Assignment 4 - a YOLO Neural Network

This assignment has two phase , one was the training on the coco dataset with the yolov5s model -on random initialize weight - which we will take about its architecture now , the second phase was training with the yolov5l on a custom dataset we made with pre trained model value

Brief of Datasets:

Coco is a computer vision dataset which is made by CVDF, Microsoft, Facebook, and Mighty AI. It be used for object segmentation, recognition in contest and super pixel stuff segmentation.

Coco has 80 object categories, 91 stuff categories and 5 captions per image. The dataset now has 328k images which more than 200k are labeled.

The first coco dataset released was in 2014 and has 83k train images and 41k images for validation and another 41k image for testing. the second released in 2015 by updating only the test images from 41k to 81k images. The last one which we use in our training is the coco2017 released in 2017 and has 118k training images, 5k validation images and 41k test images and another 123k unlabeled images. All those pictures are annotated and ready to train.

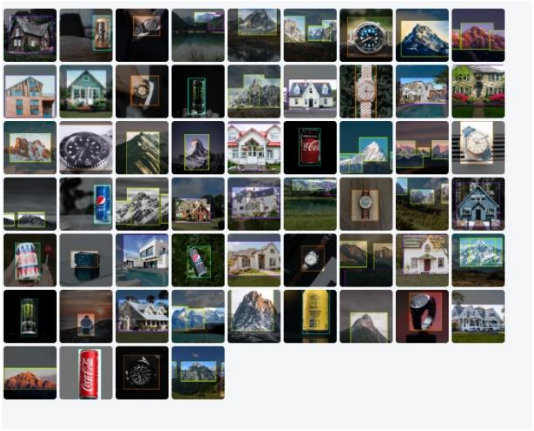
Coco dataset is used to evaluate various computer vision models, so it's used as a standard metric in computer vision.

Custom dataset I made a totally new custom dataset that have 5 object classes (mountains, forest, house, watch, cans) this dataset contains 81 images divided to 58 train pictures, 13 validation pictures and 10 test pictures as per your request.

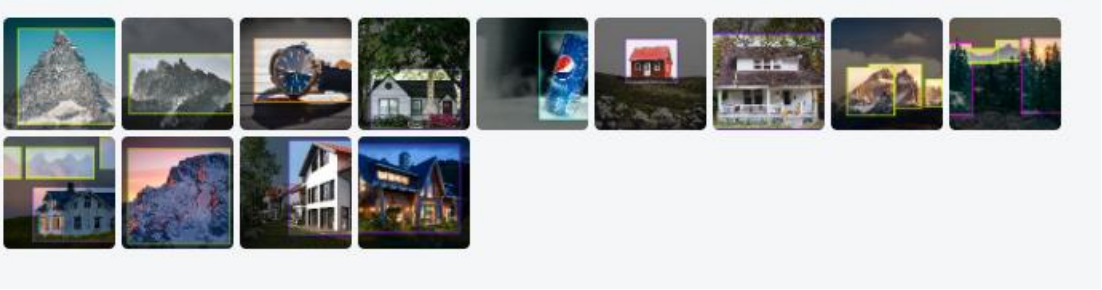
I created this data set from scratch and annotated it. The data is collected from various online sites and used roboflow to annotate it and this site has a partnership with yolo team to integrate the 2 APIs together so roboflow already export my dat**a and the**

annotation to the yolo markup language (.yaml) and the documentation of the integration between the 2 APIs could be found on yolov5 git hub.

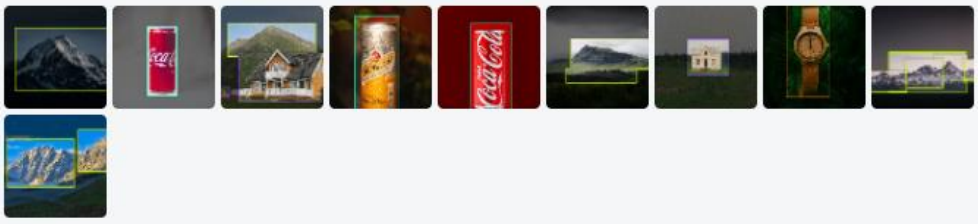
Train images



Validation images



Test images

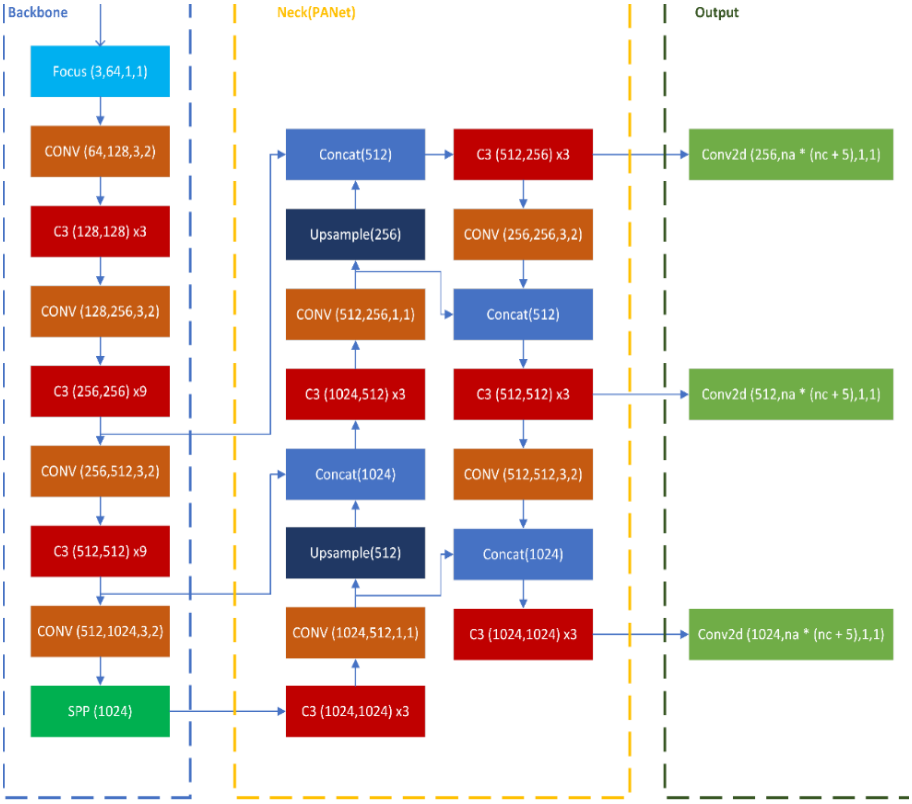


Summarize Yolo Architecture:

Now we will take about the model we used in training the Coco 17 , we choose the small version in order to have less training time as it is a complex problem with a lot of data images

This is not the small Arch of the YOlOV5 , but it is an example of how the arch overall consists of

It has a backbone that contain several convolution layer and C3 (C3 Layer is a CSP bottleneck that includes 3 convolutional layers , Essentially it does a Conv on the input tensor and it contacts the result to the same tensor passed through a convolution AND a series of bottleneck layers with e=1. Then the whole thing is passed again through a Convolution layer)



CSP stands for Cross Stage Partial layer

As per the first column, it is used in the forward function of the model to understand which tensor to use as the input value of each layer. The majority of the layers has '-1', meaning they take the last layer's output before them as their input, but there are Concat layers that take different levels as input to recreate the PANet(Head) architecture in the neck

YoloV5 Small component :

Backbone :

```
Conv, [64, 6, 2, 2
Conv, [128, 3, 2]
C3, [128,3 ,1-]
Conv, [256, 3, 2 ]
C3, [256 ,6 ,1-]
```

```
C3, [1024,3 ,1-]
SPPF, [1024, 5 ]
Conv, [512, 3, 2]
C3, [512 ,9 ,1-]
Conv, [1024, 3, 2 ]
```

Head :

```
Conv, [512, 1, 1]],
nn.Upsample, [None, 2, 'nearest']],
Concat, [1]],  # cat backbone P4
C3, [512, False]],  # 13
Conv, [256, 1, 1]],
nn.Upsample, [None, 2, 'nearest']],
Concat, [1]],  # cat backbone P3
C3, [256, False]],  # 17
```

```
Conv, [256, 3, 2]],
Concat, [1]],  # cat head P4
C3, [512, False]],  # 20 (P4/16-medium)
Conv, [512, 3, 2]],
Concat, [1]],  # cat head P5
C3, [1024, False]],  # 23 (P5/32-large)
Detect, [nc, anchors]],  # Detect(P3, P4, P5)
```

Grid size :

from Train.py grid size is the maximum of stride it use 3 multi-scale outputs at strides 8, 16 and 32

```
gs = max(int(model.stride.max()), 32)  # grid size (max stride)
```

Number of anchors:

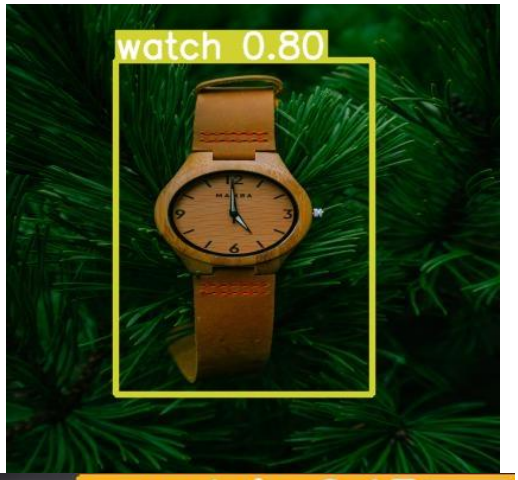
```
anchors:
  - [10,13, 16,30, 33,23]  # P3/8
  - [30,61, 62,45, 59,119]  # P4/16
  - [116,90, 156,198, 373,326]  # P5/32
```

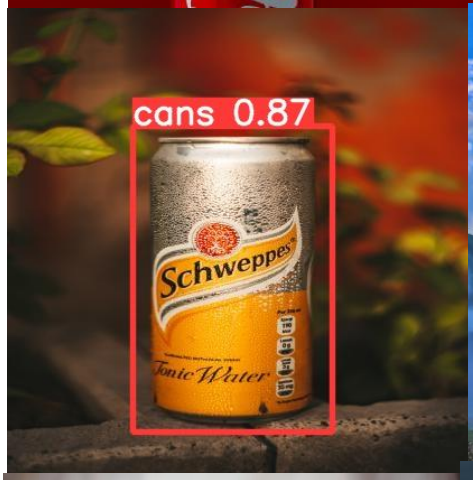
loss function :

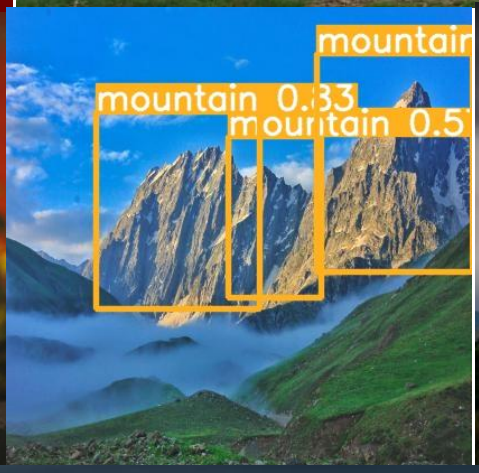
In the YOLO family, there is a compound loss is calculated based on objectness score, class probability score, and bounding box regression score

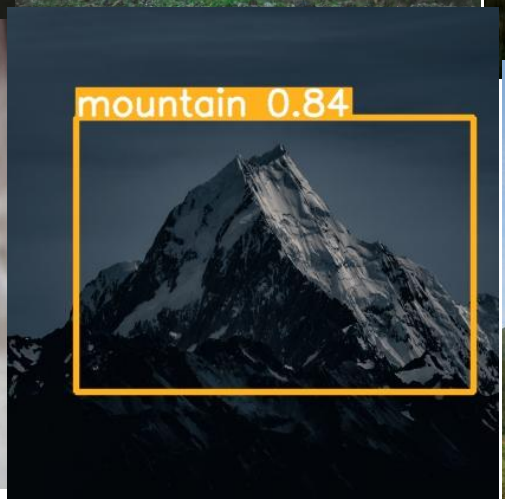
Ultralytics have used Binary Cross-Entropy with Logits Loss function from PyTorch for loss calculation of class probability and object score.

Predicted Images :









Time for each Prediction :

image 1/9 416x416 1 house, 1 mountain, Done. (0.032s)

image 2/9 jpg: 416x416 1 cans, Done. (0.031s)

image 3/9.jpg: 416x416 1 forest, 2 mountains, Done. (0.031s)

image 4/9 /.jpg: 416x416 1 house, Done. (0.031s)

image 5/9 / jpg: 416x416 1 cans, Done. (0.031s)

image 6/9 jpg: 416x416 1 cans, Done. (0.031s)

image 7/9.jpg: 416x416 1 watch, Done. (0.019s)

image 8/9.jpg: 416x416 3 mountains, Done. (0.018s)

image 9/9.jpg: 416x416 1 mountain, Done. (0.023s)

we use IOU_thresh of default Value = 0.45in all predicted images