Bangladesh University of Engineering and Technology

Department Electrical and Electronic Engineering

Level:3 **Term:** 1

Section: C **Group No.: 08**

Course name: EEE 312

Course Title: Digital Signal processing I laboratory

Project Name:

Social Distance Monitoring in Covid19 Situation

Submitted To

Tanvir Mahmud (Lecturer)

Satyaki Banik (Lecturer)

Dept. of EEE BUET

Submitted BY

Nahid Ahmed (1706145)

Sanath kumar Das (1706149)

Md. Rokonujjaman (1706150)

Ishmam Hasnat Iram (1706153)

Joy Saha (1706189)

Project Objective:

In this DSP-I sessional course project, we group 08, were assigned a project named “Social Distance monitoring in Covid-19 situation”.

This system will be able to

- Detect people from image or video
- Localize them in bounding box
- Determine distance among all people in a given frame
- From distance value, gives us “alert” signal, if social distance is not maintained, otherwise will say “safe”.

Project Methodology:

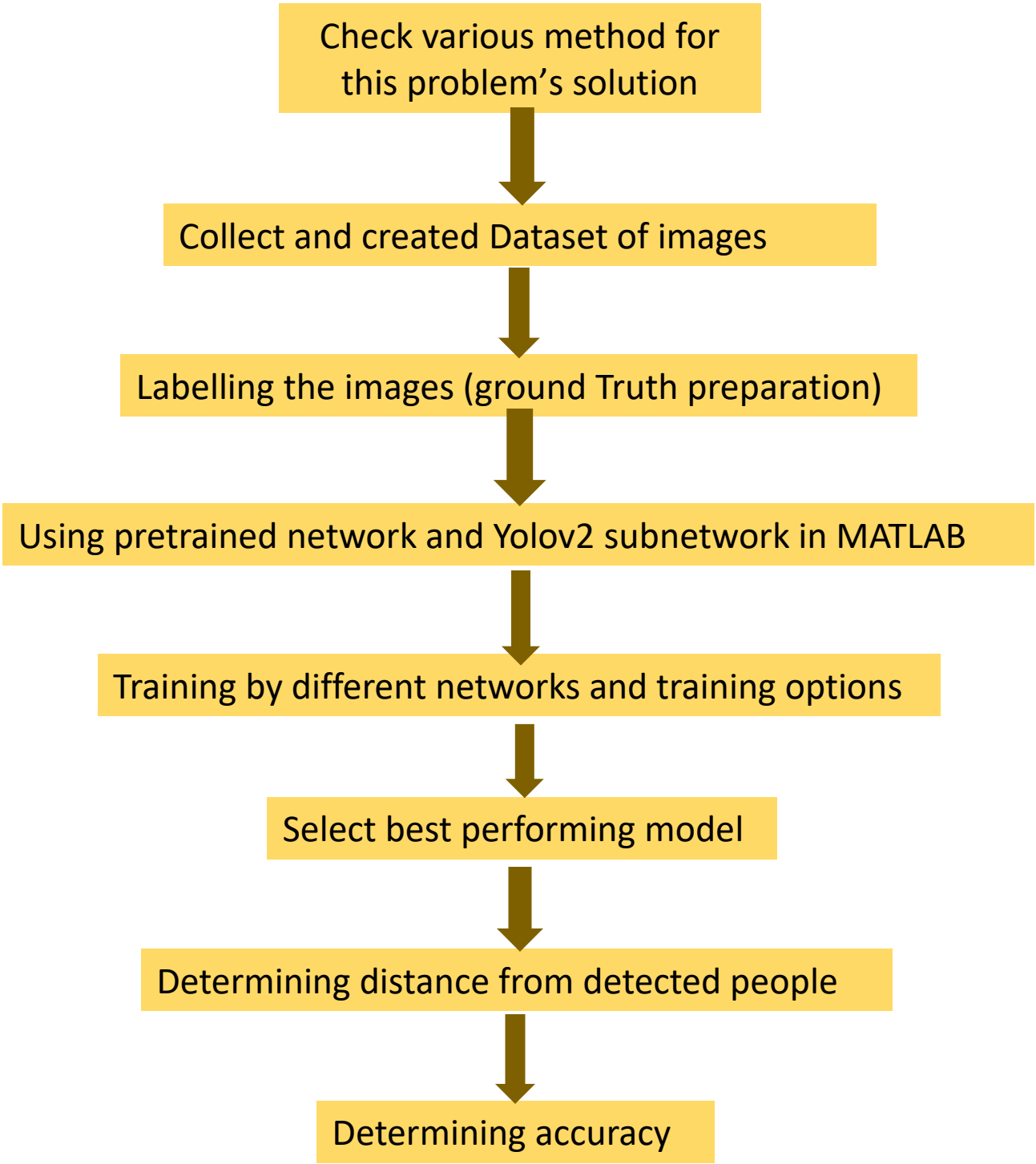
To accomplish our project goal, we decided to adopt the top technology available to us. So, we have used deep learning approach to solve our problems. We have studied some literature to find the best solution for our specific problem.

We implement this whole project in Matlab2020b.

Using pretrained network of MATLAB, we have completed this project.

We have collected a satisfactory amount of dataset, trained our model by best performing architecture, and finally tested on unseen data. Although we have faced so many difficulties to implement this project in MATLAB, we have tried our best to achieve our project objective.

Workflow


Check various method for this problem's solution
Collect and created Dataset of images
Labelling the images (ground Truth preparation)
Using pretrained network and Yolov2 subnetwork in MATLAB
Training by different networks and training options
Select best performing model
Determining distance from detected people
Determining accuracy

..

Literature Study:

Several deep learning algorithms are available and every newly developed algorithm has resolved the problems of the previous one in some way. Conventional object detection algorithms use classifier-based procedure, where the classifier runs on a slice of the image in sliding window fashion, this is how Deformable Parts Model (DPM) works**. In R-CNN ancestry (R-CNN, Fast R-CNN and Faster R-CNN) classifier run on region proposals that are considered as bounding boxes. These algorithms exhibit good performance, especially Faster R-CNN with an accuracy of 73.2% mAP, but because of their intricate pipeline, they show poor performance in the context of speed with 7 frames per second (FPS), which limit them for real-time object detection.**

This is where YOLO fits, a real-time object detection system with a creative perspective of reviewing object detection as a regression problem was introduced in 2016 by Joseph et al. **YOLO exhibits good performance as compared to previous region-based algorithms in terms of speed with 45 FPS by maintaining good detection accuracy of 63.4% mAP**. Despite good speed and performance, YOLO made notable localization errors. Moreover, YOLO has low recall.

To resolve the shortcomings of YOLO, in the same year authors of YOLO released YOLO second version where recall and localization were mainly focused without affecting classification accuracy. YOLO v2 gained a speed of 67 FPS and mAP reached 76.8%. YOLO v2 is also called YOLO 9000 because of its ability to detect objects of more than 20 classes by mutually optimizing classification and detection.

The YOLO v3 developed in 2018 brought new improvements in speed and accuracy, but the main idea remained the same.

Yolo (You Only Look Once) algorithm:

You look at one or (YOLO) is a state-of-the-art deep learning object detection. It was presented by Joseph Redmon et al. YOLO uses a single neural network to the whole image. It divides the image into regions and predicts the bounding boxes and the probabilities for each region. These bounding boxes are weighted by predicted probabilities.

YOLO detector looks full image at one time; therefore, its predictions are informed by the context in the image. It predicts with single network evaluation, unlike other object detectors such as (R-CNN) which requires thousands for a single image. YOLO algorithms take the input image and split into S ×S grids. It extracts the features from each grid. It predicts the bounding boxes with confidence scores for the predicted classes in the bounding boxes, see Fig.01. Each grid cell detects bounding boxes and confidence scores. The bounding box consists of five predictions which are represented with (x, y, w, h) and the confidence score. The (x, y) coordinates reflect the center of the bounding box of the grid cell. The (w, h) represents the width and the height of the full image. The confidence scores represent the measurement of how confident the detector is that the box contains the object to be predicted.

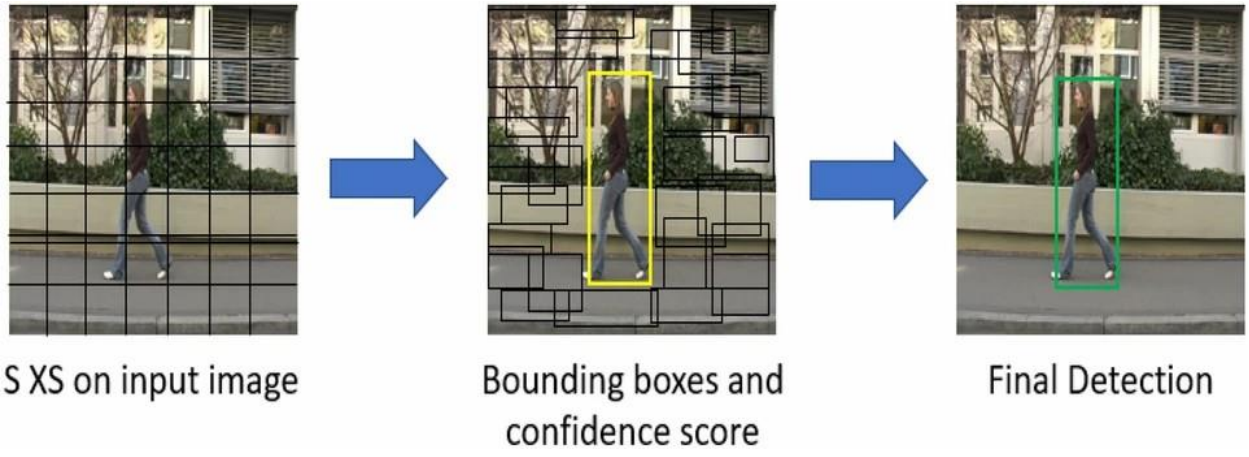


Fig.01: How YOLO works

YOLO predicts several bounding boxes for each grid cell. In the training stage, it only requires one predictor of the bounding box to be responsible for each class. The predictor is assigned to predict an object which has the highest Intersection over Union value (IoU) for the ground truth. This process leads to specialization within the bounding boxes prediction.

Loss Function

YOLO algorithms use sum-squared error between the ground truth and predictions of bounding boxes for loss. This sum squared error computes the classification, localization, and confidence losses for the model. Therefore, YOLO is optimized with the following loss function to enhance its performance during the training process,

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{i,j}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + \left(y_i - \hat{y}_i \right)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{i,j}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{i,j}^{\text{obj}} (C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_{i,j}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{s^2} 1_i^{\text{obj}} \sum_{c \in \text{class}} (p_i(c) - \hat{p}_i(c))^2,
\end{aligned}
$$

where: **λcoord** is a constant used to increase the weight for the first two terms of the loss function. **B** is the number of box predictions for each cell. $\boldsymbol{S^2}$ is the number of cells. $\mathbf{1}_{i,j}^{obj}$ is equal to 1 if there is an object in cell *i* and confidence of the *j*th predictor of this cell is the highest among all the predictors of this cell. $\mathbf{x_i}$, $\mathbf{y_i}$ represent the location of centroid of the anchor box. $\mathbf{w_i}$ is the width of the anchor box. $\mathbf{h_i}$ is the height of the anchor box. $\mathbf{C_i}$ is the confidence score whether there is an object or no. $\hat{\boldsymbol{C}}_{\boldsymbol{i}}$ is the box confidence score of the box j in cell i. **λnoobj** weights down the loss when detecting background. $\mathbf{1}_{i,j}^{noobj}$ is the complement of $\mathbf{1}_{i,j}^{obj}$. $\mathbf{1}_{i}^{obj}$ = 1 if an object appears in the cell i, otherwise 0. $\mathbf{P_i(c)}$ is the classification loss. $\hat{\boldsymbol{P}}_{\boldsymbol{i}(C)}$ is the conditional class probability for class c in cell i.

Architecture of our whole model:

.

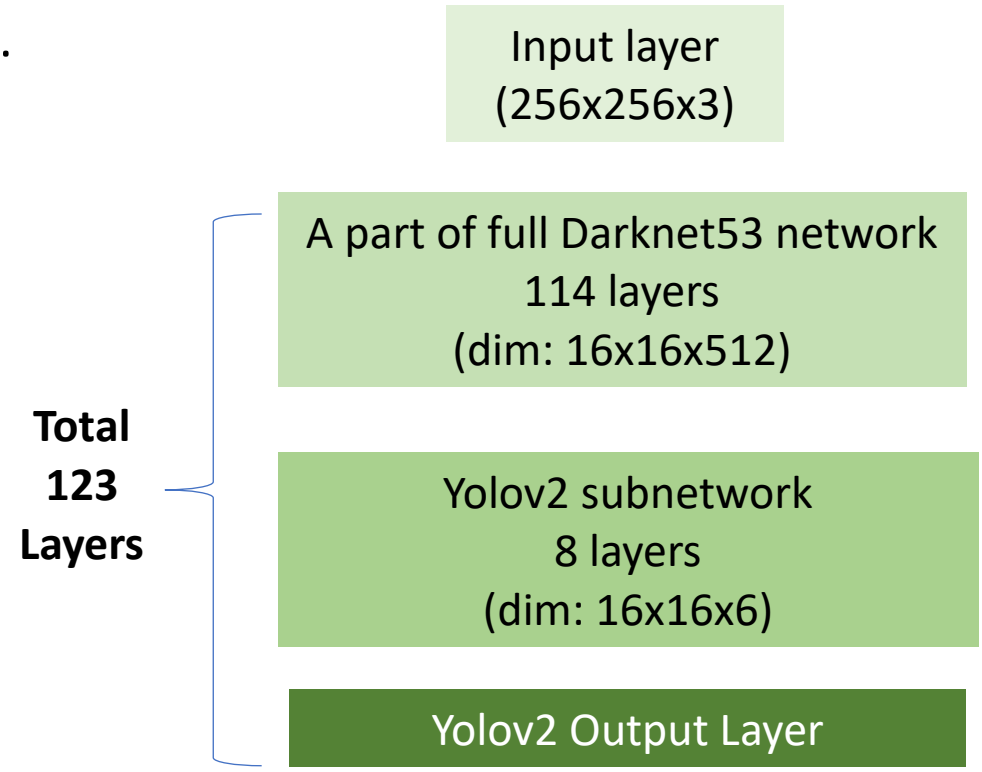


Fig 02. Full model Architecture at a glance

We have used darknet-53 as backbone network, and Yolov2 as subnetwork. In the darknet-53 portion, we have used "res14" layer as a feature extraction layer for yolov2 subnetwork. The darknet-53 is here 114 layers. There is 14 repetition of residual block. Each residual block is consisting of Convolution, Batch normalization and leaky RELU layer. Darknet-53 is extracting different features from input images and feeds them to the yolov2 subnetwork.

In the left side, we can see the 1st residual block. This block is repeated in the backbone network of our model for 14 time. The output of 14th residual block is connected with the yolov2 sub network.

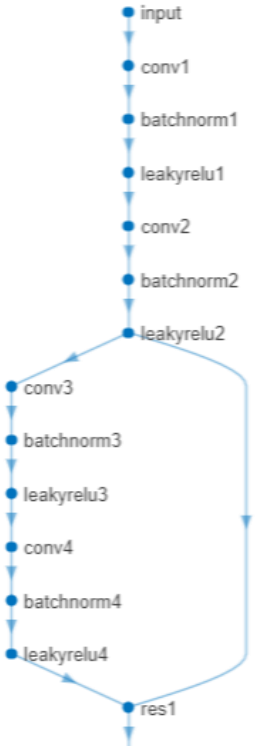


Fig 03: Input to 1st residual block connection

Yolov2 Subnetwork

This subnetwork detect object using the features of backbone network and localize them with bounding box with probability.

It has total 8 layers. **There is total 8 layers and finally there is output layer. This subnetwork consists of convolutions, batch normalization, RELU, yolov2 transform layer**. Then there is finally the output layer

The transform layer in YOLO v2 object detection network improves the stability of the network by constraining the location predictions. The transform layer extracts activations of the last convolutional layer and transforms the bounding box predictions to fall within the bounds of the ground truth**.**

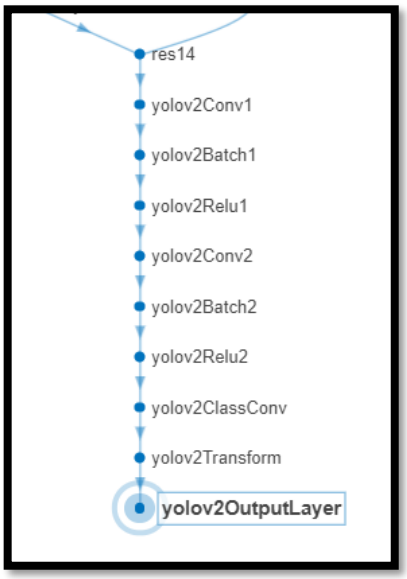


Fig. 04: Yolov2 subnetwork

MATLAB creates this subnetwork by using a single function named, '**yolov2Layers()'.** yolov2Layers uses a pretrained neural network as the base network to which it adds a detection subnetwork required for creating a YOLO v2 object detection network. Given a base network, yolov2Layers removes all the layers succeeding the feature layer in the base network and adds the detection subnetwork. So, here, all the layers after "res14" in darknet-53 are removed, as we have taken features from the "res14" layer of darknet-53.

```
[anchorBoxes]=estimateAnchorBoxes(scaledData_train,1)
featureExtractionLayer = "res14";
numClasses=1;
net=darknet53();
lgraph = yolov2Layers([256 256 3],numClasses,anchorBoxes,net,featureExtractionLayer)
```

Fig. 05: MATLAB command for creating yolov2 subnetwork

In figure 05, we can see that the input image size is 256x256x3. As we need to detect only humans or persons, so we have only single class here. We have used only single anchor box. "lgraph" stands for layer graph, which keeps all info about the whole network. Feature extraction layer is "res14" layer, one of the layers of base network darknet-53.

Thus, we have created the whole architecture for our project using pretrained darknet-53.

Dataset Collection:

We have used different source of dataset in this project. **We have total 814 images for our project.**

From them almost 300 images came from three different video by frame extraction method. These videos are pedestrian video from CCTV camera. But none of them are subsequent frames. Moreover, we have shuffled all the frames or images before training. By doing this, we have avoided the overlapping of frames.

The rest of the images, we have collected from some different data sources or randomly collected pedestrian images.

Ground Truth Data:

By using the MATLAB image Labeler app, we have annotated every image from our dataset. Though, it was a very time-consuming task, we have done it very carefully. We have annotated people from images with bounding boxes, and saves this as a ground truth data. This data will be used for training and validation. For testing we will use an unseen new video data. We have only a single class named “human”

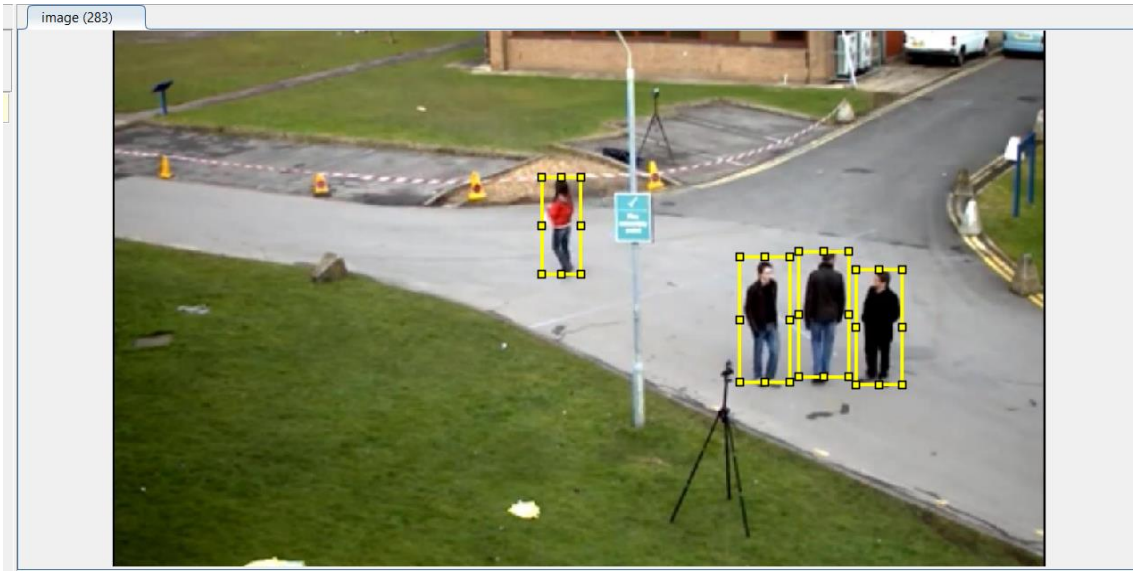


Fig 06: How we have annotated our images

Image Scaling:

There were many different sizes of images in our dataset. As our model take input **image of size 256x256x3,** we have to scaled all our images as well as annotated bounding box. After scaling, all the images dimension and bounding box coordinated will be ready for training

Train-Validation-Test set Splitting:

We have used maximum images for training, and rest of them are used for validation and testing.

There was used 730 images for training, 50 images for validation and rest if the images are used for testing.

Training the Model with Ground Truth Data:

There is no facility in MATLAB to train online. So, we had to train the model in our local computer. So, we couldn't use many epochs, as it takes much times for training. After all, our result was not so bad. **We have performed training in two stage for better performance.**

At first stage training, we have used

- ❑ Optimizer: 'ADAM'
- ❑ Uniform learning rate: 0.0001
- ❑ Mini batch size = 16 images
- ❑ Total epochs =20;

After training of 20 epochs….

- ❑ training loss was 0.2
- ❑ And validation loss was 0.4 ~ 0.5.

```
options = trainingOptions('adam', ...
        'MiniBatchSize',16, ...
        'InitialLearnRate',0.0001, ...
        'MaxEpochs',20,...
        'ExecutionEnvironment','multi-gpu',...
        'Shuffle','every-epoch',...
        'ValidationData',scaledData_val,...
        'ValidationFrequency',20);
```

Fig.07: Training options in 1st stage training

In the figure below, we can see the necessary command for our training. Here, scaledData_train is the training ground truth data, “lgraph” is layers data, “options” is the training options. It gives the detector as output and losses in each iteration in the info variable

```
%%
[detector,info] = trainYOLOv2ObjectDetector(scaledData_train,lgraph,options);
```

Fig.08: MATLAB command for training

In the next step, we have trained the same detector again, which was achieved from the 1st stage training

We have used L2-regulariztion here,

Trained for 5 epochs.

After training the loss reduced slightly.

```
%% again train
options2 = trainingOptions('adam',...
        'l2Regularization',0.0005,...
        'MiniBatchSize',16, ...
        'InitialLearnRate',0.0001, ...
        'MaxEpochs',5,...
        'ExecutionEnvironment','multi-gpu',...
        'Shuffle','every-epoch',...
        'ValidationData',scaledData_val,...
        'ValidationFrequency',20);
%%
```

Fig.09: Training options in 2nd stage training

```
%%
[detector_final,info] = trainYOLOv2ObjectDetector(scaledData_train,detector,options2);
%%
```

Fig.10: MATLAB command for training in 2nd stage

The output variable in the leftmost side in fig-10, is our final object detector. In this variable, all the weights and biases of the model is saved. Now we can use this object detector named “detector_final”, for testing our images or videos.

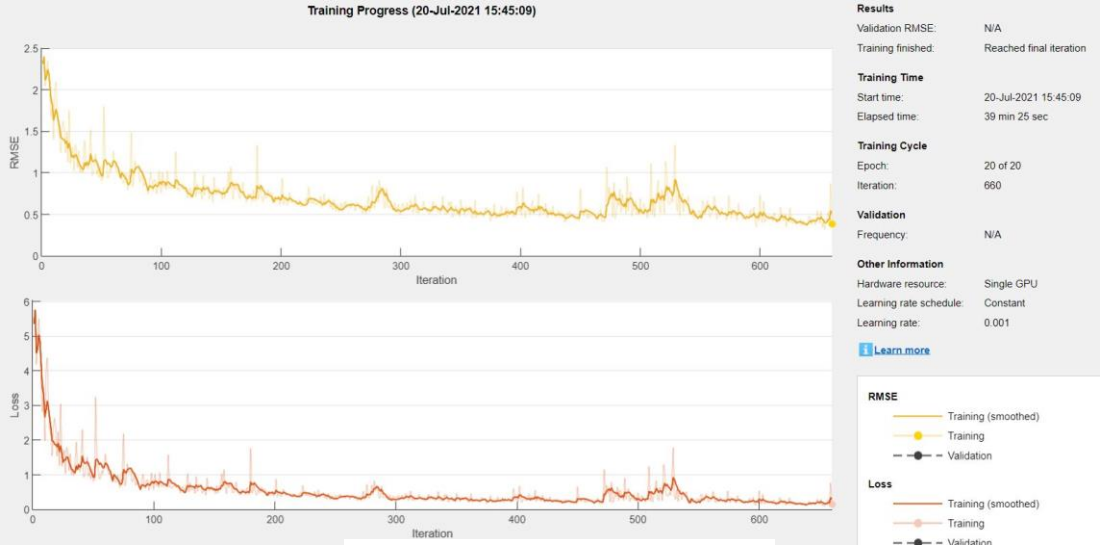


Fig.11: Training Curve

From the figure 11, we can see that, our training loss is gradually decreasing as well as training RMSE.

Testing the object detector:

The detector is able to detect persons from images man, woman, children. We have shown it in the next some figure. The Object detector which we have trained, it was detecting people, localized with bounding box with corresponding probability scores.

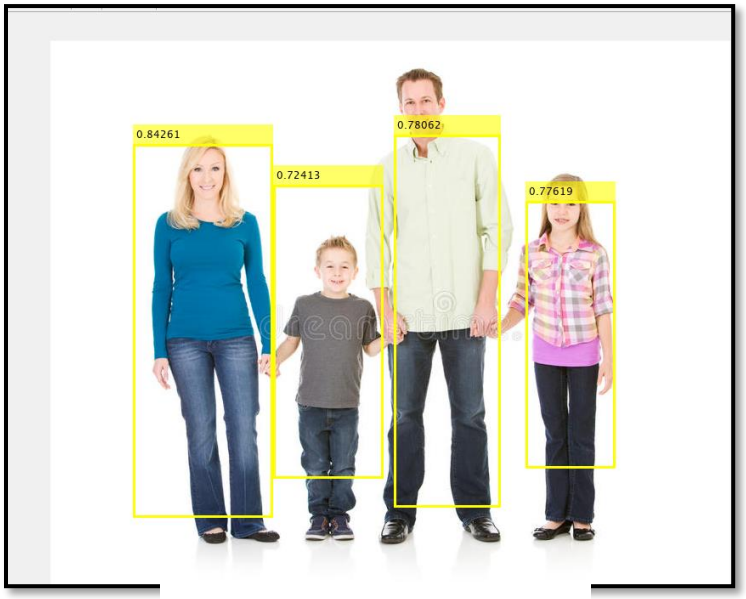
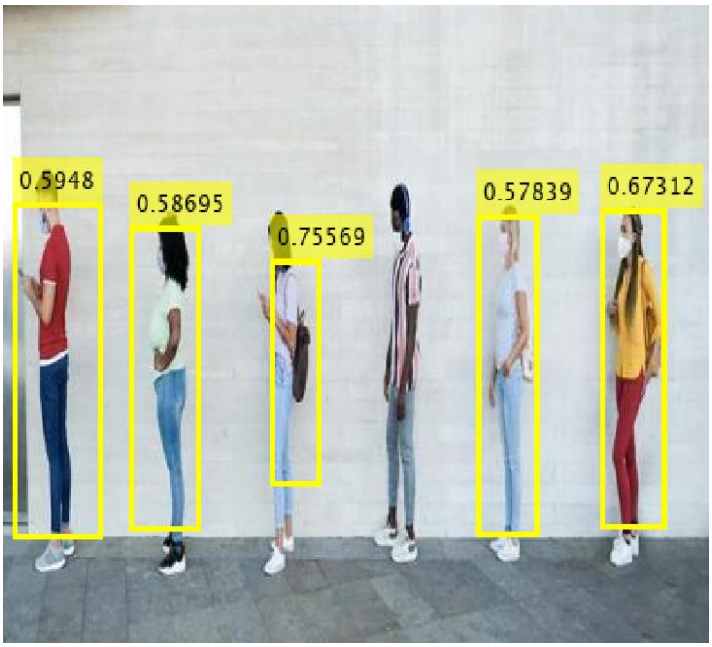


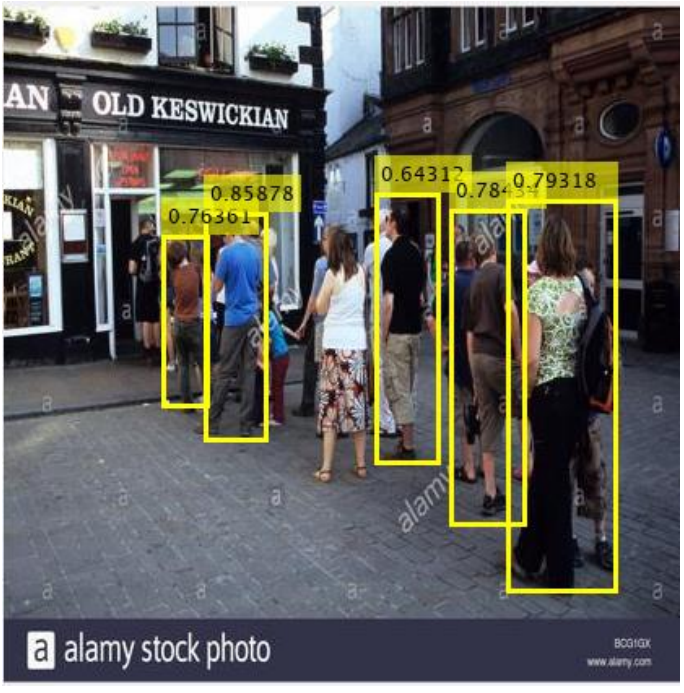
Fig.12: Test with sample image

More Sample Images after testing:




0.74465
0.53251
0.51703
0.75286
0.90813
0.8549
0.80432
0.86726


OLD KESWICKIAN
alamy stock photo


OLD KESWICKIAN
0.85878
0.76361
0.64312
0.79318
alamy stock photo

From all above images we can see that Our object detector was not 100% accurate, but its result is not so bad. By using the latest version of YOLO and latest architecture, more dataset, and training epochs we can overcome this problem in future.

Determining Distance:

After detecting people, it is time to determine distance between them. As camera capture a 3d place , make it 2d picture, some information is lost in image. So, we can't accurately determine distance between all objects in an image.

Instead of this limitation, we can approximate the distance, using Euclidian distance between the centers of bounding box of detected people in an image or video. First, we determine the centers of all bounding box from output coordinate of bounding box of our object detector. Then, using below formula we have determine distance between the all centers.

Distance between two bounding box centers = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$

If the distance between any two or more bounding box crosses the threshold distance (critical distance) which looks like 3 feet or 1 meter, the bounding boxes will be red marked, the rest of the bounding box will be marked as green. The red bounding box means the detected people are not safe, the green bounding box indicates that they are safe.

If we implement this project in real world and hardware, there will be an alarm sound, if there is any red bounding box in a given frame. thus, we can assure the safe distance

Empirically, we have seen that ,the Euclidian approximation works better when the camera is placed in certain height greaten than human height, not exact at human eye level.

Final Output Test:

We have tested our model with a video to see the accuracy of our model.

We have attached some of its screenshot of this.

Conclusion:

In this project we have made an object detector using Yolov2 algorithm. It has a satisfactory amount of accuracy. More training and latest architecture of yolo can this model performing better.

As it was a first project on deep learning, our team has learnt a lot from this project. We have tried our best to fulfill or target in this project. Although, it needs many more improvements to implement in real world.

We have learnt about neural network, image convolution, convolutional neural network, latest technology of object detection, localization and many more. This learning will be very helpful for us near future.

References

[1] Saponara, S., Elhanashi, A. & Gagliardi, A. Implementing a real-time, AI-based, people detection and social distancing measuring system for Covid-19. *J Real-Time Image Proc* (2021). https://doi.org/10.1007/s11554-021-01070-6

[2] Dataset source: https://www.cis.upenn.edu/~jshi/ped_html/

[3] MATLAB documentations and tutorials