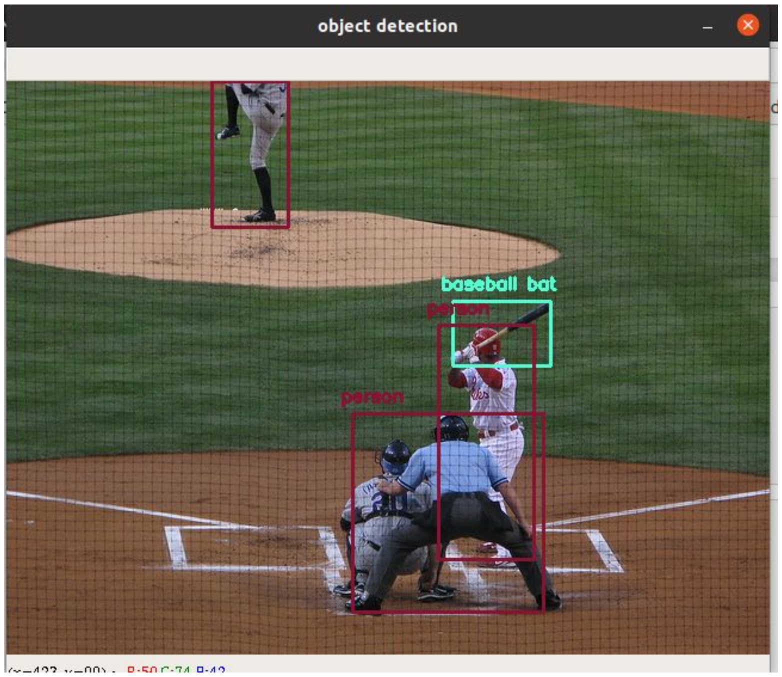


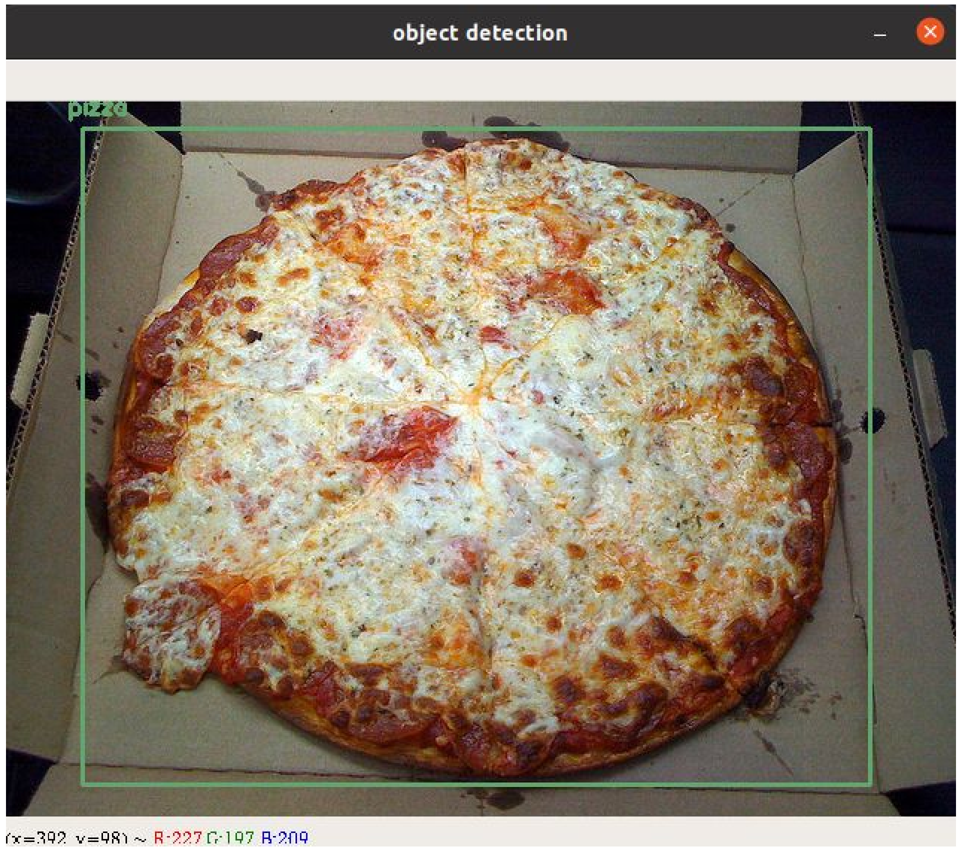
```
        default='./yolov3-coco/yolov3.weights',
        help='Path to the file which contains the weights \
                for YOLOv3.')

    parser.add_argument('-cfg', '--config',
        type=str,
        default='./yolov3-coco/yolov3.cfg',
        help='Path to the configuration file for the YOLOv3 model.')
```

It’s clear that YOLO is more time optimised and also more accurate than RCNN. Faster RCNN accuracy is near to the YOLO model but is significantly slower than YOLO.

The time taken by RCNN is exponentially larger than the time taken by the YOLO model.The graphs are the proofs.







The links to the jupyter notebooks of both the algorithms are given in references.

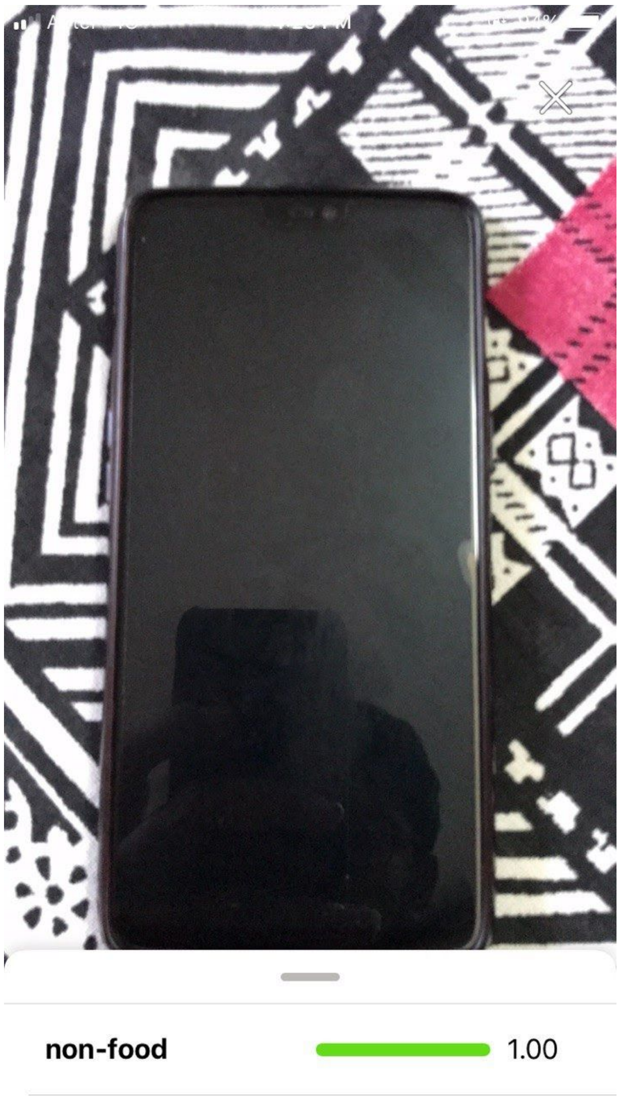
In addition to YOLO and RCNN the IBM Watson API is implemented on an iOS app to provide more insights in the object detection area.Following are the screenshots for it.

**IBM Watson Bluemix API:**

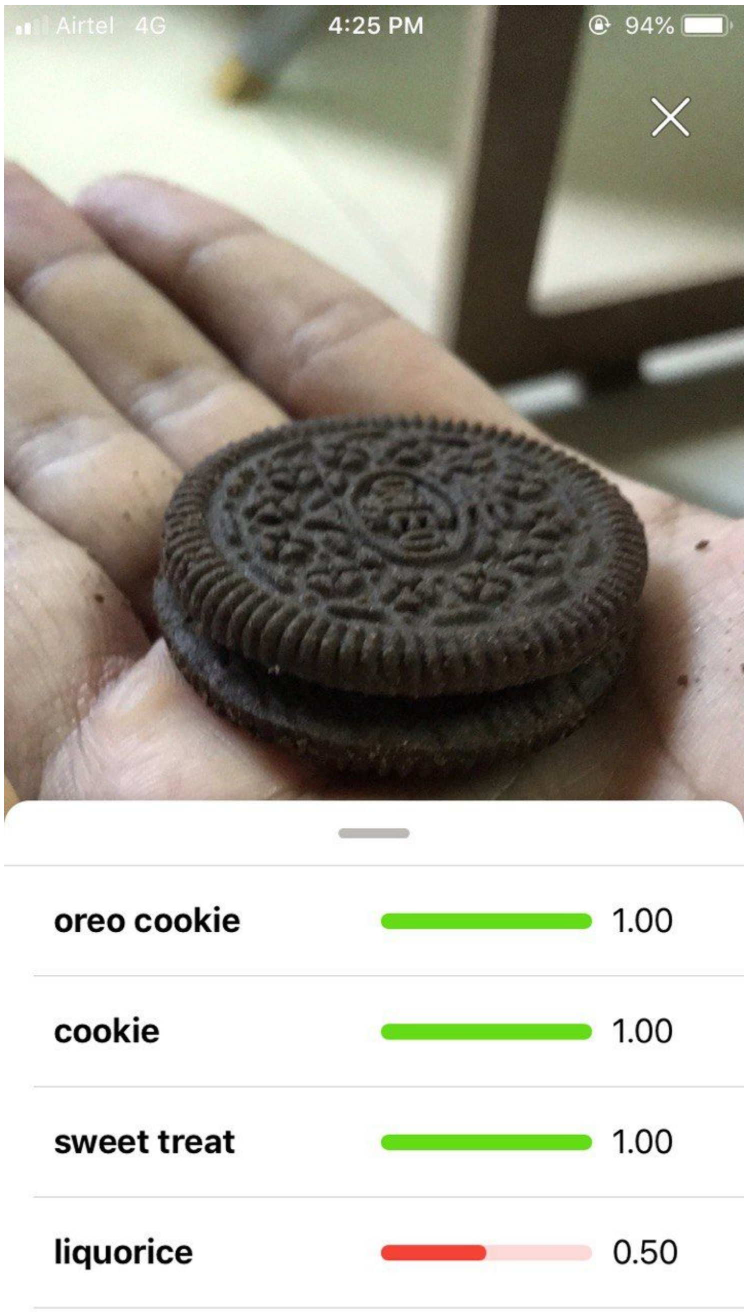
To Compare the performance fair with YOLO we also have made an app using IBM Watson Blue mix API which is similar to Google vision API. This API highly depends on Network connectivity hence is not reliable but the accuracy of this classifier is better then YOLO but not in real time.

Here are some Screenshots of implementation.

**Detecting Food Items:**

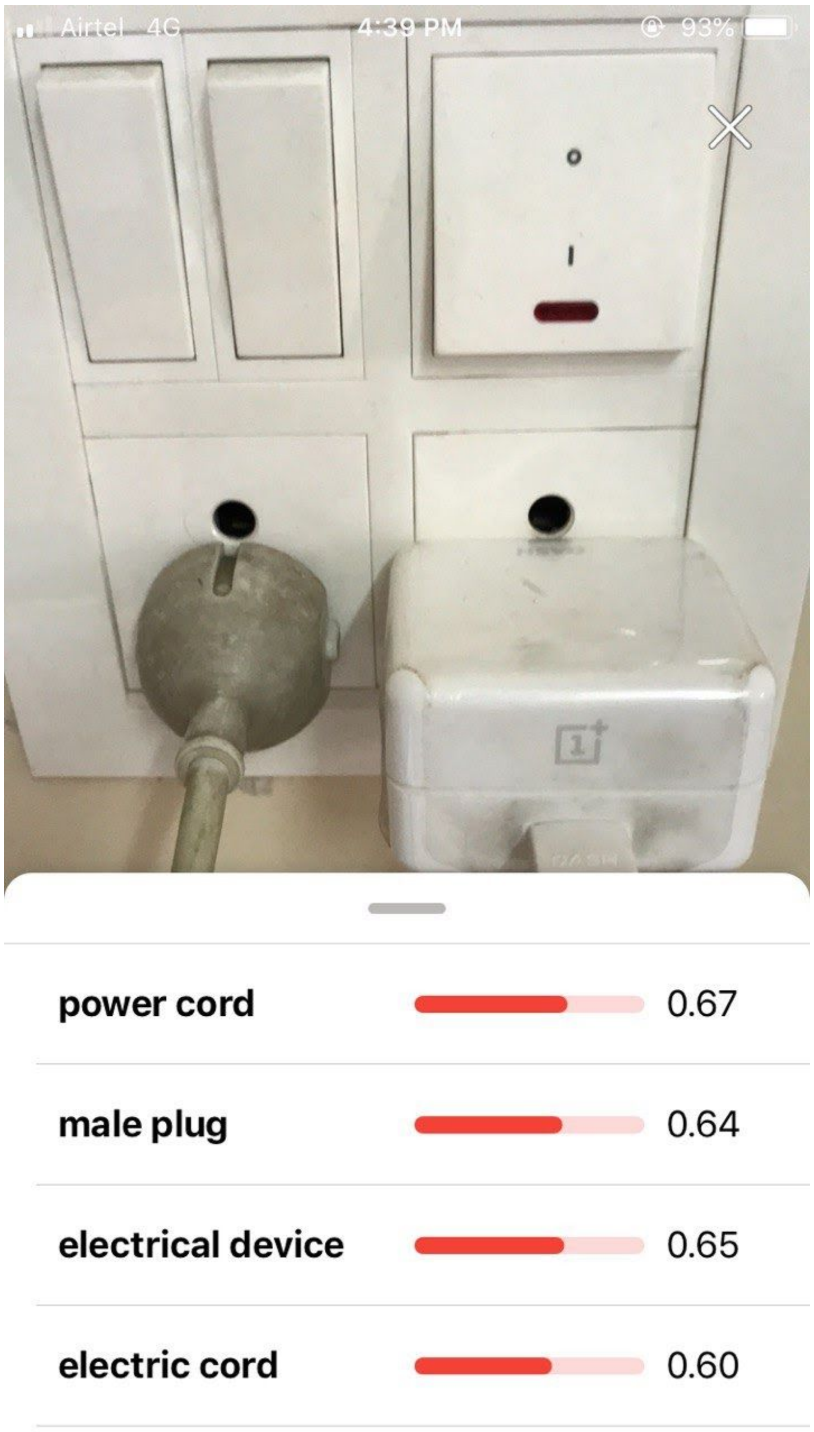


## Detecting Food items:



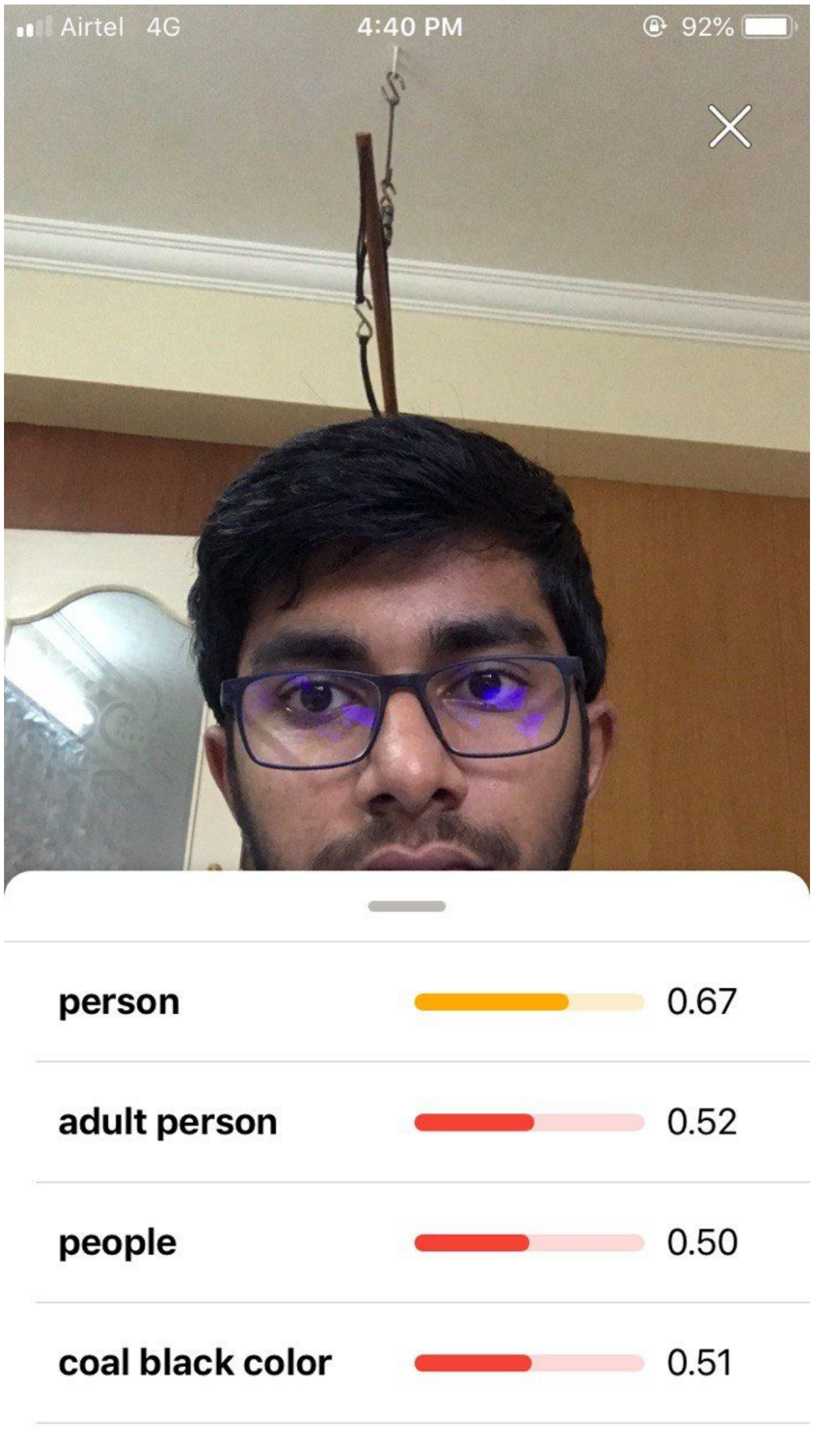
| | |
|---|---|
| oreo cookie | 1.00 |
| cookie | 1.00 |
| sweet treat | 1.00 |
| liquorice | 0.50 |

## Detecting General Items:



| | |
|---|---|
| power cord | 0.67 |
| male plug | 0.64 |
| electrical device | 0.65 |
| electric cord | 0.60 |

## Detecting Faces:



| | |
|---|---|
| person | 0.67 |
| adult person | 0.52 |
| people | 0.50 |
| coal black color | 0.51 |

## 6. Tabular Comparison with Existing Work

| | YOLO | Primitive R-CNN | Google Vision API | IBM Watson Bluemix API |
|---|---|---|---|---|
| **Realtime Performance** | Less performance hungry then other algorithms | CPU usage max out while running it. | No performance outage but network speed comes to matter | Same as Google Vision API |
| **FPS(Frames per second)** | Usually 30-50 FPS observed with decent hardware. But can increase with good GPU's | Hardly 2-5 FPS observed due to deep and more complex neural network | Depends on network availability. Generally less than 25 FPS | Depends on network availability |
| **Memory Complexity** | Use more as compared to API's but less than R-CNN | Heavy memory logging issue found which even slows down the host OS. | No memory overhead since API calling only depends on network | No memory overhead since API calling only depends on network |
| **Performance** | Performance very good and can be used for commercially application like autonomous driving (Tesla cars) | Due to poor performance it is not viable to use for real time application. But it's good start for beginners since easy to learn concepts | Again it depends on network performance like ping, data speed. But after successful deployment of 5G this will be the Industry standards. | Again it depends on network performance like ping, data speed. But after successful deployment of 5G this will be the Industry standards. |

| | | | | |
|---|---|---|---|---|
| **Time Complexity** | Takes less time to compute. | Takes more time to commute since more performance hungry | Depends on network as well as server capacity to handle the request. | Depends on network as well as server capacity to handle the request. |
| **Reliability** | Most reliable of all since it is independent of network usage | Reliable but due to poor performance, not practical for use. | Not reliable for real time since the network can fluctuate anytime.<br>If a stable network is provided then there can be the possibility of the server not able to handle a lot of request peak hours. | Not reliable for real time since the network can fluctuate anytime.<br>If a stable network is provided then there can be the possibility of the server not able to handle a lot of request peak hours. |
| **Efficiency** | Efficient if good hardware is provided. | Not efficient | Efficient provided good network | Efficient provided good network |
| **Cost** | Cost effective given that use case satisfies the cost. | Costly since very expensive hardware is required for computation | Cheap since Google Vision API (GCP credit) $300 free per month | Cheap since Watson API is bundled with other services like NLP which can be integrated. |

The above comparison boils down to the particular use case. For more reliability you may opt for YOLO, for cheaper cost you may go with API based solutions like Google Vision API which uses TensorFlow and IBM Watson blue mix API. But when it comes to learning the basics concept then it is recommended to start with basics of R-CNN which helps us to dive deeper into the world of Image Processing.

But we need to also consider the factor that the growing future of 5G network will change the game forever where the speed and performance will be so fast that one can rely on the API’s for the real time object detection but also we still need to consider risk of network fluctuation which will then boil down to use case where people should priorities speed over reliability.

**Conclusion:**

YOLO is a unified model for object detection. This model is simple to construct and can be trained directly on full images. Unlike classifier-based approaches like RCNN and its other family, YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly. Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection.

YOLO is a strong step towards closing the dataset size gap between de- tection and classification.

Many of our techniques generalize outside of object detection. Our WordTree representation of ImageNet offers a richer, more detailed output space for image classification. Dataset combination using hierarchical classification would be useful in the classification and segmentation domains. Training techniques like multi-scale training could provide benefits across a variety of visual tasks.

For future work we hope to use similar techniques for weakly supervised image segmentation. We also plan to improve our detection results using more powerful match- ing strategies for assigning weak labels to classification data during training. Computer vision is blessed with an enor- mous amount of labelled data. We will continue looking for ways to bring different sources and structures of data together to make stronger models of the visual world.

For the next coming decades we need to consider the upcoming technologies like 5G, quantum computing which will be a giant leap for the field of AI. With the help of quantum computing's performance and with the help of 5G connectivity this project will give scope in many fields like remote medical operation performed by doctors (AI in healthcare), Autonomous driving etc.

**References:**

1. Du, Juan. (2018). Understanding of Object Detection Based on CNN Family and YOLO. Journal of Physics: Conference Series. 1004. 012029. 10.1088/1742-6596/1004/1/012029. https://www.researchgate.net/publication/324754316_Understanding_of_Object_Detection_Based_on_CNN_Family_and_YOLO

2. S. Wang, L. Niu and N. Li, "Research on Image Recognition of Insulators Based on YOLO Algorithm," 2018 International Conference on Power System Technology (POWERCON), Guangzhou, 2018, pp. 3871-3874, doi: 10.1109/POWERCON.2018.8602149. https://ieeexplore.ieee.org/document/8602149

3. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhad University of Washington, Allen Institute for AI, Facebook AI Research ONR N00014-13-1-0720, NSF IIS-1338054, and The Allen Distinguished Investigator Award.

   https://arxiv.org/pdf/1506.02640.pdf

4. Joseph Redmon Ali Farhadi: University of Washington, Allen Institute for AI
   https://arxiv.org/pdf/1612.08242v1.pdf

5. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition(CVPR), 2014 IEEE Conference on, pages 580–587. IEEE,2014.

**Reference Links:**

1. https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

2. https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006

3. https://arxiv.org/pdf/1612.08242v1.pdf

4. https://arxiv.org/pdf/1506.02640v5.pdf

5. https://pjreddie.com/darknet/yolo/

6. https://github.com/pjreddie/darknet

**Link for Code:**

https://github.com/sudnyeshtalekar/YOLO

https://colab.research.google.com/drive/1QvlazVKw6jdF9nGwOOiF1EChhhJntx2w?usp=sharing

**Link for iOS App:**

www.github.com/pateldevang

## Appendix:

### A. Appendix for Acronyms:

| | |
|---|---|
| DL | Deep Learning |
| CNN | convolutional neural network |
| R-CNN | Region-based Convolutional Neural Networks |
| YOLO | You Only Look Once |
| GCP | Google Cloud Platform |
| DIP | Digital Image Processing |

### B. Appendix for Individual Contribution Details in Group:

| **Sr. No** | **Registration Number** | **Role and Responsibility** |
|---|---|---|
| 1 | 18BEE0201 | YOLO Implementation |
| 2 | 18BCE0791 | Research and documentation |
| 3 | 18BCE0809 | Comparison and analysis of result |
| 4 | 18BCE0811 | Research and documentation |