Project Submission – Parth Sakhare

Problem Statement:

- Build A Personal Safety Equipment Detection System

Dataset Details:

- Hardhat/ head detection dataset
- Dataset annotation format: Pascal VOC (XMLs)
- Dataset Size:

  →Data with Annotations: 4,750

  →Data without Annotations (Test): 250

- Dataset classes of interest:

  → Head

  → Hardhat

Goal:

1. Here our goal is to build a vision-based safety equipment detection system, which can be implemented for real time purposes.
2. As a safety equipment here the vision-based system should process the image/frame(in case of videos) and detect whether there is helmets in the image or only heads(without helmet).

Solution Implementation Guide:

- Detailed study and implementation and correposding references can be found from the git repo- https://github.com/parthsak/A-Personal-Safety-Equipment-Detection-System

Guide to Zip Submission:

- The submission zip file contains following files

Submission.pdf – Contains documentation regarding the assignment
videoplayback_result_final.mp4: Output video containing helmet detected frames(is playable in chrome as mentioned)
Test_annotations: This folder contains annotations of test images in Pascal Voc xml format.

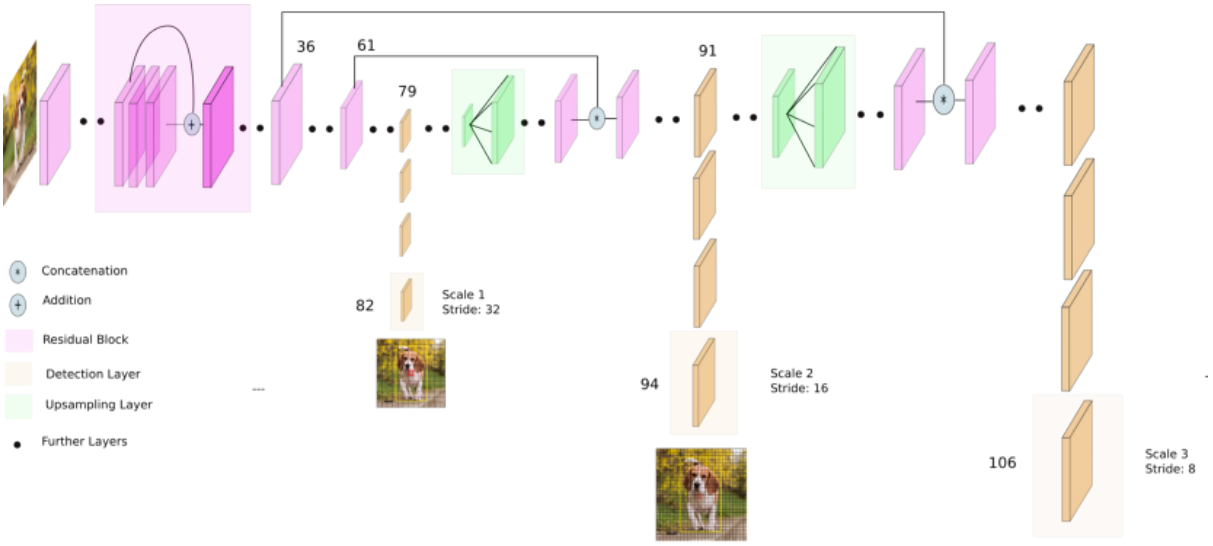
YoloV3

YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. The YOLO machine learning algorithm uses features learned by a deep convolutional neural network to detect an object. Versions 1-3 of YOLO were created by Joseph Redmon and Ali Farhadi, and the third version of the YOLO machine learning algorithm is a more accurate version of the original ML algorithm.

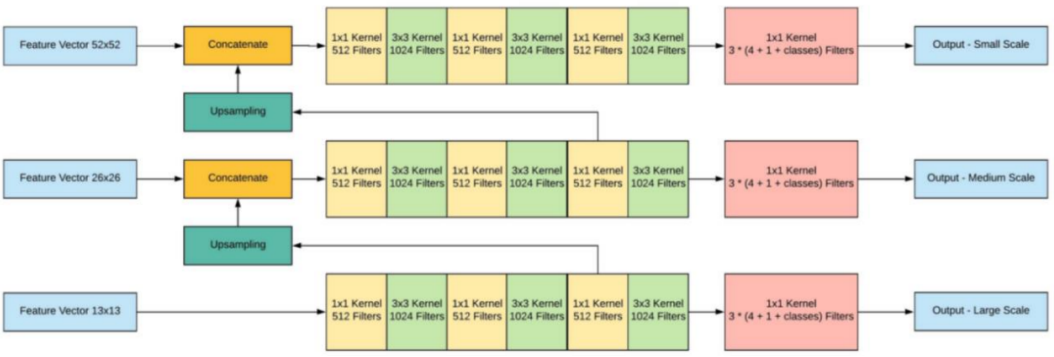
Yolo Working:

YOLO is a Convolutional Neural Network (CNN) for performing object detection in real-time. CNNs are classifier-based systems that can process input images as structured arrays of data and recognize patterns between them. YOLO has the advantage of being much faster than other networks and still maintains accuracy.

It allows the model to look at the whole image at test time, so its predictions are informed by the global context in the image. YOLO and other convolutional neural network algorithms “score” regions based on their similarities to predefined classes. High-scoring regions are noted as positive detections of whatever class they most closely identify with. Yolo-v2 was less accurate than algorithms like RetinaNet, and SSD outperformed it in terms of accuracy. It still, however, was one of the fastest. But that speed has been traded off for boosts in accuracy in YOLO v3. This has to do with the increase in **complexity of underlying architecture called Darknet.**
First, YOLO v3 uses a variant of Darknet, which originally has 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3. This is the reason behind the slowness of YOLO v2.



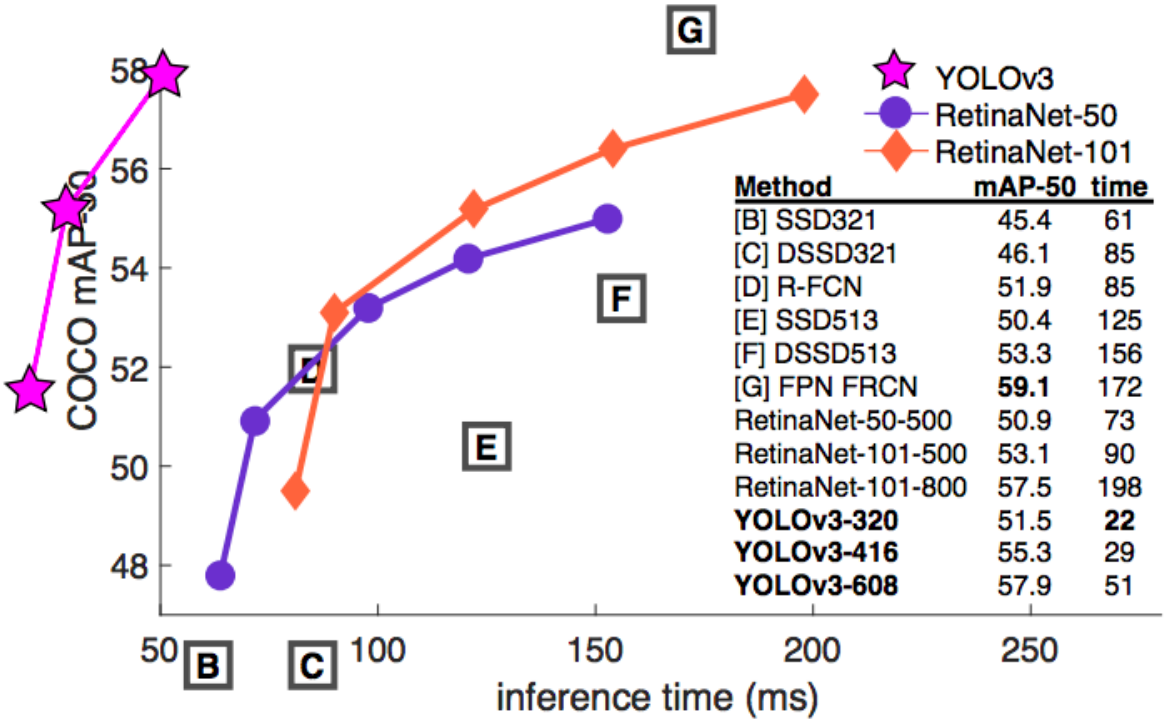
YOLO v3 network Architecture



Detection at three Scales

The YOLO v3 architecture boasts of residual skip connections, and upsampling. **The most salient feature of v3 is that it makes detections at three different scales.** YOLO is a fully convolutional network and its eventual output is generated by applying a 1 x 1 kernel on a feature map. In YOLO v3, **the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network.**

Why Yolo V3?



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

From above figure we can see that, maybe mean average precision of Yolov3 is not good as compared to Retinanet, but FPS/real-time inference time is better for yoloV3 as compared to them. That's why we have used YoloV3 here

Approach:

1. We implemented the yolov3 implementation pipeline using Tensorflow2.
2. We made tfrecords from the image and annotation pairs
3. Train and validation split was done(ratio- 0.95:0.05)

4. Build the model from scratch(referenced from other repos, using darknet block, darknet model, Yoloconv, yolov3, iou measurement, image transformation, bounding box, yolo loss calculation)
5. Load the YoloV3 model trained on Ms-coco dataset with 80 classifiers(while training, pass class 2 of the model)
6. Then, processed the images to YoloV3 model for training
7. Model was trained in fit mode using transfer learning through darknet-mode(30 epochs trained with some usual callbacks, having batch-size **4**)
8. Technical parameters:

```
# image dimension
image_size = 416
# num of classes for the dataset
num_classes = 2
# status of transfer learning
transfer = 'darknet'

#useful in transfer learning with different number of classes
weights_num_classes = 80
# weights file path
weights = 'HardHead_Dataset/models/yolov3.tf'
# define learning rate
learning_rate = 1e-03

#Use if wishing to train with more than 1 GPU.
multi_gpu = False
# batch size
batch_size = 4
# epochs
epochs = 30
```

9. After training is done, we processed the video frames using opencv and create output video using opencv videowriter.
10. Then build the xml files/annotation files to store the object detection result for the test-images.

Future work:

Train with more data, and more efficient and try with tiny YoloV3, RetinaNet, and SSD300.

References:

- https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
- https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e
- https://github.com/zzh8829/yolov3-tf2
- https://arxiv.org/abs/1804.02767?e05802c1_page=1