BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS

Department of Automotive Technologies

Vehicle testing and validation

Training of 2D object detection neural network on infrared image dataset

Author: Shyamsundar Sudharsanam

Consultant:Márton Cserni

Budapest, 2022. 05. 17.

Table of contents

1.Overview:

The IR images taken from the Zalazone is not detecting the objects properly with the object detector, as the image taken by the camera is not aligned, i.e taken at some height and hence the detector is not working as expected. The purpose for doing this project is to train, test and detect the YOLOR model with the FLIR dataset and comparing the performance with YOLOV5 models, so as to use this models for IR images taken from ZalaZone . Most importantly, we are not using any pretrained weights for training and all weight parameters are learned from the scratch.

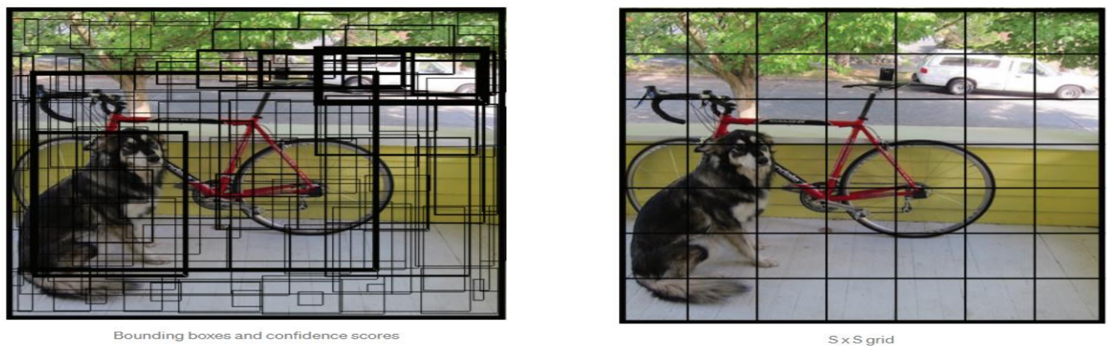
In addition, the GPU resources to train the model for 300 epochs requires huge memory and this is a constraint for using free COLAB version which provides TESLA T4 cores of size 15GB for a runtime of 10- 12 hours per day. Hence, we limited to run for 10 epochs only and investigated the mean Average precision mAP@0.5 (which is IOU 0.5).

2.Object Detection

Object detection deals with the localization and classification of objects contained in an image or video, and it comes down to drawing bounding boxes around detected objects. It differs from the Image classification which basically sends an entire image through a classifier (such as a CNN), and it gives out a tag associated with a label, but clearly, they don’t give any indication on where this tag might be located in the image. There are two types of object detector are there one is two stage and other one is single stage object detector. The Single Stage object detector are more suitable for real time detection because of the time

3. YOLO – You Look Only Once

The YOLO — You Only Look Once — network uses features from the entire image to predict the bounding boxes, moreover, it predicts all bounding boxes across all classes for an image simultaneously. This means that YOLO reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision.



1. Figure, How YOLO Model bounding box and grids are created [1]

YOLO divides the input image into an S × S grid. If a grid cell contains the center of an object, it is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores associated with the boxes. The confidence scores indicate how confident the

model is that the box contains an object, and additionally how accurate the box is. Formally, the confidence is defined as P(Objectness) x IOU_truthpred. Simply put, if no object exists in a cell, the confidence score becomes zero as P(Objectness) will be zero. Otherwise, the confidence score is equal to the intersection over union (IOU) between the predicted box and the ground truth.
Each of these B bounding boxes 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(w) and height(h) are predicted relative to the whole image. The confidence prediction represents the IOU between the predicted box and any ground truth box.

Each grid cell also predicts C conditional class probabilities, P(Class-i |Object). These probabilities are conditioned on the grid cell containing an object. Regardless of the number of associated bounding boxes, only one set of class probabilities per grid cell is predicted.

Finally, the conditional class probabilities and the individual box confidence predictions are multiplied. This gives us the 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
YOLO performs classification and bounding box regression in one step, making it much faster than most CNN-based approaches. For instance, YOLO is more than 1000x faster than R-CNN and 100x faster than Fast R-CNN. Furthermore, its improved variants such as YOLOv3 achieved 57.9% mAP on the MS COCO dataset. This combination of speed and accuracy makes YOLO models ideal for complex object detection scenarios

Non Max Suppression:

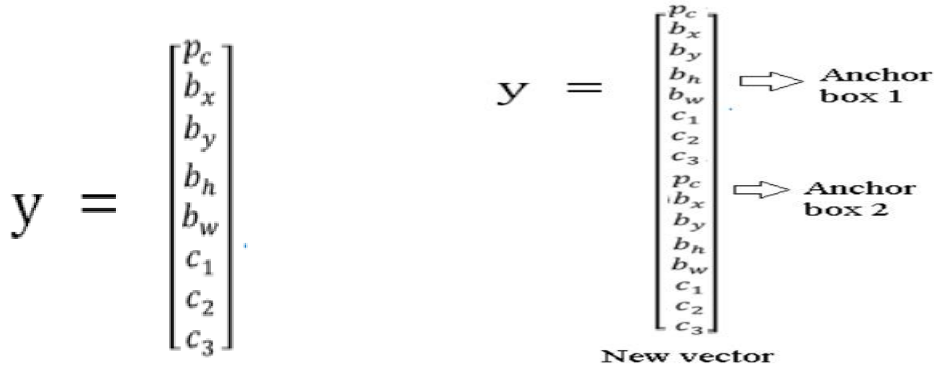
If there are more than one box detected for the object, we have to remove the bounding box this is what the Non max suppression does, it only keeps the Bounding box of maximum and removes the other

Intersection Over Union:

If the IOU is strong, we say the bounding box is aligned with the ground truth, and it is generally calculated by dividing the area of Overlap by the area of Union
Normal consideration is > 0.5 which yields a good IOU

Anchor Box:

If the grid cell , has more than one Object present in it and if we want to predict multiple objects , we use anchor box to do this , it has combination of two vectors


$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$
$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$
Anchor box 1
Anchor box 2
New vector

2. Figure, YOLO vector(Left) and Anchor box vector (Right), [2]

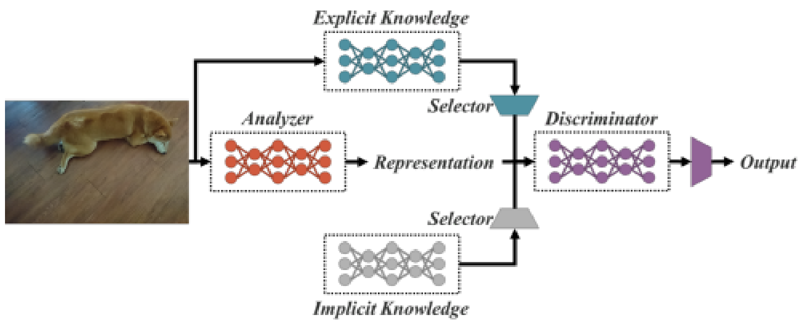
3.1 Versions of YOLO:

1. YOLO: it was built using darknet neural networks
2. YOLO V2 : : It uses more anchor boxes and a new bounding box regression algorithm, thus improving performance
3. YOLO V3: deeper feature detector network, slightly more fast with 30ms per inference.
4. YOLO V4: an upgrade, it breaks the object detection task into two main stages. The first one, regression, to identify the object positioning via bounding boxes; the second one it's the classification, to determine the object's class.
5. YOLO V5: this sees an upgrade in performances on YOLOv4, characterized by a mosaic augmentation technique

4.YOLOR – You Look only One Representation

The YOLOR research paper describes an approach to combine explicit knowledge, defined as learning based on given data and input, with implicit knowledge learned subconsciously. Therefore, the concept of YOLOR is based on encoding implicit and explicit knowledge together, similar to how mammalian brains process implicit and explicit knowledge in conjunction with each other. The unified network proposed in YOLOR generates a unified representation to serve a variety of tasks all at once.

In terms of neural networks, knowledge obtained from observation is known as Explicit Deep Learning and corresponds to the shallow layers of a network. Implicit Deep Learning however corresponds to the deeper layers of a network, usually where the features are extracted.



3. Figure, YOLOR Model architecture, [3]

Prima of Convolutional Neural Networks was specific to solve only specific objective, they produce the feature maps from the inputs through different filters . But the CNN can be used to solve to get outputs from inputs and different outputs rather than one.

Unified Network:

This is where unified networks come into the picture, we can now expand our equation to incorporate the implicit model with g theta and the explicit error from observation x together with the implicit error from z which they term the latent code. In layman's terms it just refers to the representation of compressed data that makes up the implicit knowledge. We can further simplify the equation to this.

$$y = f_\theta(\mathbf{x}) + \epsilon + g_\phi(\epsilon_{ex}(\mathbf{x}), \epsilon_{im}(\mathbf{z}))$$
$$\text{minimize } \epsilon + g_\phi(\epsilon_{ex}(\mathbf{x}), \epsilon_{im}(\mathbf{z})) \qquad (2)$$

(3.1)

Implicit knowledge can be utilized for manifold space reduction, kernel space alignment and more functions. Also the model outperforms the other scaled YOLOV4 model and were clearly articulated in the YOLOR Paper .

5.Training on FLIR Data :

For Training on Infrared Images , we relied on FLIR dataset which was downloaded from the below link that contains the RGB and the thermal images , in this particular project we want to test the YOLOR using the Infrared Images . Regarding the Training , Test and validation Split, you don't have to go for any further split as all were made available in the FLIR website

Dataset Link: https://www.flir.in/oem/adas/adas-dataset-form/

Object Categories used in FLIR dataset

Category ID	Class	Category ID	Class	Category ID	Class
0	person	6	train	16	dog
1	bike	7	truck	36	skateboard
2	car	9	light	72	stroller
3	motor	10	hydrant	76	scooter
5	bus	11	sign	78	other vehicle

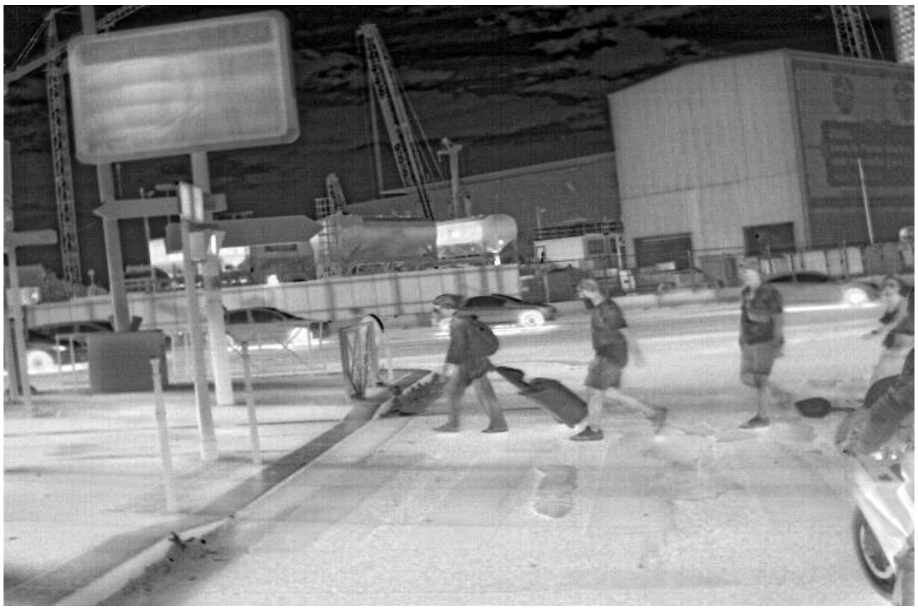
1. Table, Class IDs, and labels in the FLIR dataset

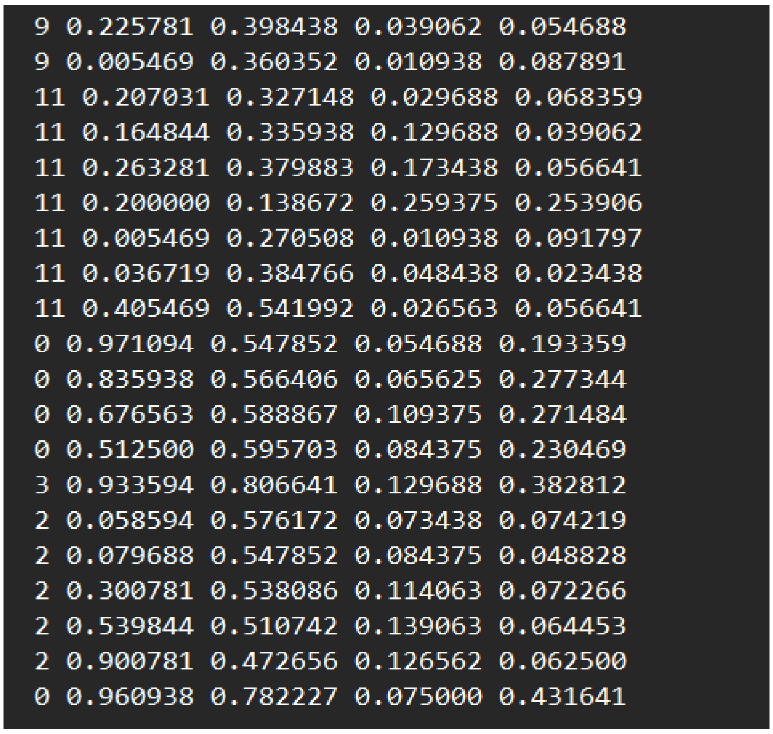
5.1 Label Extraction

Unlike the conventional Image classification labels where we have only class ID, object detection state of architecture models like YOLO requires the Class id and the bounding box details . Under each category of data that is test , train and validation , the JSON file will be there, you can use this for segregation of labels . But Remember, Since the FLIR dataset has been made according to COCO format you have to convert that in YOLO format .The difference is that the X and Y for COCO is on the upper left whereas for the YOLO the X and Y are the center point .

Code to extract the labels and convert to yolor format is available under here
https://github.com/SudharsanamShyamsundar/FLIR_YOLOR/

Labelling the class Id and bounding box for an image to YOLO format . You can see below for the given image , how the class id and bounding box (X,Y,W,H) are created.


4. Figure, Input Image used in extracting the class id and bounding box details



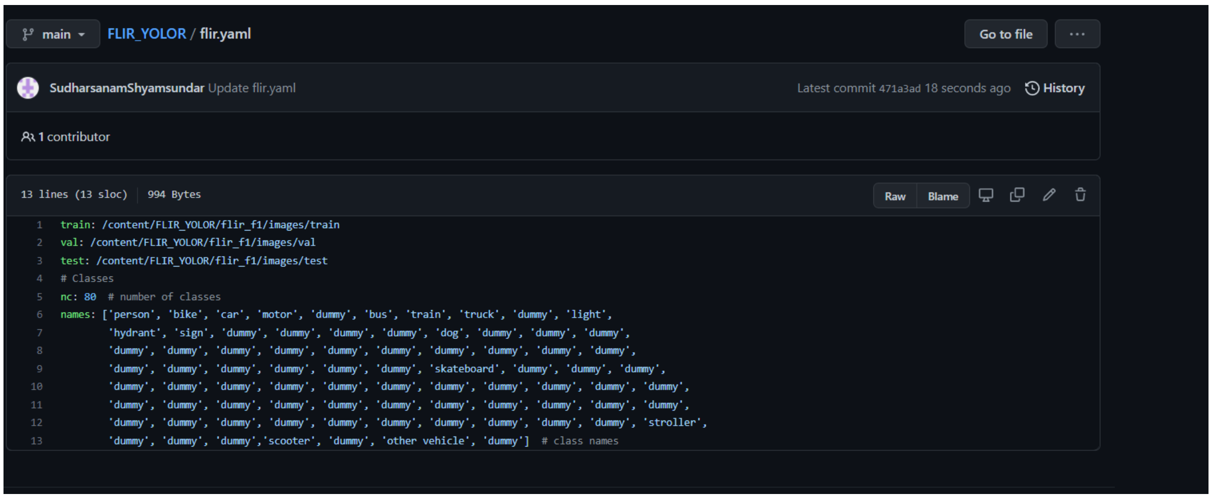
```
9 0.225781 0.398438 0.039062 0.054688
9 0.005469 0.360352 0.010938 0.087891
11 0.207031 0.327148 0.029688 0.068359
11 0.164844 0.335938 0.129688 0.039062
11 0.263281 0.379883 0.173438 0.056641
11 0.200000 0.138672 0.259375 0.253906
11 0.005469 0.270508 0.010938 0.091797
11 0.036719 0.384766 0.048438 0.023438
11 0.405469 0.541992 0.026563 0.056641
0 0.971094 0.547852 0.054688 0.193359
0 0.835938 0.566406 0.065625 0.277344
0 0.676563 0.588867 0.109375 0.271484
0 0.512500 0.595703 0.084375 0.230469
3 0.933594 0.806641 0.129688 0.382812
2 0.058594 0.576172 0.073438 0.074219
2 0.079688 0.547852 0.084375 0.048828
2 0.300781 0.538086 0.114063 0.072266
2 0.539844 0.510742 0.139063 0.064453
2 0.900781 0.472656 0.126562 0.062500
0 0.960938 0.782227 0.075000 0.431641
```

5. Figure, Class Id and bounding box from image

As you can see in input image, there are 4 persons and hence the class id(0) can be found 4 times . Those numbers present are class Id, center x , center Y , width , height. Why zero(ClassId) is assigned for Person is because in coco the first index position is assigned for person and hence it extracted as 0. Note that FLIR has adapted the COCO where it has 80 class in total

5.2 YAML File Creation :

We need to create the new config file (YAML) as we are training the model in custom dataset . The train and test script uses the yaml file as a parser to fetch the images , which should contain the path where the images are located , total number of classes and the class names . For reference you can find the coco.yaml under data subdirectory . Note that it should contain all test, train and validation folder path


main
FLIR_YOLOR / flir.yaml
Go to file
SudharsanamShyamsundar Update flir.yaml
Latest commit 471a3ad 18 seconds ago
History
1 contributor
13 lines (13 sloc) | 994 Bytes
Raw
Blame

```yaml
train: /content/FLIR_YOLOR/flir_f1/images/train
val: /content/FLIR_YOLOR/flir_f1/images/val
test: /content/FLIR_YOLOR/flir_f1/images/test
# Classes
nc: 80  # number of classes
names: ['person', 'bike', 'car', 'motor', 'dummy', 'bus', 'train', 'truck', 'dummy', 'light',
        'hydrant', 'sign', 'dummy', 'dummy', 'dummy', 'dummy', 'dog', 'dummy', 'dummy', 'dummy',
        'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy',
        'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'skateboard', 'dummy', 'dummy', 'dummy',
        'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy',
        'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy',
        'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'dummy', 'stroller',
        'dummy', 'dummy', 'dummy','scooter', 'dummy', 'other vehicle', 'dummy']  # class names
```

6. Figure, YAML Config file for YOLOR Training

We have placed dummy , in order to differentiate from the coco.yaml because FLIR dataset has only 15 categories

5.3 Project Setup :

We have to place the images and labels directory under the parent directory and subdirctories of each one should contain the train, validation and the test.
Since there are around 10k+ images in the dataset, we placed the dataset in the Kaggle and downloaded using the Kaggle API command. This resulted in a faster download and avoids the necessity to rely on the google drive , later on downloaded files are moved to the respective folders(Training , test and Validation) using the python script

https://www.kaggle.com/datasets/benceveges/yolo-flir

Furthermore ,the necessary libraries(mish-cuda, wavelets) are installed inside the YOLOR project Repository :

5.4 Training:

Github : https://github.com/WongKinYiu/yolor

The YOLOR was trained with the same strategy of YOLO V4 with 300 epochs and again 150 for finetuning , but this requires a significant amount of GPU resources.

For training the model, I have used the NVIDIA cloud and the google Colab which has provided the T4 GPU .I ran for only 10 epochs with the batch size of 24 (feeding 24 samples at a time) in a distributed fashion .Most Important , Didn't used any pretrained weights and trained model from zero

Weight Parameters: 37M (37265016)
Total Layers: 665
Epochs: 10
Batch Size :24
Number of iterations = total samples / batch size

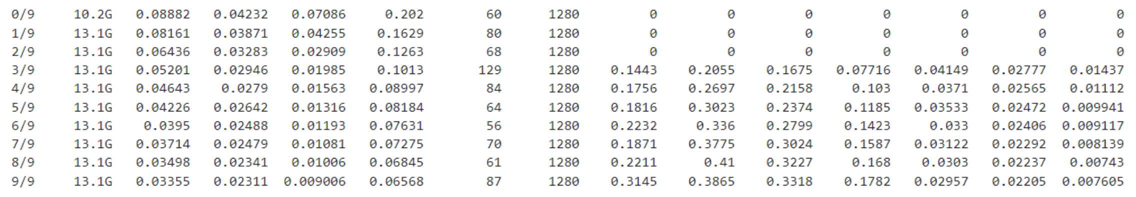
All other parameters were passed in argparser (configuration, weights)

```
!python -m torch.distributed.launch --nproc_per_node 4 --
master_port 9527 train.py --batch-size 24 --img 1280 1280 --
data flir.yaml --cfg cfg/yolor_p6.cfg --weights '' --device 0,1,2,3 --
sync-bn --name yolor_p6 --
hyp '/dli/task/FLIR_YOLOR/data/hyp.scratch.1280.yaml' --epochs 10
```

The weights are not initialized as you see weights ' '

After the training, you can see the results under runs folder(results.txt) where for each epoch precision , recall , map details can be found

Results :



```
     0/9     10.2G   0.08882   0.04232   0.07086     0.202        60      1280         0         0         0         0         0         0         0
     1/9     13.1G   0.08161   0.03871   0.04255    0.1629        80      1280         0         0         0         0         0         0         0
     2/9     13.1G   0.06436   0.03283   0.02909    0.1263        68      1280         0         0         0         0         0         0         0
     3/9     13.1G   0.05201   0.02946   0.01985    0.1013       129      1280    0.1443    0.2055    0.1675   0.07716   0.04149   0.02777   0.01437
     4/9     13.1G   0.04643    0.0279   0.01563   0.08997        84      1280    0.1756    0.2697    0.2158     0.103    0.0371   0.02565   0.01112
     5/9     13.1G   0.04226   0.02642   0.01316   0.08184        64      1280    0.1816    0.3023    0.2374    0.1185   0.03533   0.02472  0.009941
     6/9     13.1G    0.0395   0.02488   0.01193   0.07631        56      1280    0.2232     0.336    0.2799    0.1423     0.033   0.02406  0.009117
     7/9     13.1G   0.03714   0.02479   0.01081   0.07275        70      1280    0.1871    0.3775    0.3024    0.1587   0.03122   0.02292  0.008139
     8/9     13.1G   0.03498   0.02341   0.01006   0.06845        61      1280    0.2211      0.41    0.3227     0.168    0.0303   0.02237   0.00743
     9/9     13.1G   0.03355   0.02311  0.009006   0.06568        87      1280    0.3145    0.3865    0.3318    0.1782   0.02957   0.02205  0.007605
```

7. Figure, Results after 10 epochs completed

5.5 Metrics:

Mean average precision is the most common metric value which is used for Object detection , it is like average precision is calculated by taking the mean AP over all classes. The true positives and false positives are totally depended on the IOU threshold. Instance if we give the IOU threshold as 0.7 and your IOU is 0.8 then it is considered as True positive . On the other hand, if the IOU is less than 0.7 it is considered as false positive . That's why the ideal IOU threshold is chosen after wise consideration .

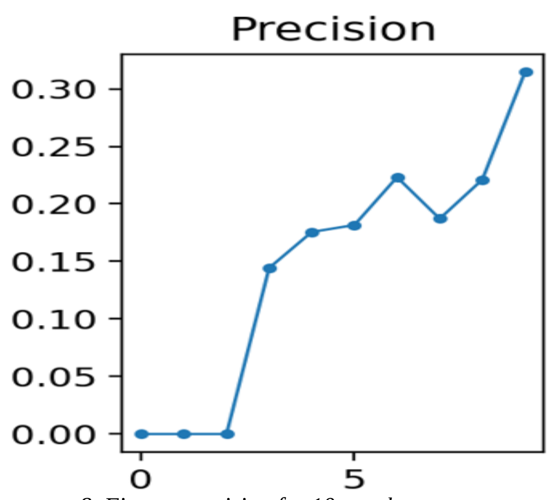
mAP doesn't mean it was average of precision , here average precision means finding the area under precision recall curve and is averaged over all categories to find mAP

Precision :

How accurate your predictions are correct :

$$Precision = \frac{\sum TP}{\sum TP+FP} \tag{5.1}$$

TP – True Positive (predicted as positive it was correct)
FP - False Positive(predicted as positive it was incorrect)



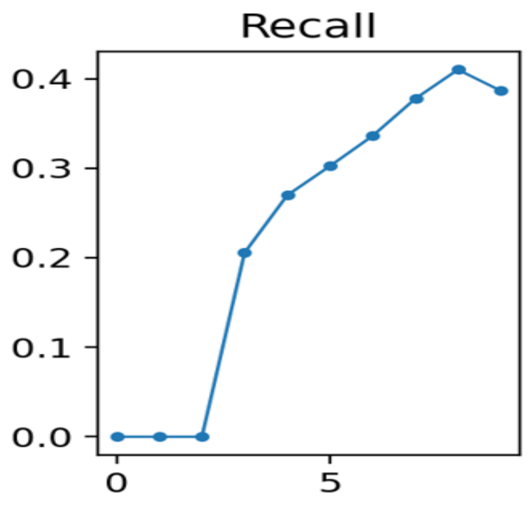
8. Figure, precision for 10 epochs

The precision is increasing in each epoch as you see at the end of 10th epoch it clocked at 0.32

Recall:
It measures how well you find all positives ;

$$Recall = \frac{\sum TP}{\sum TP+FN} \qquad (5.2)$$

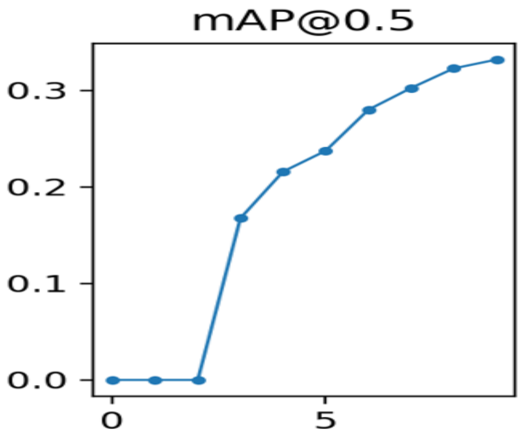
TN : True Positive (predicted as positive and was correct)
FN : False Negative (Failed to predict an object it was present)



9. Figure, Recall for 10 epochs

The recall is increasing in each epoch up until the 9th epoch where it is 0.41 as you see ,at the end of 10th epoch it is 0.38. We couldn’t able to run for more epochs as a result of GPU shortage and as an alternative it can be improved by placing the true samples (which means if want to localize more cars , you should have more cars in your image so that recall will improve)

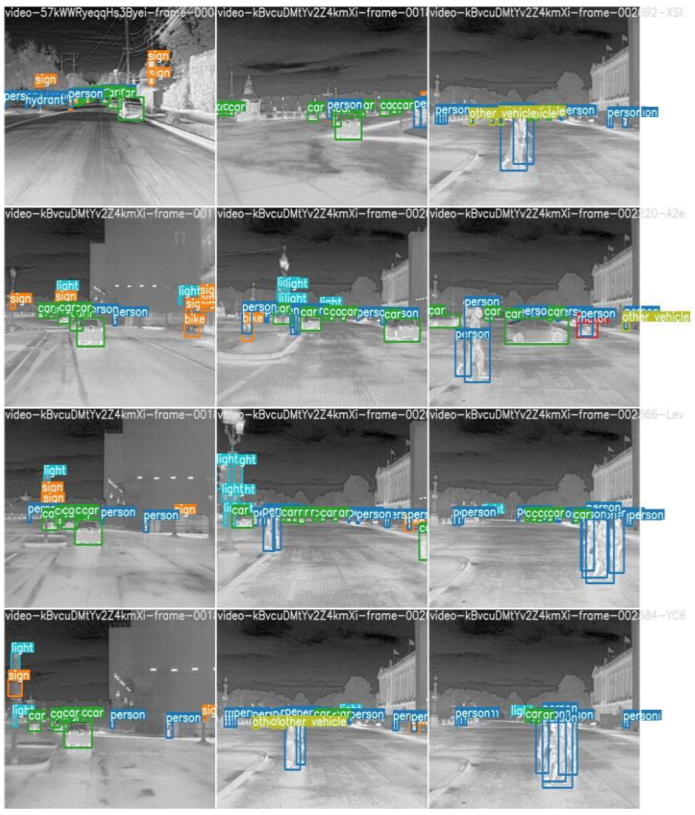
The mean Average precision at IOU 0.5(threshold) at the end of 10th epoch is 0.306, when trained with the thermal images



10. Figure, mAP for 10 epochs

All the results loges can be found under the link
https://github.com/SudharsanamShyamsundar/FLIR_YOLOR

Sample predictions by the model is shown below

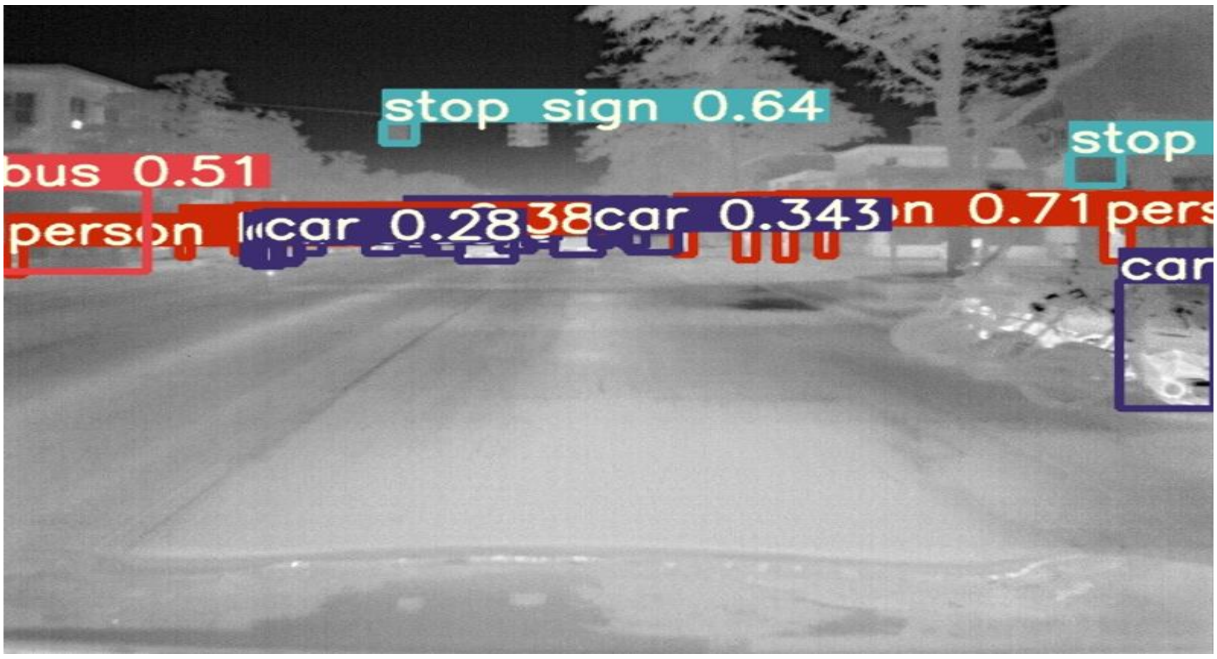

11. Figure, prediction for validation dataset

6.Testing and Detection :

Detection : After 10 epochs , the model has been used to detect the objects

While we are trying to detect the objects present , the good amount of false positive can be seen from the test image, as you can see the motorcyles are present in image, but the model has predicted it as car for one of motorcycle . On the other hand , it doesn't predicted the motorcycle at all clear indication of false negatives

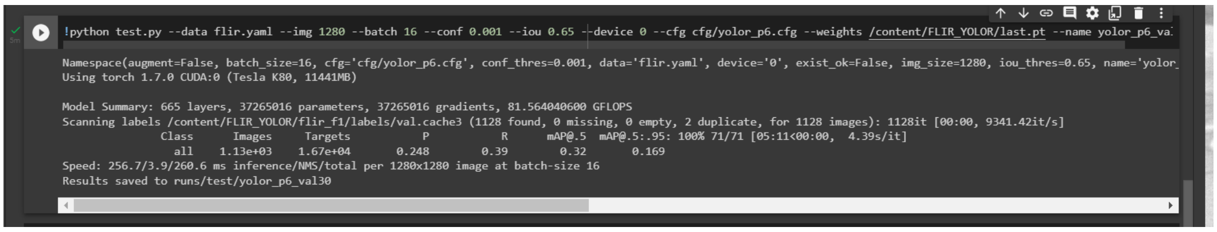
12. Figure, one of test dataset used for detection



13. Figure, test dataset detection resulted in False positive and False negative

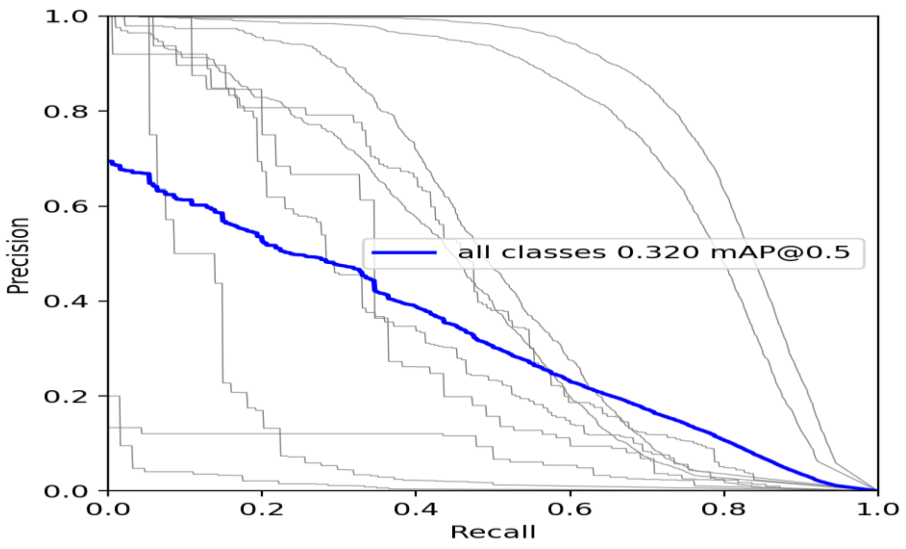
Testing**:**

After training , the weights (last.pt) are passed as parser and the other configurations as originally used for testing the yolor was retained including the IOU



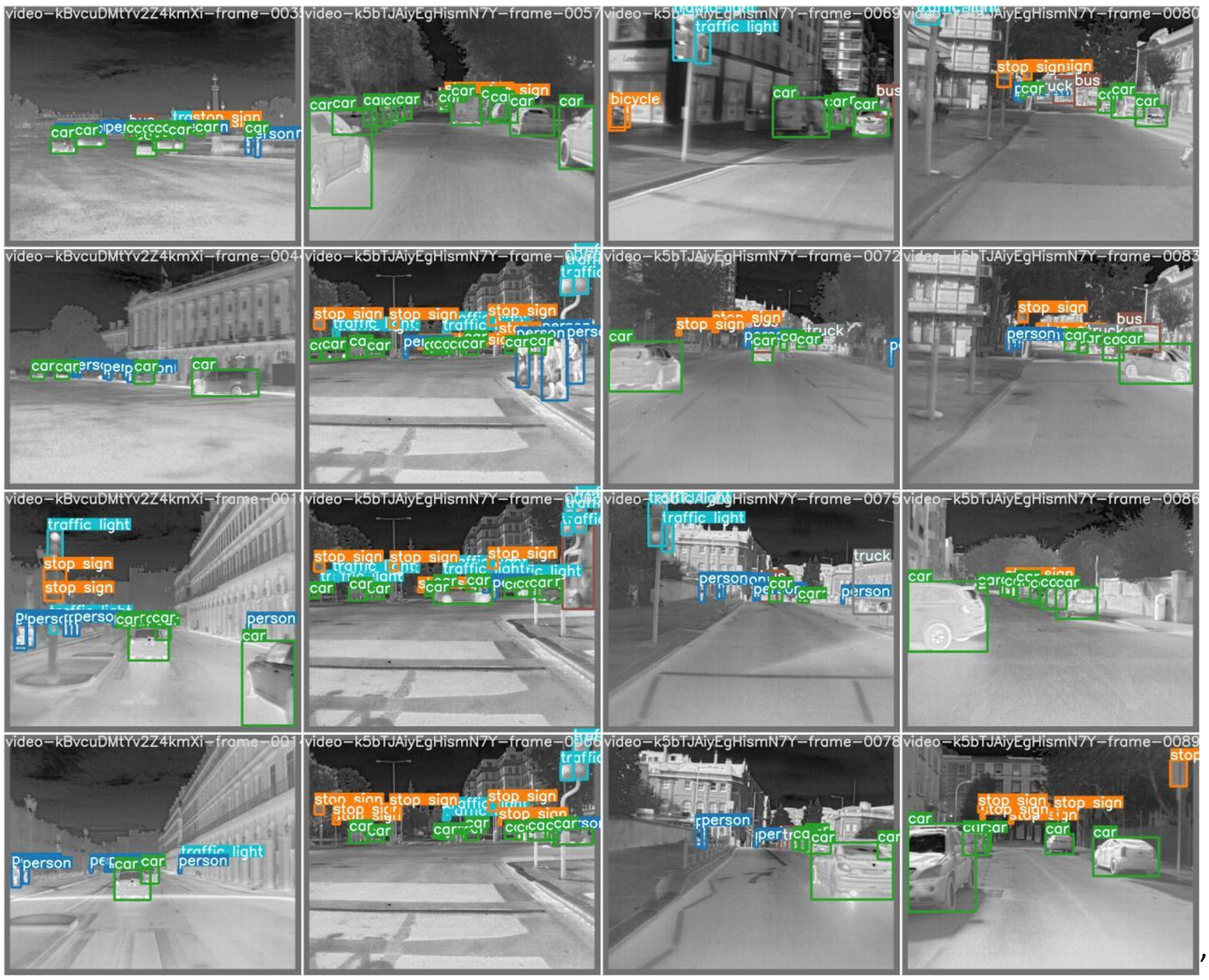
```
!python test.py --data flir.yaml --img 1280 --batch 16 --conf 0.001 --iou 0.65 --device 0 --cfg cfg/yolor_p6.cfg --weights /content/FLIR_YOLOR/last.pt --name yolor_p6_va

Namespace(augment=False, batch_size=16, cfg='cfg/yolor_p6.cfg', conf_thres=0.001, data='flir.yaml', device='0', exist_ok=False, img_size=1280, iou_thres=0.65, name='yolor_
Using torch 1.7.0 CUDA:0 (Tesla K80, 11441MB)

Model Summary: 665 layers, 37265016 parameters, 37265016 gradients, 81.564040600 GFLOPS
Scanning labels /content/FLIR_YOLOR/flir_f1/labels/val.cache3 (1128 found, 0 missing, 0 empty, 2 duplicate, for 1128 images): 1128it [00:00, 9341.42it/s]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100% 71/71 [05:11<00:00,  4.39s/it]
                 all    1.13e+03    1.67e+04       0.248        0.39        0.32       0.169
Speed: 256.7/3.9/260.6 ms inference/NMS/total per 1280x1280 image at batch-size 16
Results saved to runs/test/yolor_p6_val30
```

14. Figure, test script used for testing the model



15. Figure, precision recall curve for IOU @0,5 after 10 epochs

The mean Average Precision is very less here as you see , it is just **0.320** , This is as a result of running 10 epochs, In the research paper, they have run 300 epochs. We don’t have the resouces to run this much epochs


,

16. Figure, Object detections for test dataset

7.Conclusion:

We were able to test,train and detect , unfortunately because of limited GPU resources we cant able to run for more epochs as a result the mAP is less . The same version is compared with the YOLO V5 and this results almost same mAP is evident there are number of false positive and false negative arised. As a initial idea to investigate the model performance on detecting the IR datasets using YOLOR model has resulted in almost same performance with YOLOV5 , however the author says we can perform different kinds of tasks using this model which we haven’t explored . Most importantly, huge resources is required to train the model for 300 epochs which we don’t have the facility and sticked to only 10 epochs and trained the model from scratch without using the pretrained weights

References

[1] Open Source - https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31

[2] Open Source - https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31

[3] YOLOR paper -https://arxiv.org/pdf/2105.04206.pdf

[4] Yolor source code - https://github.com/WongKinYiu/yolor

[5] YOLOV5 source code - https://github.com/ultralytics/yolov5

[6] IEEE Journal on Object detection in Thermal Spectrum for ADAS - https://arxiv.org/ftp/arxiv/papers/2109/2109.09854.pdf

List of figures

List of tables