Project: Detect cyclist in image/video

1. Definition

Overview

Object detection is an important computer vision task that allows a model to classify a label to the identified object and localize it in the given image. I am motivated by the idea that a camera based sensing device can go through the visuals of our physical surroundings and interpret the world like humans do. This has a major impact on how we as society interpret objects, resources and interact with them.

I would like to understand how vision sensing can process the vision signals and derive useful information like label, texture, orientation etc. from it. **I like outdoors and specially bicycling and would like to understand bicyclists on streets.**

Wikipedia describes this task as "***Object detection*** *is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos*. *Well-researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance*"

A seminal work in this domain is named "Viola–Jones object detection framework" which was published by Viola, P. and Jones, M. in the proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition

Some of the recent works focus on Neural Network and Deep Learning based approaches. It is well described in a review by Rohith Gandhi in a towardsdatascience.com blog titled "R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms"

We can use one of these algorithms based on their benchmark to propose a solution.

Problem Statement

Given an outdoor image frame, detect bicyclists and all the other objects in view.

- Detection of bicyclist should localize its position with respect to the image frame
- Draw a bounding box in the output image frame with a label classification confidence score.

We measure the confidence score in percentage for classification and evaluate object detection performance using AP (Average Precision) metric

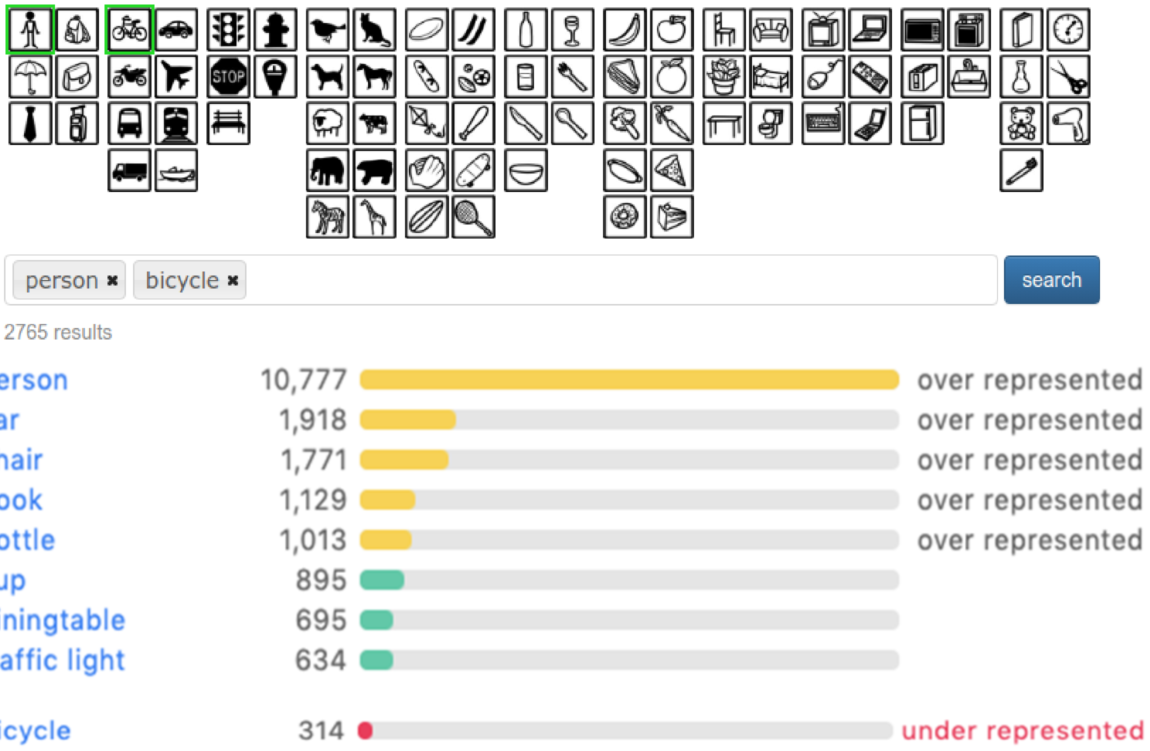
2. Analysis

A. Data Exploration

We will use a sampled version of the MS COCO dataset (https://cocodataset.org) since our storage and compute needs are to be kept low.

Figure: Annotated or labeled images for objects in MS COCO dataset

COCO is a large-scale object detection, segmentation, and captioning dataset. COCO has several features. However, we will use the object instance detection from its annotated 1.5 million object instances with 80 classes and median image ratio is 640x480.



2765 results

person	10,777	over represented
car	1,918	over represented
chair	1,771	over represented
book	1,129	over represented
bottle	1,013	over represented
cup	895	
diningtable	695	
traffic light	634	
bicycle	314	under represented

Figure: Typical objects annotation distribution in the dataset (low bicycle count), above is the explore webpage for MS COCO annotation label based search functionality.

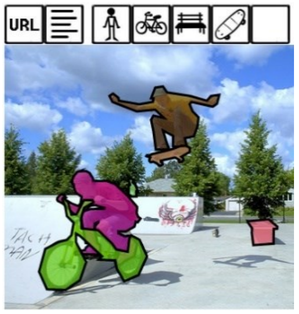
In the figure above, we can see the various label distributions and representation in the entire dataset. An explore functionality in the webpage (https://cocodataset.org/#explore) or pycocotools dataset API is used for this purpose.

We use MS COCO dataset explorer to visualize the existing data and visually understand the distribution by searching for a particular annotation label like “person” or “bicycle”. There are about 2765 images in train & validation set which contains both these labels.

```
“image”: [{
    “id”: 1342334,
    “width”: 640,
    “height”: 640,
    “file_name: “ford-t.jpg”,
    “license”: 1,
    “date_captured”: “2022-02-01  15:13”
}]
```

```
“annotations”: [{
    ”segmentation”: [[34, 55, 10, 71, 76, 23, 98, 43, 11, 8]],
    “area”: 600.4,
    “iscrowd”: 1,
    “Image_id:” 122214,
    “bbox”: [473.05, 395.45, 38.65, 28.92],
    “category_id”: 15,
    “id”: 934
}]
```

Figure: JSON format for raw image metadata and corresponding annotations with label & position



We will describe the annotated dataset. The data itself has various categories in annotation and “category_id” labels one of the bounding boxes for an image.

Let us look at one of the images that contains bicycle and person labels in the MS COCO dataset explore web page. We will use these annotations to create a sampled dataset to learn person and bicycle classes. In our model building we want to learn a new class “cyclist” in the super category person. Next section will describe the sampling strategy and updates to the annotation.

B. Data Sampling & Visualization

To detect cyclist in an image we use a computer vision task called object detection. There is no dataset available specifically for this outcome. Hence, we use the MS COCO dataset and sample the images where we observe people with bicycles and annotate them as cyclist. We will see a sampling approach in the dataset that makes use of IoU (Intersection over Union) approach to find overlap in the annotations and assign a label to person, categorizing them as cyclist.

There are about 2643 images in train & 122 in validation set which contains person and bicycle labels. We use the below workflow to create our label of interest.

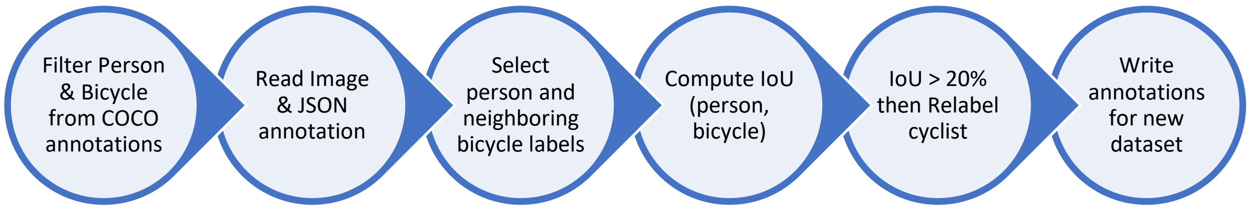

Filter Person & Bicycle from COCO annotations
Read Image & JSON annotation
Select person and neighboring bicycle labels
Compute IoU (person, bicycle)
IoU > 20% then Relabel cyclist
Write annotations for new dataset

Figure: Workflow for processing the dataset and updating the label as cyclist.

Deep learning model will read training data files with these annotation labels and bounding box during the training phase and then later in the evaluation phase we understand the models overfitness to the data and do model selection.

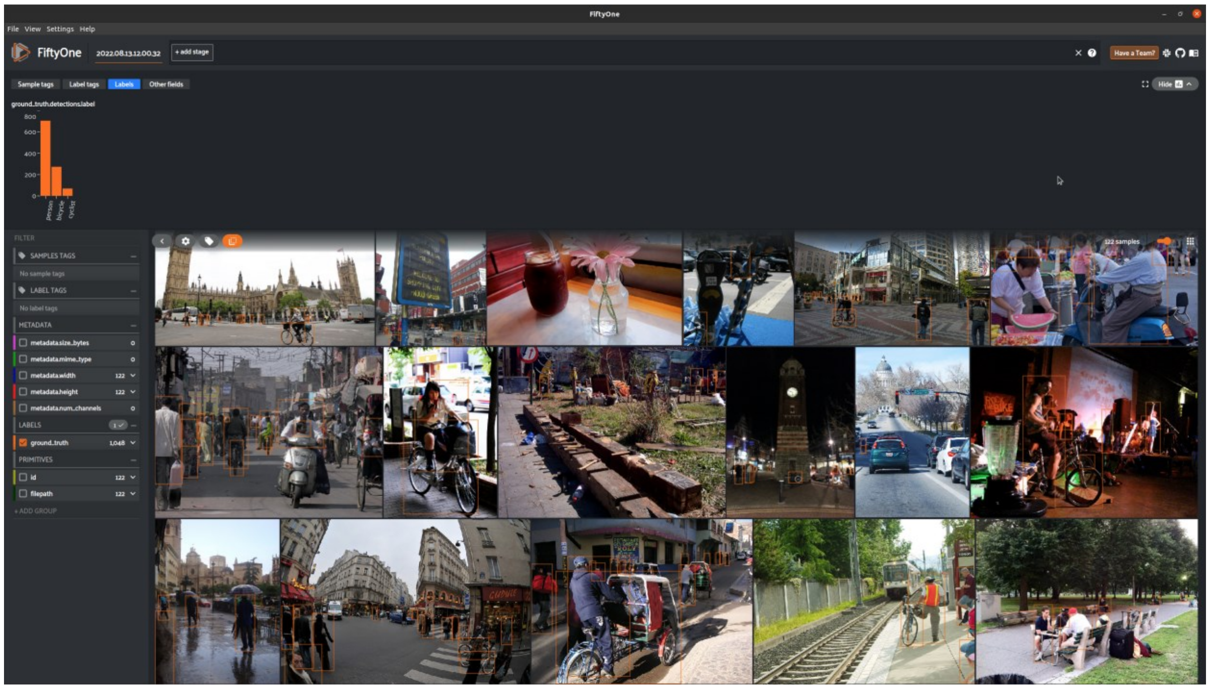


Figure: Visualize the labels in FiftyOne. There are 706 person, 273 bicycle and 69 cyclist labels in the created annotations for the object detection task.

- To visualize the new updated dataset, we use an open sourced tool FiftyOne from Voxel51 (https://voxel51.com/docs/fiftyone/index.html).
- This procedure is covered in the sampling notebook at https://github.com/karmarv/udacity-aws-sagemaker-capstone/blob/main/samples/coco_sampler.ipynb

C. Algorithms & Techniques

Object detection is the computer vision based approach to classify and localize objects in an image. We use the dataset generated above to supervise and train a Deep learning based model to detect cyclist in an image.

Current object detectors can be divided into two categories: Networks separating the tasks of determining the location of objects and their classification, where Faster R-CNN is one of the most famous ones, and networks which predict bounding boxes and class scores at once, with the YOLO and SSD networks being famous architectures.

We would like to approach by using a Yolo based approach to detect objects in the given image. It predicts the bounding boxes and classes for the whole image in one run of the algorithm, instead of selecting the inserting parts only. In YOLO, the architecture splits the input image into m x m grid, and then further each grid generates 2 bounding boxes and the class probabilities of those bounding boxes.

Advantages:

- Faster Speed
- It is a highly generalized network
- It processes each frame at the rate of 45 fps for a larger network and 150 fps for a smaller network and near real-time.

Disadvantages:

- It is very difficult to detect small objects from the image.
- It is very difficult to detect objects which are very close to each other because of the grid.
- It has comparatively more localization and low recall error compared to faster RCNN.

Yolo

The YOLO algorithm was first introduced in 2015 to solve the object detection problem. As the name suggests, the object detection problem is when a computer is given an image and has to detect where certain objects in that image are.

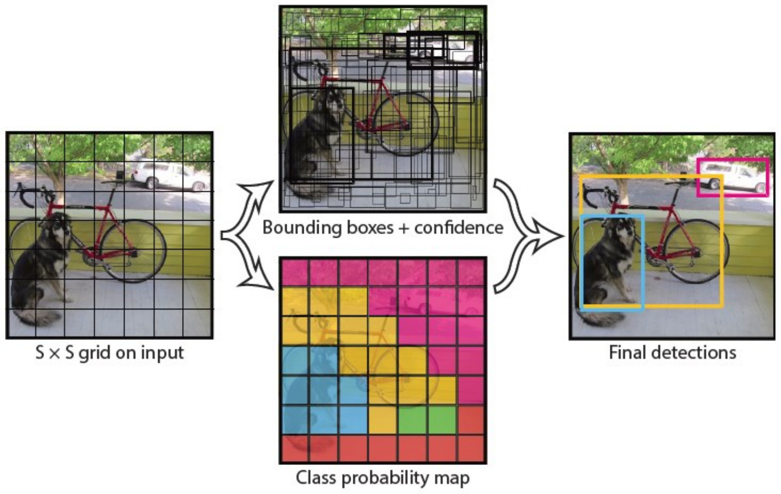


Figure: Input is 448×448×3 tensor representing the RGB values. output is a 7×7×30 tensor with grid size S=7.

For each grid cell model predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B ∗ 5 + C) tensor.

This simplified structure allows for a simple output layer enabling high processing throughput in FPS.

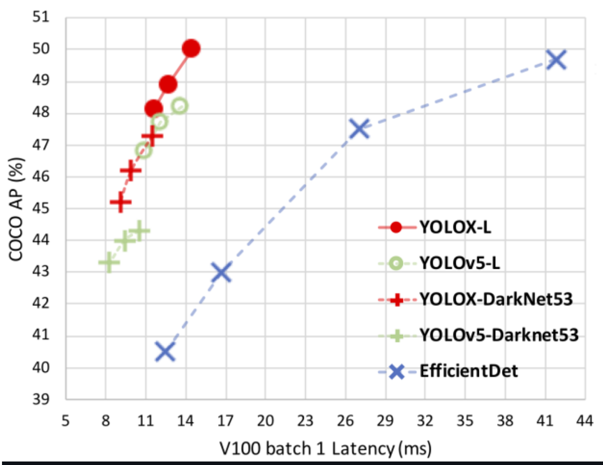
The YOLO algorithm works by predicting three different features: (a.) **Bounding box**: (25, 5, 185, 183), (b.) **Confidence**: 0.9567, (c.) **Class**: cyclist. The bounding box represent the location of the object in the image and confidence is the probabilistic score of the classification. The YOLOv1 algorithm takes the image as input and outputs a tensor that can be broken up into the three properties of a bounding box: location, confidence, and class.

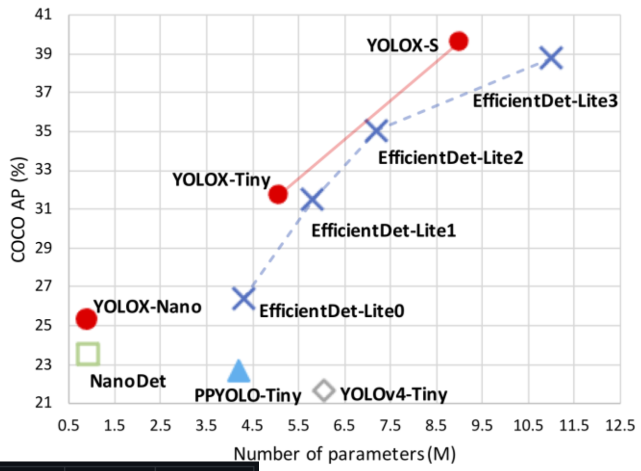
YoloX

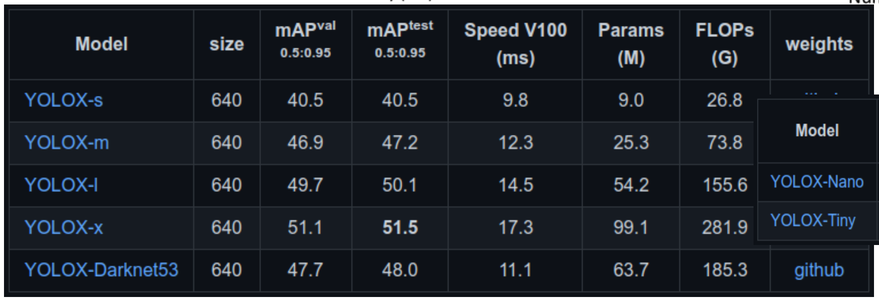
The YOLOX uses a Darknet-53 backbone (FPN or Feature Pyramid Network) to learn features like YoloV3. The FPN extracts information at a different scale. As the number of channels increases, the length and width of the image decreases. So, the 256 channel transition output extract features at a smaller scale while the 1024 channel output extracts features at a larger scale since the 1024 channel output has less information from the original image to work with. The major difference between YoloV3 and YoloX is in the head. We will discuss that in the architecture or implementation section.

D. Benchmark

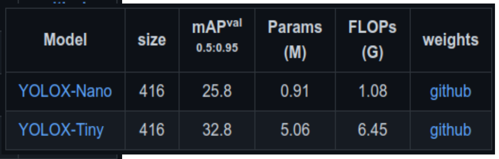
We look at the various benchmarks reported for YOLOX in terms of mAP (mean Average Precision) which is precision score averaged over IoU (0.5 to 0.95 in steps of 0.05).







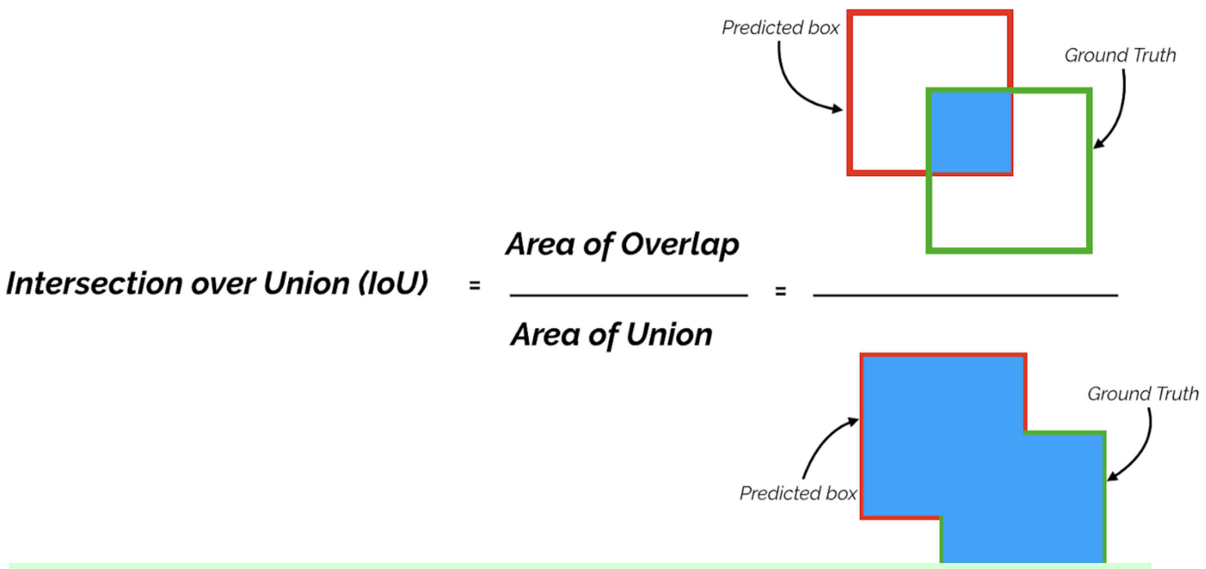
Model	size	mAP^val 0.5:0.95	mAP^test 0.5:0.95	Speed V100 (ms)	Params (M)	FLOPs (G)	weights
YOLOX-s	640	40.5	40.5	9.8	9.0	26.8	
YOLOX-m	640	46.9	47.2	12.3	25.3	73.8	
YOLOX-l	640	49.7	50.1	14.5	54.2	155.6	
YOLOX-x	640	51.1	**51.5**	17.3	99.1	281.9	
YOLOX-Darknet53	640	47.7	48.0	11.1	63.7	185.3	github

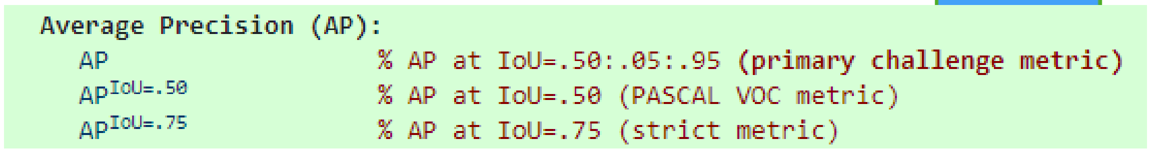


Model	size	mAP^val 0.5:0.95	Params (M)	FLOPs (G)	weights
YOLOX-Nano	416	25.8	0.91	1.08	github
YOLOX-Tiny	416	32.8	5.06	6.45	github

Figure: (above) Comparative benchmark for the latency and number of parameters in the model. (below) table represents the mAP metric for various model size evaluated on MS COCO dataset.

In our evaluations, we would like to use the YoloX nano and small variant for our evaluation considering the low computational requirements of these models.


Predicted box
Ground Truth
Intersection over Union (IoU) = Area of Overlap / Area of Union =
Ground Truth
Predicted box


```
Average Precision (AP):
   AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
   AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
   AP^IoU=.75          % AP at IoU=.75 (strict metric)
```

Average precision is the metric used to evaluate COCO dataset and it related with how accurately and object is detected by the object detection model inference step.

3. Methodology

A. Preprocessing

The dataset contains 706 person, 273 bicycle and 69 cyclist labels in the sampled/updated annotations for the object detection task. This step sets the dataset to be loaded with simple augmentation like flip, rotate and also includes the mosaic augmentation approach. The mosaic augmentation combines multiple images into a single image like a collage and also maintains the bounding box labelled in the new image. This has shown to perform well on many benchmarks and is also proved in the YoloV4 and YoloV5 papers.

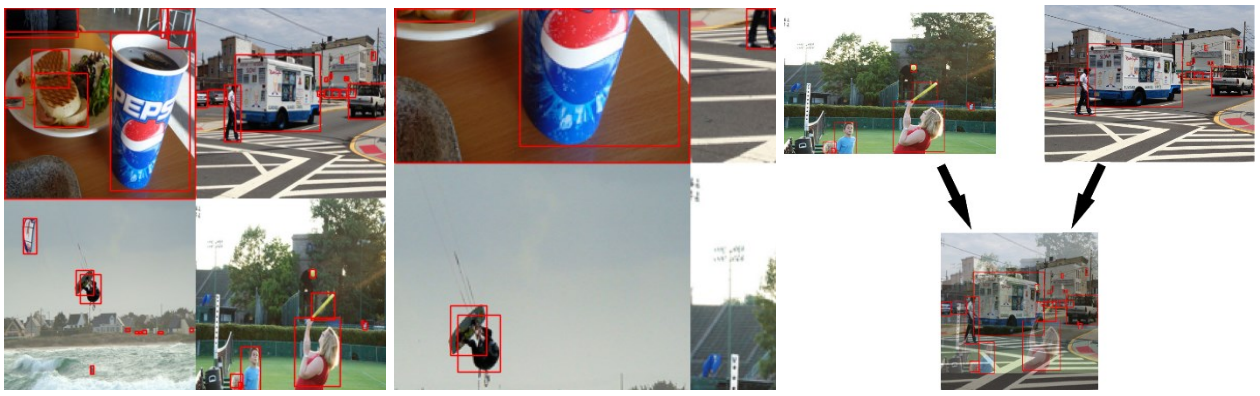


Figure: Strategies for Mosaic and MixUp augmentations in YoloX or YoloV4 model setup

B. Implementation

Architecture and use of YOLOX based model

The input into both the YOLOv3 head and the YOLOX head is the 3 outputs from the FPN (darknet) backbone at three different scales — 1024, 512, 256 channels.

The output of the two heads is essentially the exact same with dimensions (H×W×features) which is just like the original YOLO. The difference between the two heads is that YOLOv3 uses a coupled head and YOLOX uses a decoupled head. So, the output of YOLOX is actually 3 tensors each holding different information instead of 1 massive tensor with all information.

The three tensors YOLOX outputs hold the same information as the tensor YOLOv3 outputs shown in the next figure with YOLO heads.:

1. **Cls**: The class of each bounding box
2. **Reg**: The 4 parts to the bounding box (x, y, w, h)
3. **IoU** (Obj): For some reason, the authors use IoU instead of Obj, but this output is just how confident is the network that there's an object in the bounding box (objectness)

Just like with the original output, each "pixel" in the height and width of the output is a different bounding box prediction. So, there are H*W different predictions.

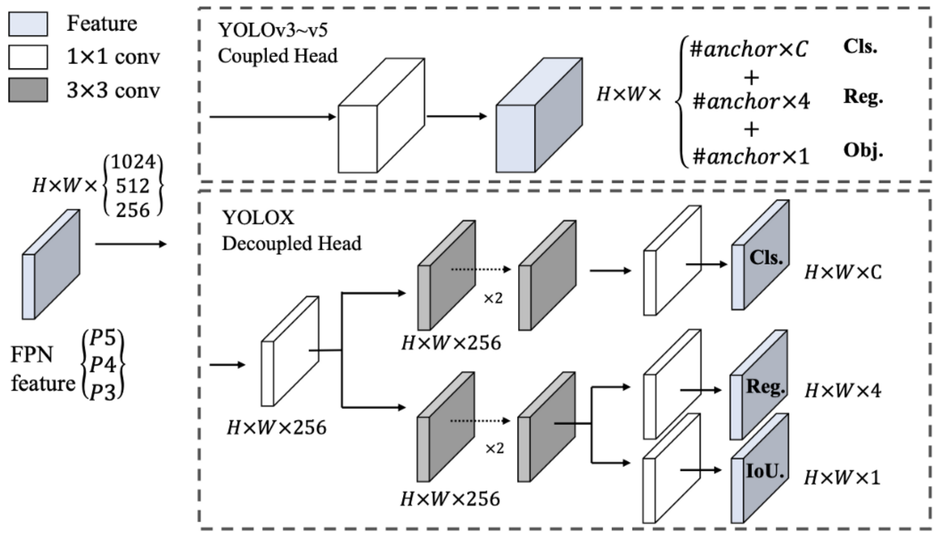


Figure 2: Illustration of the difference between YOLOv3 head and the proposed decoupled head. For each level of FPN feature, we first adopt a 1 × 1 conv layer to reduce the feature channel to 256 and then add two parallel branches with two 3 × 3 conv layers each for classification and regression tasks respectively. IoU branch is added on the regression branch.

Figure: YOLOv3 vs. YOLOX head from the paper: https://paperswithcode.com/method/yolox

The outputs listed above are only for a single output in the FPN. Remember there are three outputs from the FPN which are fed into the heads of YOLOv3 and YOLOX. This means there are actually three different outputs from each of the heads instead of 1. So, the output of YOLOv3 is actually

(3×H×W×features) and the output of YOLOX is actually 3 of each of the Cls, Reg, and IoU (obj) outputs making 9 totals outputs.

Anchor-free prediction

One of the most important changes YOLOX made was not using anchors whereas YOLOv3 heavily relies on anchors. YOLOX simply has the model directly predict the bounding box dimensions as opposed to predicting an offset from an anchor box. To directly predict a bounding box, YOLOX uses a decoupled head as shown in the figure above.

Label Assignment

Not all predictions are equal. Some are clearly garbage and we don't even want our model to optimize them. To differentiate between good and bad predictions, YOLOX uses something called SimOTA which is used for dynamic label assignment. Label assignment helps the model be more stable as it trains. Instead of optimizing all predictions (in YOLOX the number of predictions is somewhere around 1344 for an input image of 256), we can use label assignment to get the best predictions. Then, we can optimize the best predictions to make them even better.

Loss Functions

There are three outputs to the YOLOX model, and each output has its own loss function as they need to be optimized in different ways.

- **Class Loss***:* the class output has the following shape: H×W×C. So for each prediction, the model predicts a vector of C class elements to be chosen from. To optimize these predictions, we can put both the predictions (of shape H×W×C) and the ground truth labels (also of shape H×W×C) through a Binary Cross Entropy (BCE) loss function.
- **Regression Loss**: Optimizing the regression (bounding box prediction) outputs H×W×4 where each prediction is (x, y, w, h). YOLOX uses an evaluation metric called IoU (Intersection Over Union)
- **Objectness Loss**: Like the class loss function, we will use BCE to optimize the objectness predictions.

The final loss function is a combination of the three losses stated above and is defined as follows:

$$L = \frac{1}{N_{pos}} L_{cls} + reg_{weight} * \frac{1}{N_{pos}} L_{reg} + \frac{1}{N_{pos}} L_{obj}$$

Figure: Total loss function

The loss function is basically the sum of all losses averaged over the number of positive labels. We used SimOTA to assign labels to each prediction. reg_weight is a balancing term used to weigh the regression loss over the other losses as it's most important to optimize. The authors use a weight of 5.0.

Non-Max Suppression (NMS)

Non-max suppression is a very good way of pruning bounding boxes without knowing where ground truths are in the image. To do this, the algorithm basically removes predictions with high overlap. The way nonmax suppression removes bounding boxes with a high overlap is by using the IoU score between overlapping bounding boxes. Those with a high IoU are removed so that a single bounding box is kept.

C. Refinement

We run experiments and tune the model (nano, small and medium) with batch 4, 8, 16, 32 etc. We tried higher batch size for smaller models as they fit in the Tesla V100 Ubuntu 20.10 linux instance.

Initially we started with the minimal baseline model of "yolox_nano" which gave us a baseline but did not have the capacity to classify between the person and cyclist class due to low label count of cyclist as well as the small model capacity to learn that feature.

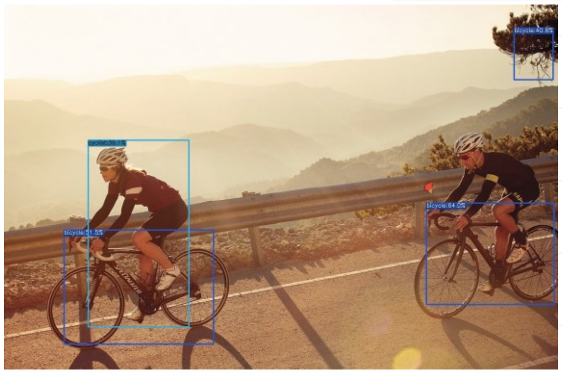
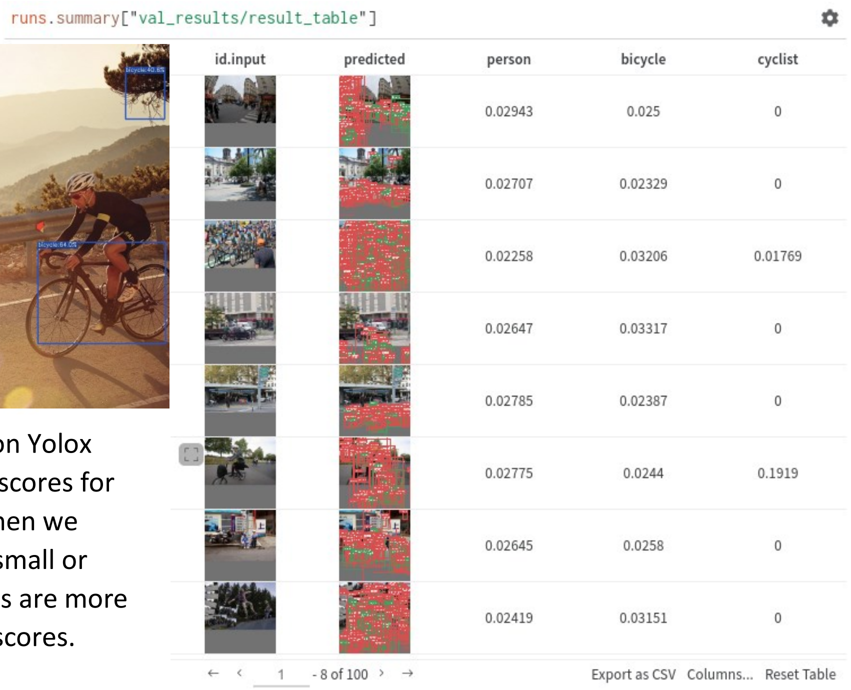


Figure: Baseline model result on Yolox nano variant. We see that the scores for the cyclist is very low. Later when we observe these results on Yolo small or medium variant the predictions are more confident and have better AP scores.



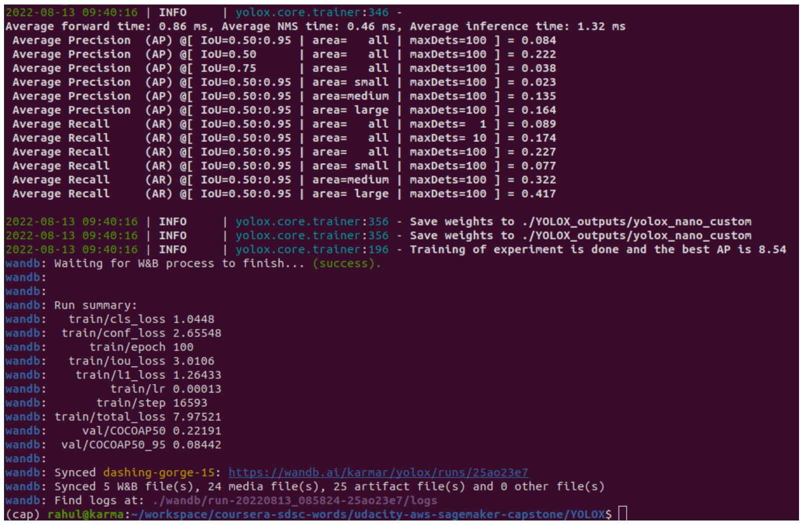
```
runs.summary["val_results/result_table"]
```

id.input	predicted	person	bicycle	cyclist
		0.02943	0.025	0
		0.02707	0.02329	0
		0.02258	0.03206	0.01769
		0.02647	0.03317	0
		0.02785	0.02387	0
		0.02775	0.0244	0.1919
		0.02645	0.0258	0
		0.02419	0.03151	0

We use weights & biases dashboard at https://wandb.ai for comparing the loss convergence as well as the AP score verification.

The various scores that are being calculated by using the COCO Evaluator pycocotools API can be seen in the console log dumped towards the end of the training epochs.

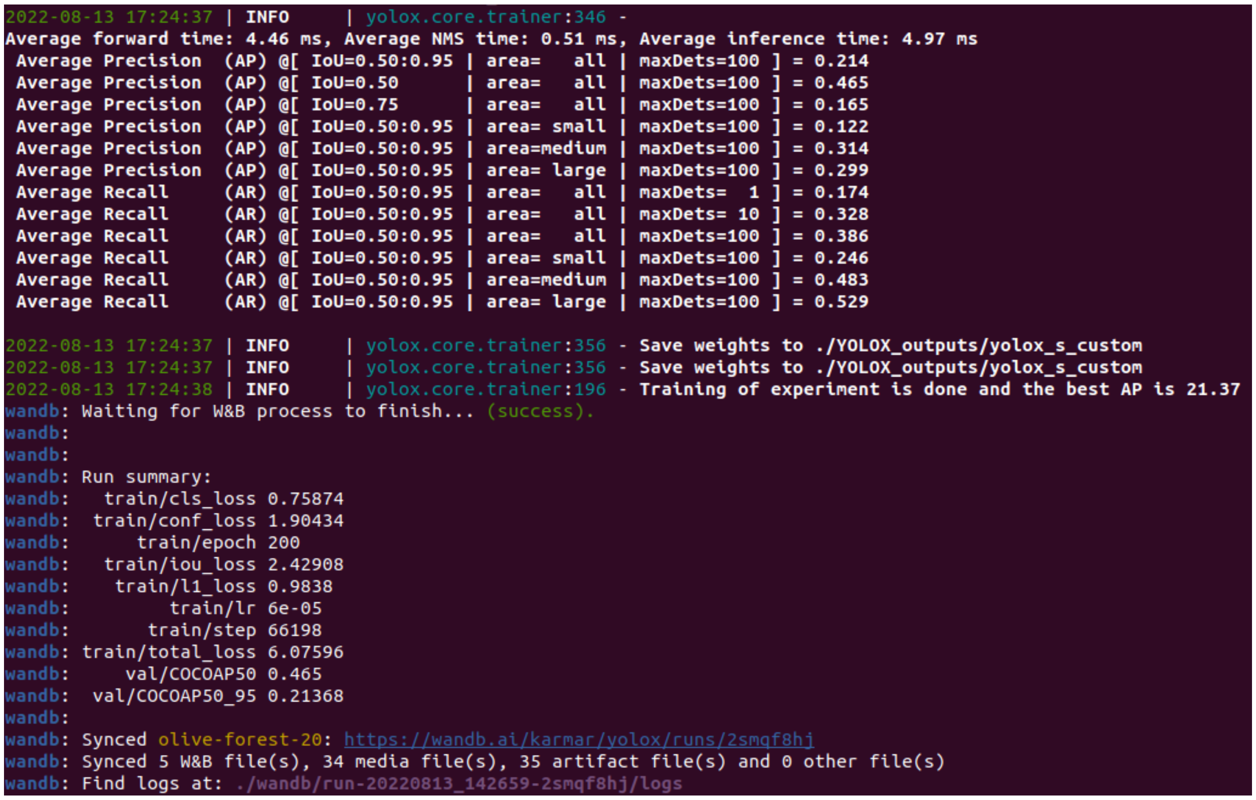
We report the AP @ [IoU=0.50:0.95], [IoU=0.50] and, [IoU=0.75]. The various training and validation loss metric can be seen in the consolidated weights and biases dashboard of our results.



```
2022-08-13 09:40:16 | INFO     | yolox.core.trainer:346 -
Average forward time: 0.86 ms, Average NMS time: 0.46 ms, Average inference time: 1.32 ms
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.084
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.222
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.038
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.023
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.135
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.164
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.089
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.174
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.227
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.077
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.322
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.417

2022-08-13 09:40:16 | INFO     | yolox.core.trainer:356 - Save weights to ./YOLOX_outputs/yolox_nano_custom
2022-08-13 09:40:16 | INFO     | yolox.core.trainer:356 - Save weights to ./YOLOX_outputs/yolox_nano_custom
2022-08-13 09:40:16 | INFO     | yolox.core.trainer:196 - Training of experiment is done and the best AP is 8.54
wandb: Waiting for W&B process to finish... (success).
wandb:
wandb:
wandb: Run summary:
wandb:   train/cls_loss 1.0448
wandb:  train/conf_loss 2.65548
wandb:      train/epoch 100
wandb:   train/iou_loss 3.0106
wandb:    train/l1_loss 1.26433
wandb:         train/lr 0.00013
wandb:       train/step 16593
wandb: train/total_loss 7.97521
wandb:     val/COCOAP50 0.22191
wandb:  val/COCOAP50_95 0.08442
wandb:
wandb: Synced dashing-gorge-15: https://wandb.ai/karmar/yolox/runs/25ao23e7
wandb: Synced 5 W&B file(s), 24 media file(s), 25 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20220813_085824-25ao23e7/logs
(cap) rahul@karma:~/workspace/coursera-sdsc-words/udacity-aws-sagemaker-capstone/YOLOX$
```

This result dramatically improves as we moved to higher batch sizes as well as from YOLOX-nano to a large model capacity like YOLOX-s and YOLOX-m. We see from the established benchmarks that as the model size increases the AP increases and the execution time also increases. We see this reflected in the AP reported below in the final evaluation of the YOLO-s model training. COCO Evaluator computes



```
2022-08-13 17:24:37 | INFO     | yolox.core.trainer:346 -
Average forward time: 4.46 ms, Average NMS time: 0.51 ms, Average inference time: 4.97 ms
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.214
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.465
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.165
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.122
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.314
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.299
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.174
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.328
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.386
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.246
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.483
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.529

2022-08-13 17:24:37 | INFO     | yolox.core.trainer:356 - Save weights to ./YOLOX_outputs/yolox_s_custom
2022-08-13 17:24:37 | INFO     | yolox.core.trainer:356 - Save weights to ./YOLOX_outputs/yolox_s_custom
2022-08-13 17:24:38 | INFO     | yolox.core.trainer:196 - Training of experiment is done and the best AP is 21.37
wandb: Waiting for W&B process to finish... (success).
wandb:
wandb:
wandb: Run summary:
wandb:   train/cls_loss 0.75874
wandb:  train/conf_loss 1.90434
wandb:      train/epoch 200
wandb:   train/iou_loss 2.42908
wandb:    train/l1_loss 0.9838
wandb:         train/lr 6e-05
wandb:       train/step 66198
wandb: train/total_loss 6.07596
wandb:     val/COCOAP50 0.465
wandb:  val/COCOAP50_95 0.21368
wandb:
wandb: Synced olive-forest-20: https://wandb.ai/karmar/yolox/runs/2smqf8hj
wandb: Synced 5 W&B file(s), 34 media file(s), 35 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20220813_142659-2smqf8hj/logs
```

Average Precision and Recall for multiple IoU which gives a scale of results over large and small detection objects in the ground truth.

The learned model weights are saved in output folder as “best_ckpt.pth”. This is used during the empirical test on unknown image and video data.

4. Results

A. Model Evaluation and Validation

We evaluate the model upon training with a validation dataset. This validation dataset is used to understand whether the model overfits or underfits the training data. Following models in the table are trained with the COCO like cyclist labeled dataset. The input image size for YOLOX-nano is (416x416) while for other models it is (640x640). This is to keep the computational complexity low for the nano version of the model.

MODEL	AP-IOU 0.50-0.95	AP-0.50	TIME (MS)	HYPER PARAMS
YOLO-NANO	0.127	0.315	1.18	Batch 32, Epo 200

YOLO-S	0.207	0.453	2.94	Batch 16, Epo 200
YOLO-M	0.204	0.438	6.20	Batch 08, Epo 200

Let us look at the wandb dashboard to understand the loss convergence and corresponding AP evaluations. The first wandb figure shows us that most of the low capacity models take more training epochs to attain a certain validation AP @ [IoU=0.50:0.95] score. At epoch 50 we can see that higher model capacity is already able to show better results in validation and later continues to perform better. However, a lower capacity model like nano plateaus around 80-100 epochs and only rises towards the end due to learning rate jumps.

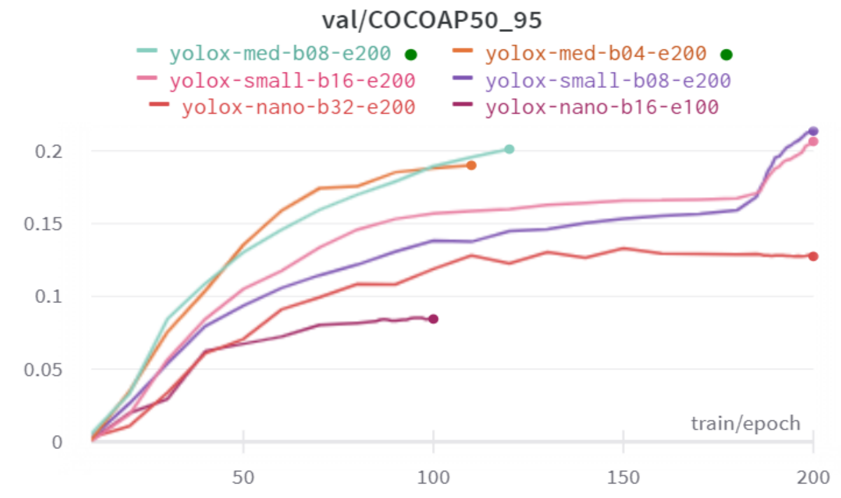


Figure: Evaluation of AP @ [IoU=0.50:0.95] for all the experiments performed. Higher batch & model capacity leads to high AP.

We see that the AP @ [IoU=0.50] follows a similar pattern but has higher score. This is due to the fact that this is a more lenient metric and must be observed carefully. A stricter metric will be where IoU>=0.75 which means we score for the objects whose overlap with the ground truth is more than 75%. Such a hard constraint is difficult to realize with the detectors at hand but it is considered when the falsified detection has a large penalty. We have logged this metric to understand the deviation from the ground truth data and .

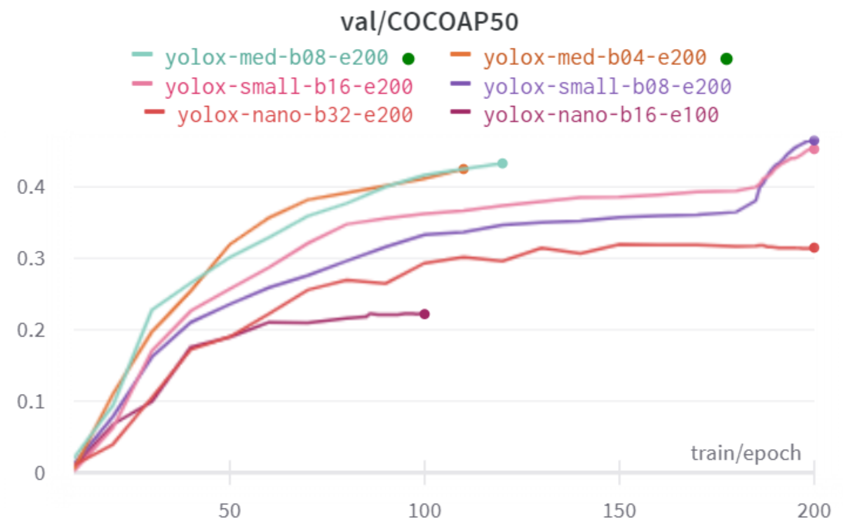


Figure: Evaluation of AP @ [IoU=0.50] for all the experiments

We can also look at the metric where we score based on the large, medium, and small size of the detected object and report the AP accordingly. This allows us to understand the model's performance based on the size of the object being detected. To understand this score we can look at the logs in the previous section. The values appear to be stable and should not show any major variation.

Now we take a look yolox-medium the training loss of the entire run. We will observe that yolox-nano appears to minimize the loss much early while small and medium takes more time to do it.

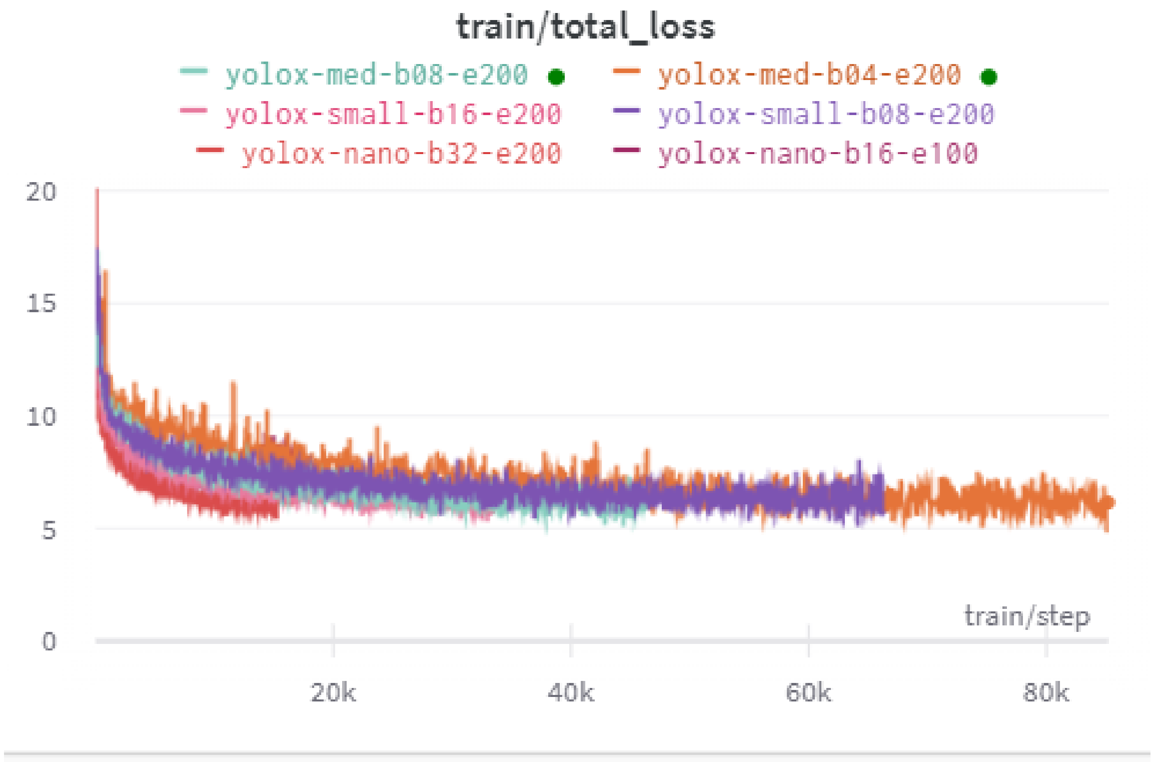


Figure: Evaluation of training loss when trained. This is fixed by using a linear combination of the individual losses

The larger models take more time to arrive at these convergence values. We should be very careful in checking the slope of these quantity. In case there is a graph that starts rising again. There we have a case of overfitting on the training data and bad AP evaluation score as well. This is the situation where we should attempt hyperparameter tuning as well as dataset update iteration. In our training both the yolo-medium models have out-performed the yolo-small models by a minor base point but has shown stability with varying batch sizes.

B. Justification

In our experiments we do fall behind the standard COCO benchmark by half. This can be due to multiple reasons. 1) an order of magnitude smaller sampled training and validation data size , 2) smaller models tested due to compute limitations, 3) limited augmentation strategies.

The final model is significant and does apply correctly to simple cases where there are few objects clearly separated and in-view. An AP >= 45% appears quite significant in the COCO benchmark as well on the visual tests. However, we do have multiple cases where it fails to do so.

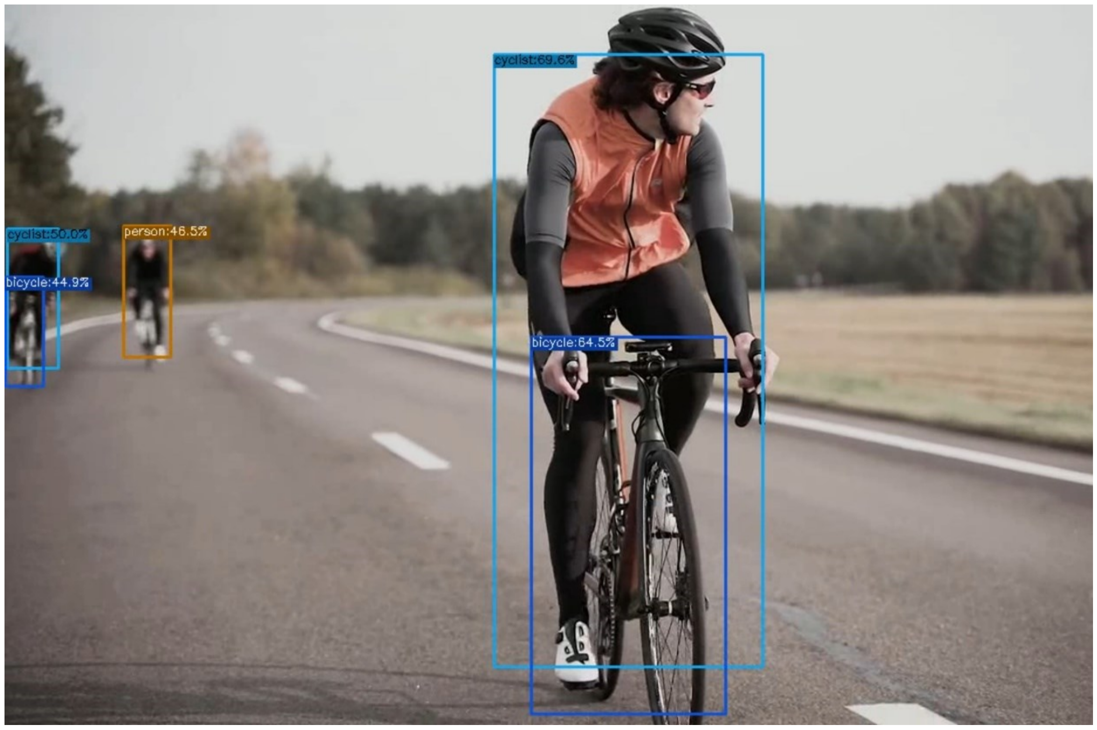


Figure: Failed case with the person behind being detected but no bicycle. However, the other two cyclists are being labelled correctly by this YOLO-nano trained detector.

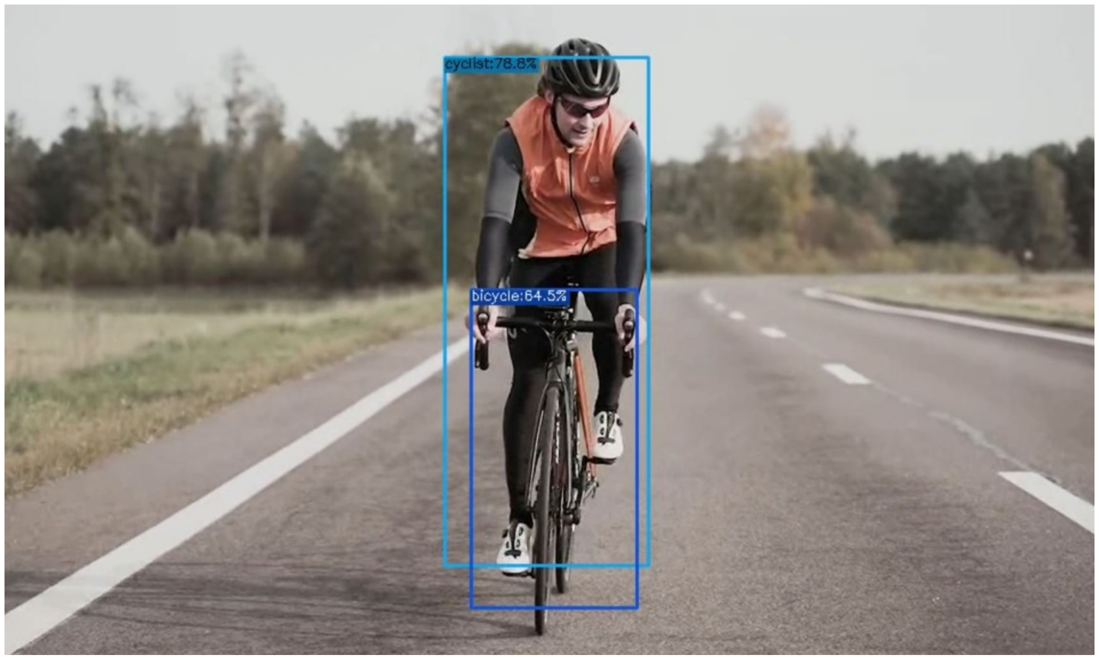


Figure: Success case with the person classified as cyclist along with bicycle. However, the image seems rather perfect for such a detection by YOLO-nano trained detector.