Project Submission – Sayan Hazra

Project Title: 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

Goal:

- Here our goal is to build a vision-based safety equipment detection system, which can be implemented for real time purpose.
- As a safety equipement 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: We have implemented the vision based detection system with the help of YoloV3 model.

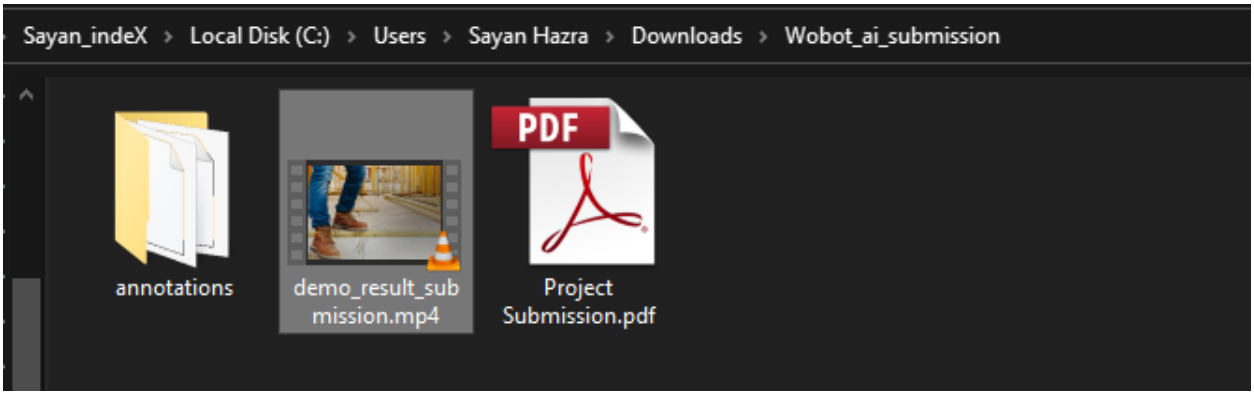
Solution Implementation Guide

1. Detailed study and implementation and correposding references can be found from the git repo- https://github.com/sayan0506/HardHead-Detection-for-Safety-Surveilance-using-YoloV3

2. The implementation pipeline can be found in colab notebook - https://colab.research.google.com/drive/1Z3Y62pOOKiU1cuIMkFWf5j2udOcW14N5?usp=sharing

Guide to Zip Submission:

1. The submission zip file contains following files



Project_submission.pdf – Contains documentation regarding the assignment

Demo_result.mp4: Output video containing helmet detected frames(is playable in chrome as mentioned)

Annotation: This folder contains annotations result of test images in Pascal Voc xml format.

Brief summary on YoloV3:

Introduction:

YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. YOLO uses features learned by a deep convolutional neural network to detect an object.

How Yolo works:

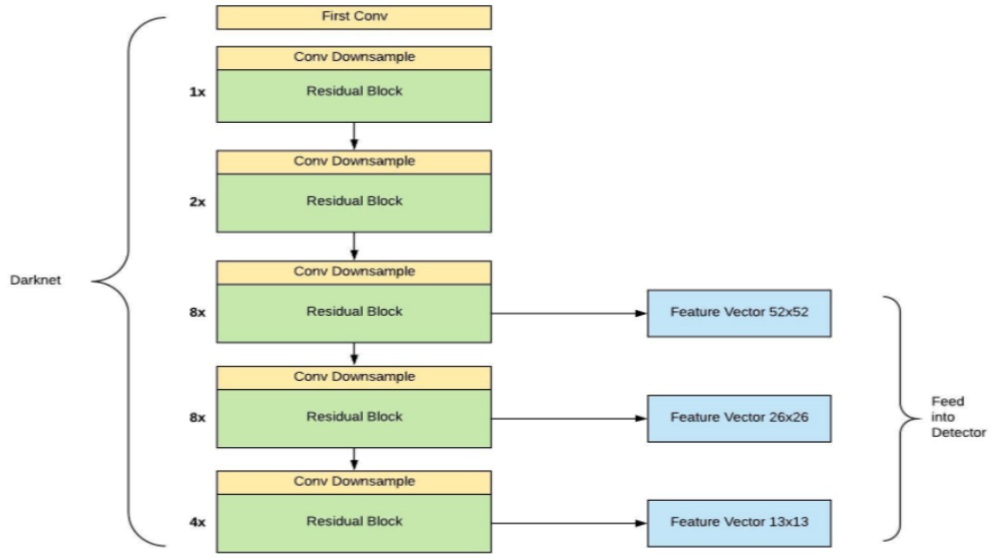
How does YOLOv3 work? (Overview)

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 identify patterns between them (view image below). 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.

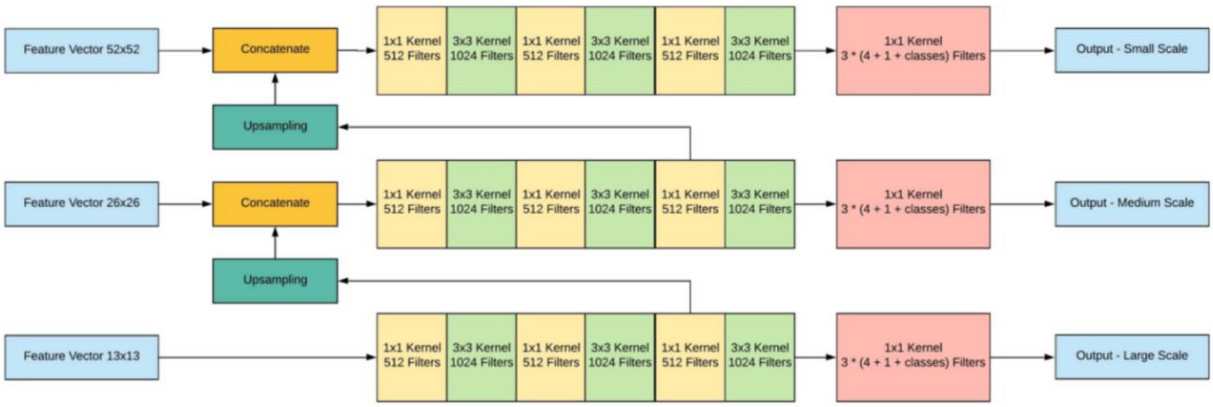
High-scoring regions are noted as positive detections of whatever class they most closely identify with. For example, in a live feed of traffic, YOLO can be used to detect different kinds of vehicles depending on which regions of the video score highly in comparison to predefined classes of vehicles.

In this Yolo3 pretrained-darknet53 is used as backend for feature extraction(without classification head). AS it was proved to be better backbone as compared to the others. Using Darknet53 YoloV3 implements the multiscale object detection in a efficient way.



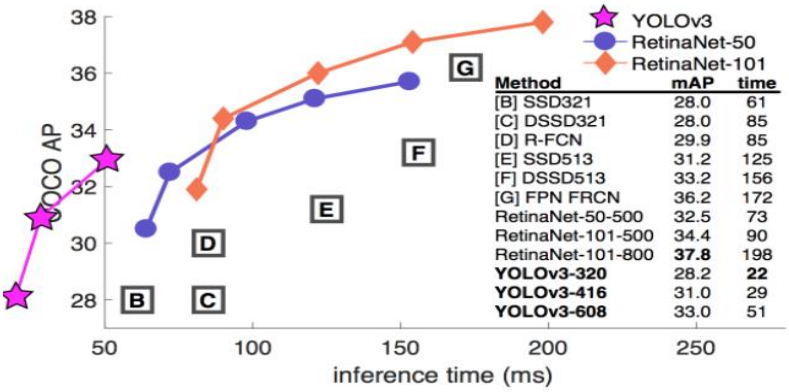
Above figure is the foundation of multi-scale detector.

Yolo Architecture diagram(from the perspective of dataflow)



In depth details can be found in the reference link.

Why YoloV3?



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. 4. 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 16)

8. Technical parameters-

```
 num of classes for the dataset
num_classes = 2
# status of transfer learning
transfer = 'darknet'
# specify num class for `weights` file if different,
#useful in transfer learning with different number of classes
weights_num_classes = 80
# weights file path
weights = '/content/drive/MyDrive/HardHead_Dataset/models/yolov3.tf'
# define learning rate
learning_rate = 1e-03
#'mode', 'fit', ['fit', 'eager_fit', 'eager_tf'],
#                  'fit: model.fit, '
#                  'eager_fit: model.fit(run_eagerly=True), '
#                  'eager_tf: custom GradientTape')
# further info on mode can be found from eager execution url - https://www
.tensorflow.org/guide/eager
mode = 'fit'
#Use if wishing to train with more than 1 GPU.
multi_gpu = False
# batch size
batch_size = 16
# 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, and SSD300.

Reference:

1. **https://viso.ai/deep-learning/yolov3-overview/**

2. **https://towardsdatascience.com/dive-really-deep-into-yolo-v3-a-beginners-guide-9e3d2666280e**

3. **https://github.com/zzh8829/yolov3-tf2**