2020

Internship Project Report

SUPRAGYA SHRESTHA
INTERN

SAMSUNG RESEARCH INSTITUTE NOIDA

Table of Contents

Introduction

Project Statement

To generate text results for human activities from videos.

Project Overview

In today's world, security is a big concern for everyone. The overgrowing use of CCTV has given rise to various problems, such as human surveillance personnel and storage space shortages. This problem is going to be a big problem for the future, and Machine Learning and especially Deep Learning can be used to solve this problem. Machine Learning models can be used to impersonate surveillance personnel, answer the first problem, and solve the storage space issue. Machine Learning models can be used to generate text detections for human activity detection, which I am trying to achieve here.

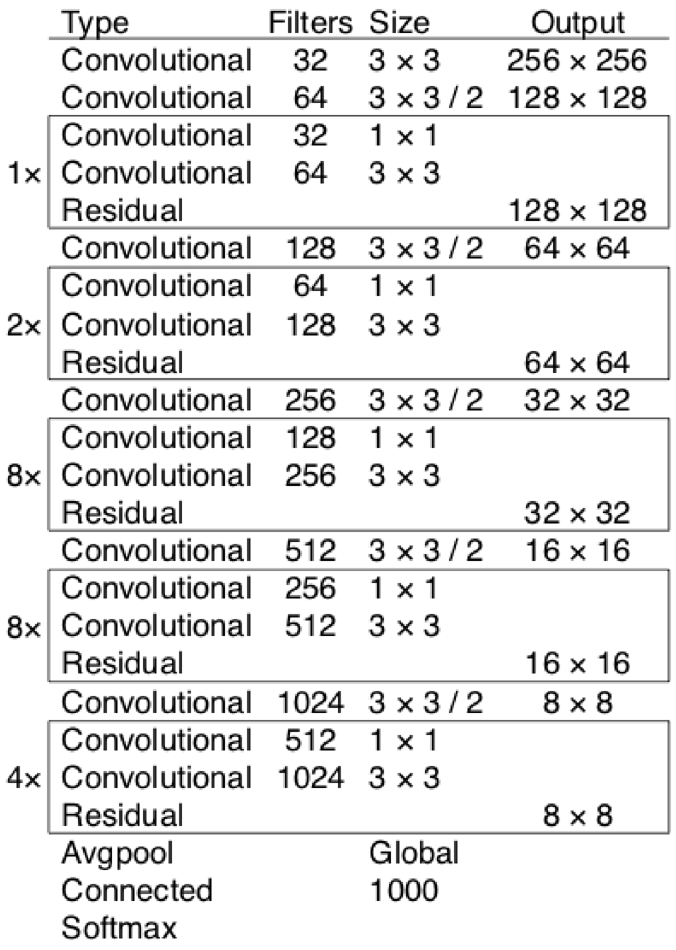
Most Activity Detection Machine Learning models present in today's world are typically more or less video classification models that take small videos with prelabelled human activity. They predict whether the video has a particular event or not. Also, the corresponding dataset for these models ranges in size of TBs, which is not possible for me to use with the hardware available with me. So I had to look for another approach to this problem. I finally came up with the idea of using the real-time object detection model Yolo which is typically used to detect objects in an image. However, it can also be used to identify actions if a dataset for human actions in the image format is available. I was able to find such a dataset, and so I have used this approach throughout my project.

Yolo

Yolo or You Only Look Once is one of the most powerful Object Detection Algorithms. Many of object detection systems need to go through the image more than one time to be able to detect all the objects in the image, or it has to go through two stages to detect the objects. YOLO doesn’t need to go through these boring processes. It only need to look once at the image to detect all the objects and that is why they chose the name (You Only Look Once) and that is actually the reason why YOLO is a very fast model.

So, it not only classifies the image into a category, but it can also detect multiple Objects within an Image. This Algorithm applies a single Neural network to the Full Image. It means that this network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN). In YOLO v3 paper, the authors present new, deeper architecture of feature extractor called Darknet-53. As its name suggests, it contains of 53 convolutional layers, each followed by batch normalization layer and Leaky ReLU activation. No form of pooling is used, and a convolutional layer with stride 2 is used to downsample the feature maps. This helps in preventing loss of low-level features often attributed to pooling.

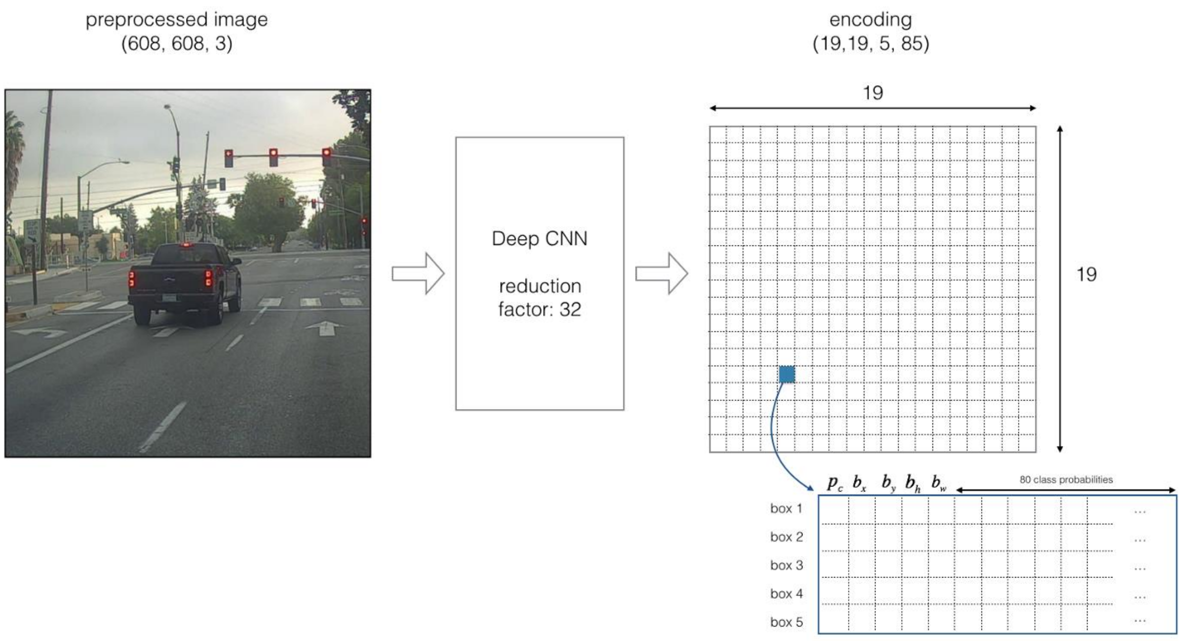


	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

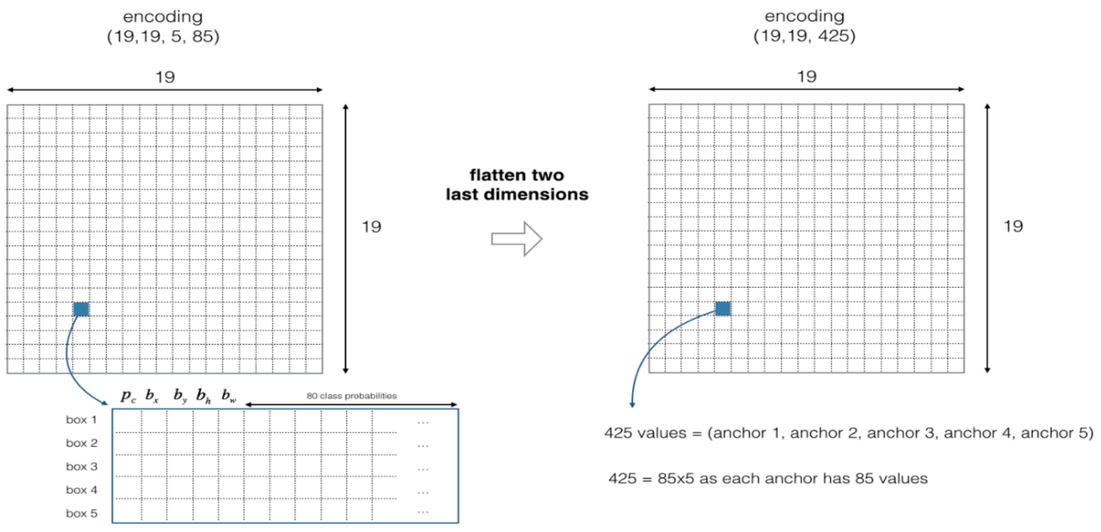
The network downsamples the image by a factor called the stride of the network. For example, if the stride of the network is 32, then an input image of size 416 x 416 will yield an output of size 13 x 13. Generally, stride of any layer in the network is equal to the factor by which the output of the layer is smaller than the input image to the network.

We can understand the working of the yolo algorithm by a simple example:

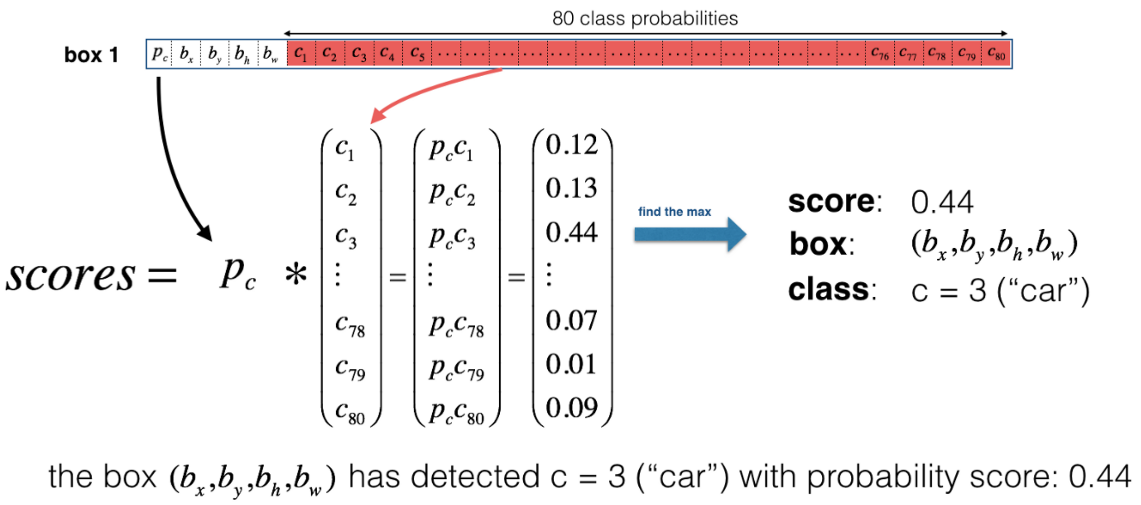
Suppose you have an image of dimension 608*608*3(3 represents an RGB image) and also that we are using 5 anchor boxes, one for confidence score and 4 other for bounding box representation. boxes. So we can think of the YOLO architecture as the following: IMAGE (m, 608, 608, 3) -> DEEP CNN -> ENCODING (m, 19, 19, 5, 85).



If the center/midpoint of an object falls into a grid cell, that grid cell is responsible for detecting that object. Since we are using 5 anchor boxes, each of the 19x19 cells thus encodes information about 5 boxes. Anchor boxes are defined only by their width and height. For simplicity, we will flatten the last two dimensions of the shape (19, 19, 5, 85) encoding. So the output of the Deep CNN is (19, 19, 425):


encoding
(19,19, 5, 85)
19
19
p_c b_x b_y b_h b_w
80 class probabilities
box 1
box 2
box 3
box 4
box 5
flatten two last dimensions
encoding
(19,19, 425)
19
19
425 values = (anchor 1, anchor 2, anchor 3, anchor 4, anchor 5)
425 = 85x5 as each anchor has 85 values

Now, for each box (of each cell) we will compute the following element wise product and extract a probability that the box contains a certain class.


80 class probabilities
box 1
p_c b_x b_y b_h b_w c_1 c_2 c_3 c_4 c_5 … c_{76} c_{77} c_{78} c_{79} c_{80}
find the max

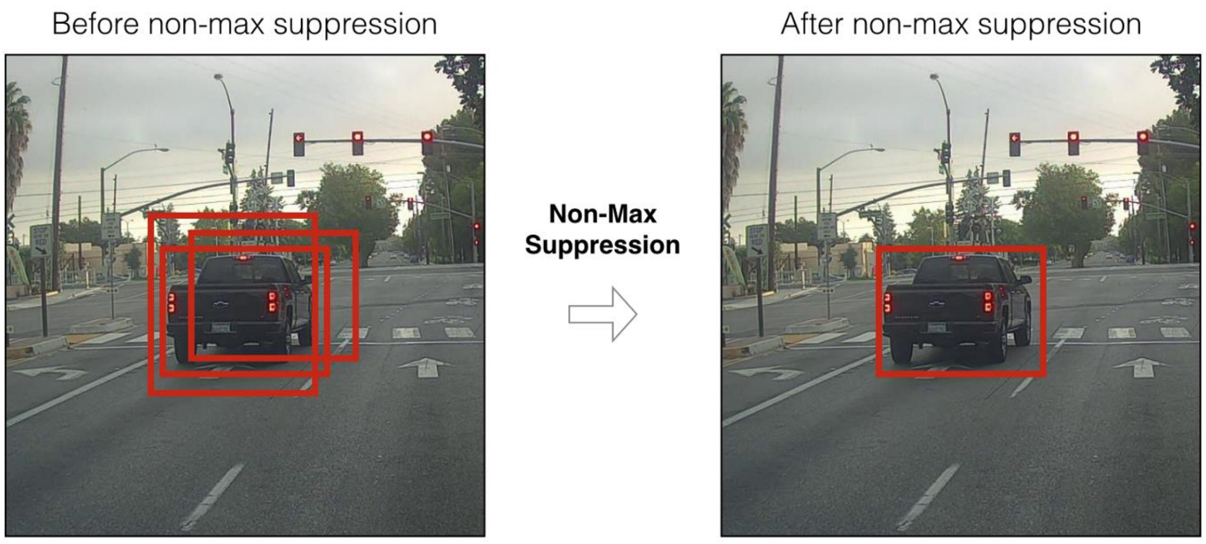
$$scores = p_c * \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{78} \\ c_{79} \\ c_{80} \end{pmatrix} = \begin{pmatrix} p_c c_1 \\ p_c c_2 \\ p_c c_3 \\ \vdots \\ p_c c_{78} \\ p_c c_{79} \\ p_c c_{80} \end{pmatrix} = \begin{pmatrix} 0.12 \\ 0.13 \\ 0.44 \\ \vdots \\ 0.07 \\ 0.01 \\ 0.09 \end{pmatrix}$$

score: 0.44
box: (b_x, b_y, b_h, b_w)
class: c = 3 (“car”)

the box (b_x, b_y, b_h, b_w) has detected c = 3 (“car”) with probability score: 0.44

For the final output generation it carries out the following steps:

- Get rid of boxes with a low score (meaning, the box is not very confident about detecting a class)
- Select only one box when several boxes overlap with each other and detect the same object (Non-max suppression). It is done by Computing the overlap of one box with all other boxes, and remove boxes that overlap it more than iou_threshold.



Dataset preparation

I have primarily used the Stanford 40 dataset for the human activity detection dataset for my model. I have taken 13 classes for my dataset from the Stanford 40 dataset. However, I have also added two other classes, namely "Knives" and "Handguns" in my dataset from the google open image dataset for adding a little security feature in my detections. Also to add additional depth to my dataset I have added a few "Negative" samples which doesn’t belong to any of my classes. But after all this, my custom dataset presents us with very few images typically a couple of hundreds in number, so I have further used data augmentation to generate more pictures from the small number of images I had with me. I have used Clodsa Tool to augment my dataset and, after that, create corresponding annotations in the proper format to use the same in training. I have generated a dataset having over 30k images having, on average 2k images per class.

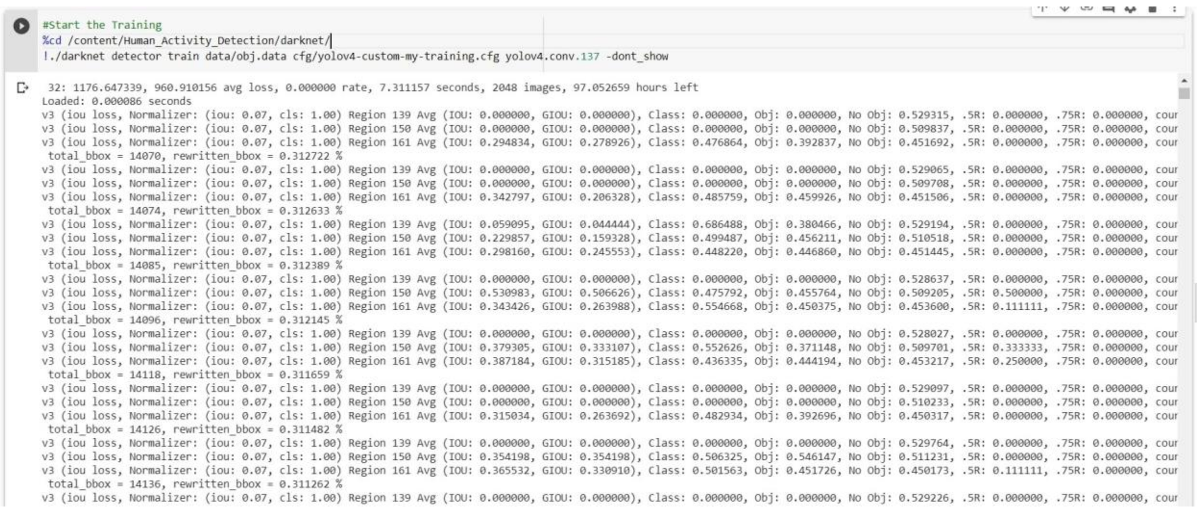
Here is a list of all the classes in my dataset:

- applauding
- reading
- taking_photos
- texting_message
- waving_hands
- jumping
- playing_guitar
- phoning
- running
- smoking
- watching_TV
- using_a_computer
- Handgun
- Knife
- climbing

Training

I have trained the Yolo v4 model on my custom dataset, and I have trained it for a total of 15000 iterations, and I stopped the training here because I saw that my model was starting to overfit the training data more and more as I continued further training. I have used Google Colab for training my model as it gives me the access to use their GPU over the cloud which speeds up the whole training process by a huge amount.

Here is a how you train the model on Colab:



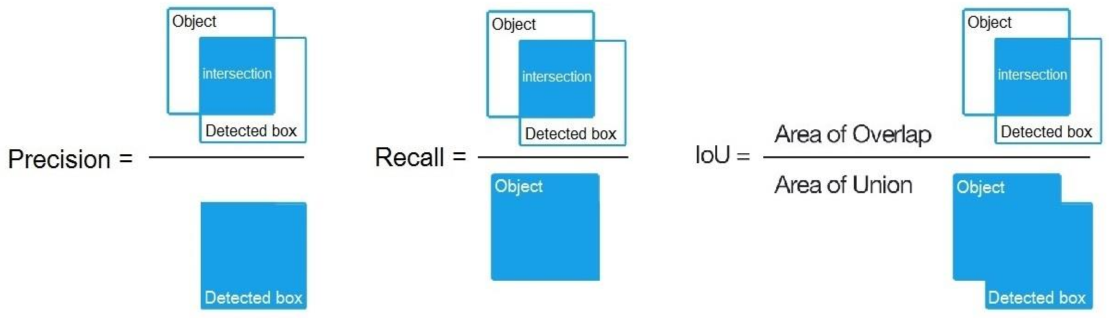
```
#Start the Training
%cd /content/Human_Activity_Detection/darknet/
!./darknet detector train data/obj.data cfg/yolov4-custom-my-training.cfg yolov4.conv.137 -dont_show
```

```
 32: 1176.647339, 960.910156 avg loss, 0.000000 rate, 7.311157 seconds, 2048 images, 97.052659 hours left
Loaded: 0.000086 seconds
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.529315, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.509837, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.294834, GIOU: 0.278926), Class: 0.476864, Obj: 0.392837, No Obj: 0.451692, .5R: 0.000000, .75R: 0.000000, cour
 total_bbox = 14070, rewritten_bbox = 0.312722 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.529065, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.509708, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.342797, GIOU: 0.206328), Class: 0.485759, Obj: 0.459926, No Obj: 0.451506, .5R: 0.000000, .75R: 0.000000, cour
 total_bbox = 14074, rewritten_bbox = 0.312633 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.059095, GIOU: 0.044444), Class: 0.686488, Obj: 0.380466, No Obj: 0.529194, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.229857, GIOU: 0.159328), Class: 0.499487, Obj: 0.456211, No Obj: 0.510518, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.298160, GIOU: 0.245553), Class: 0.448220, Obj: 0.446860, No Obj: 0.451445, .5R: 0.000000, .75R: 0.000000, cour
 total_bbox = 14085, rewritten_bbox = 0.312389 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.528637, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.530983, GIOU: 0.506626), Class: 0.475792, Obj: 0.455764, No Obj: 0.509205, .5R: 0.500000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.343426, GIOU: 0.263988), Class: 0.554668, Obj: 0.450375, No Obj: 0.453600, .5R: 0.111111, .75R: 0.000000, cour
 total_bbox = 14096, rewritten_bbox = 0.312145 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.528027, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.379305, GIOU: 0.333107), Class: 0.552626, Obj: 0.371148, No Obj: 0.509701, .5R: 0.333333, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.387184, GIOU: 0.315185), Class: 0.436335, Obj: 0.444194, No Obj: 0.453217, .5R: 0.250000, .75R: 0.000000, cour
 total_bbox = 14118, rewritten_bbox = 0.311659 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.529097, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.510233, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.315034, GIOU: 0.263692), Class: 0.482934, Obj: 0.392696, No Obj: 0.450317, .5R: 0.000000, .75R: 0.000000, cour
 total_bbox = 14126, rewritten_bbox = 0.311482 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.529764, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 150 Avg (IOU: 0.354198, GIOU: 0.354198), Class: 0.506325, Obj: 0.546147, No Obj: 0.511231, .5R: 0.000000, .75R: 0.000000, cour
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 161 Avg (IOU: 0.365532, GIOU: 0.330910), Class: 0.501563, Obj: 0.451726, No Obj: 0.450173, .5R: 0.111111, .75R: 0.000000, cour
 total_bbox = 14136, rewritten_bbox = 0.311262 %
v3 (iou loss, Normalizer: (iou: 0.07, cls: 1.00) Region 139 Avg (IOU: 0.000000, GIOU: 0.000000), Class: 0.000000, Obj: 0.000000, No Obj: 0.529226, .5R: 0.000000, .75R: 0.000000, cour
```

Testing

I have tested my model on the custom test data that I generated by manual annotations. I have used the method of IoU to generate the results for my model along with the percentage of successful detections for each class.

IoU stands for Intersect Over Union. IOU can be computed as Area of Intersection divided over Area of Union of two boxes, so IOU must be ≥0 and ≤1. When predicting bounding boxes, we need the find the IOU between the predicted bounding box and the ground truth box to be ~1.



Other criterion used while testing the Machine Learning model are Precision, Recall and Mean Average Precision(mAP) which is the mean of area under the precision-recall curve for all the classes.

ML Model Results

Train Data --

```
 calculation mAP (mean average precision)...
30804 images
 detections_count = 40004, unique_truth_count = 27942
class_id = 0, name = applauding, ap = 100.00%       (TP = 1988, FP = 7)
class_id = 1, name = reading, ap = 100.00%          (TP = 1715, FP = 2)
class_id = 2, name = taking_photos, ap = 100.00%            (TP = 1379, FP = 3)
class_id = 3, name = texting_message, ap = 100.00%          (TP = 1351, FP = 2)
class_id = 4, name = waving_hands, ap = 100.00%             (TP = 1470, FP = 14)
class_id = 5, name = jumping, ap = 100.00%          (TP = 2065, FP = 0)
class_id = 6, name = playing_guitar, ap = 99.85%    (TP = 2012, FP = 4)
class_id = 7, name = phoning, ap = 100.00%          (TP = 1812, FP = 0)
class_id = 8, name = running, ap = 100.00%          (TP = 1757, FP = 3)
class_id = 9, name = smoking, ap = 100.00%          (TP = 1687, FP = 7)
class_id = 10, name = watching_TV, ap = 99.99%      (TP = 1550, FP = 5)
class_id = 11, name = using_a_computer, ap = 99.99%         (TP = 1608, FP = 3)
class_id = 12, name = Handgun, ap = 98.30%          (TP = 2848, FP = 145)
class_id = 13, name = Knife, ap = 96.62%    (TP = 2433, FP = 161)
class_id = 14, name = climbing, ap = 100.00%        (TP = 2065, FP = 5)

 for conf_thresh = 0.25, precision = 0.99, recall = 0.99, F1-score = 0.99
 for conf_thresh = 0.25, TP = 27740, FP = 361, FN = 202, average IoU = 84.96 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.996500, or 99.65 %
Total Detection Time: 2329 Seconds
```

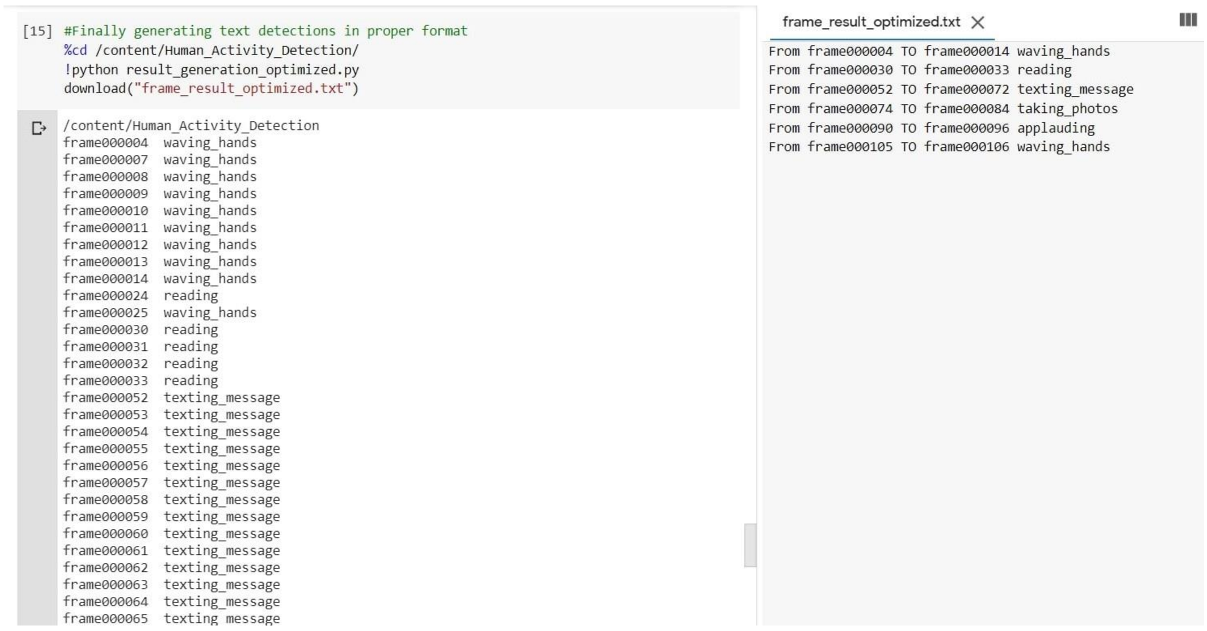
Test Data --

```
calculation mAP (mean average precision)...
1152 images
 detections_count = 2825, unique_truth_count = 1228
class_id = 0, name = applauding, ap = 81.04%        (TP = 23, FP = 3)
class_id = 1, name = reading, ap = 64.10%           (TP = 19, FP = 8)
class_id = 2, name = taking_photos, ap = 77.48%     (TP = 32, FP = 2)
class_id = 3, name = texting_message, ap = 78.91%           (TP = 55, FP = 2)
class_id = 4, name = waving_hands, ap = 96.74%              (TP = 53, FP = 6)
class_id = 5, name = jumping, ap = 67.22%           (TP = 32, FP = 15)
class_id = 6, name = playing_guitar, ap = 93.40%    (TP = 54, FP = 2)
class_id = 7, name = phoning, ap = 79.38%           (TP = 33, FP = 7)
class_id = 8, name = running, ap = 90.55%           (TP = 52, FP = 6)
class_id = 9, name = smoking, ap = 26.89%           (TP = 10, FP = 4)
class_id = 10, name = watching_TV, ap = 25.21%      (TP = 11, FP = 7)
class_id = 11, name = using_a_computer, ap = 78.17%         (TP = 36, FP = 6)
class_id = 12, name = Handgun, ap = 84.99%          (TP = 141, FP = 115)
class_id = 13, name = Knife, ap = 89.47%     (TP = 341, FP = 57)
class_id = 14, name = climbing, ap = 50.11%           (TP = 54, FP = 45)

 for conf_thresh = 0.25, precision = 0.77, recall = 0.77, F1-score = 0.77
 for conf_thresh = 0.25, TP = 946, FP = 285, FN = 282, average IoU = 60.72 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.722421, or 72.24 %
Total Detection Time: 28 Seconds
```

TP = True Positive, FP = False Positive, FN = False Negative, IoU = Intersection over Union, mAP = Mean Average Precision, conf_thresh = Confidence Threshold, ap = Average Precision, F1-score = F-score

Text generation from videos

Following the main idea of my project, I have written a python script that extracts image frames from a video with a specific delay and then performs the detections on each of the frames and then generates a text file with the results. This a workaround I had to do with the constraints I had with me. My script generates two text files, one with all the detections from every frame and another with action intervals.



```
[15] #Finally generating text detections in proper format
     %cd /content/Human_Activity_Detection/
     !python result_generation_optimized.py
     download("frame_result_optimized.txt")

/content/Human_Activity_Detection
frame000004  waving_hands
frame000007  waving_hands
frame000008  waving_hands
frame000009  waving_hands
frame000010  waving_hands
frame000011  waving_hands
frame000012  waving_hands
frame000013  waving_hands
frame000014  waving_hands
frame000024  reading
frame000025  waving_hands
frame000030  reading
frame000031  reading
frame000032  reading
frame000033  reading
frame000052  texting_message
frame000053  texting_message
frame000054  texting_message
frame000055  texting_message
frame000056  texting_message
frame000057  texting_message
frame000058  texting_message
frame000059  texting_message
frame000060  texting_message
frame000061  texting_message
frame000062  texting_message
frame000063  texting_message
frame000064  texting_message
frame000065  texting_message
```

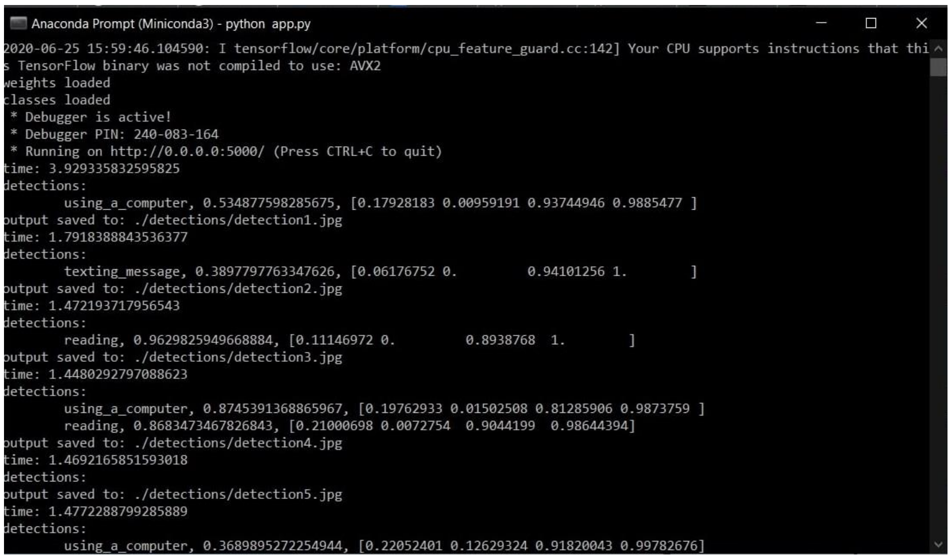
frame_result_optimized.txt

```
From frame000004 TO frame000014 waving_hands
From frame000030 TO frame000033 reading
From frame000052 TO frame000072 texting_message
From frame000074 TO frame000084 taking_photos
From frame000090 TO frame000096 applauding
From frame000105 TO frame000106 waving_hands
```

API

For creating a web app for making detections more user friendly we need an API. I have used an API that can perform the detections for the user using the computing power available at the server and generate a JSON response, which is sent to the user for further use.

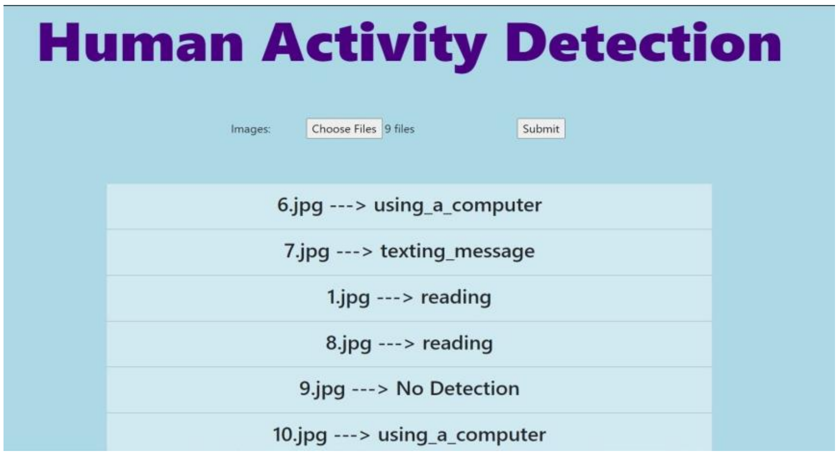
The local Web server is created with Flask, enabling us to use yolo custom model to do the detections. Our API converts our yolo written in C to a more python and browser friendly TensorFlow model and do the detections using the same. I have changed the webserver to add CORS feature so that it can connect to any web page freely and removed some features of the original API available here to suit my needs properly.



```
2020-06-25 15:59:46.104590: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports instructions that thi
s TensorFlow binary was not compiled to use: AVX2
weights loaded
classes loaded
 * Debugger is active!
 * Debugger PIN: 240-083-164
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
time: 3.929335832595825
detections:
        using_a_computer, 0.534877598285675, [0.17928183 0.00959191 0.93744946 0.9885477 ]
output saved to: ./detections/detection1.jpg
time: 1.7918388843536377
detections:
        texting_message, 0.3897797763347626, [0.06176752 0.         0.94101256 1.        ]
output saved to: ./detections/detection2.jpg
time: 1.472193717956543
detections:
        reading, 0.9629825949668884, [0.11146972 0.         0.8938768  1.        ]
output saved to: ./detections/detection3.jpg
time: 1.4480292797088623
detections:
        using_a_computer, 0.8745391368865967, [0.19762933 0.01502508 0.81285906 0.9873759 ]
        reading, 0.8683473467826843, [0.21000698 0.0072754  0.9044199  0.98644394]
output saved to: ./detections/detection4.jpg
time: 1.4692165851593018
detections:
output saved to: ./detections/detection5.jpg
time: 1.4772288799285889
detections:
        using_a_computer, 0.3689895272254944, [0.22052401 0.12629324 0.91820043 0.99782676]
```

Web App

I have to build a small web app to easily visualize the working of my model by only using your browser and also the API I used earlier. My web app lets the users select images of their choice and then result in the text detections generated by using the API.



Human Activity Detection

6.jpg ---> using_a_computer

7.jpg ---> texting_message

1.jpg ---> reading

8.jpg ---> reading

9.jpg ---> No Detection

10.jpg ---> using_a_computer

Conclusion

I can honestly say that my time spent interning with the Samsung Research Institute Noida resulted in one of the best summers of my life. I have gained professional skills during this internship that will surely help in my future. For the internship project, I have built a Machine Learning model for generating text detections from video files. My model turned out to be reasonably accurate when performing on videos, which are similar to the dataset used but perform rather poorly on difficult videos such as which have dark lighting conditions or when its quality is poor. Also, I have built an API that performs over my model and creates a local webserver to be used freely by the web application which I have made.

Needless to say, the technical aspects of the work I've done are not flawless and could be improved provided enough time. My model can be further made more accurate by training it on a dataset, which is quite bigger and detailed than the one I used. My model is essentially an object detection algorithm, but this project can be further made more robust by adding an object tracking and recognition elements so that it can generate a separate data for each person it encounters. The model can be run on a high-end PC so that its fps increases, and then it could be used as real-time detection. The idea of web detections can further be expanded by making a full-fledged web app that would empower its users to perform detections by just installing the app over their system without any hassle.

In a nutshell, my internship has been an excellent and rewarding experience. As someone with no significant practical experience with Machine Learning, I believe my time spent researching and discovering it was well worth it and contributed to finding an acceptable solution to build a fully Machine Learning solution to the problem statement. For over the course of this internship, I have learned many new things about the vast field of machine learning and all the latest technologies it is creating every day and about its enormous future in the techno world. I have learned about the Yolo Algorithm, Google Colab, Flask, and all other technologies and that to while doing this fun and practical project. Two main things that I've learned the importance of are time-management skills and self-motivation while doing this project.