Stroma Machine Learning Engineer Technical Interview Reports

Batuhan Berkay AYDIN

1. Summary

While working on this technical interview, I tried many things that I couldn't. I am not working too many fine tuning operations and changing hyperparameters while training processes in my jobs. For that reason while I am working on the interview, It was a good opportunity for me.

I chose the Yolov5 object detection algorithm for bolt and nut detection. Because Yolov5 is a stable algorithm, when support is requested, easy answers can be obtained and resources can be found, and also can be easily optimized for edge devices for example all Nvidia Jetson boards and Rockchip SOC with TPU based boards (Asus Tinker Edge R, RockPi N10, Orange Pi5 etc.). After training I chose a simple Hungarian and Kalman multiple object tracker known as “SORT” to work fast on edge devices. I used json for the config file. In the config file, the user simply can change object detection, multiple object tracking parameters. For optimizing for edge devices I pruned trained models and modified yolov5 C++ TensorRT inference code. I will explain these processes in the following sections.

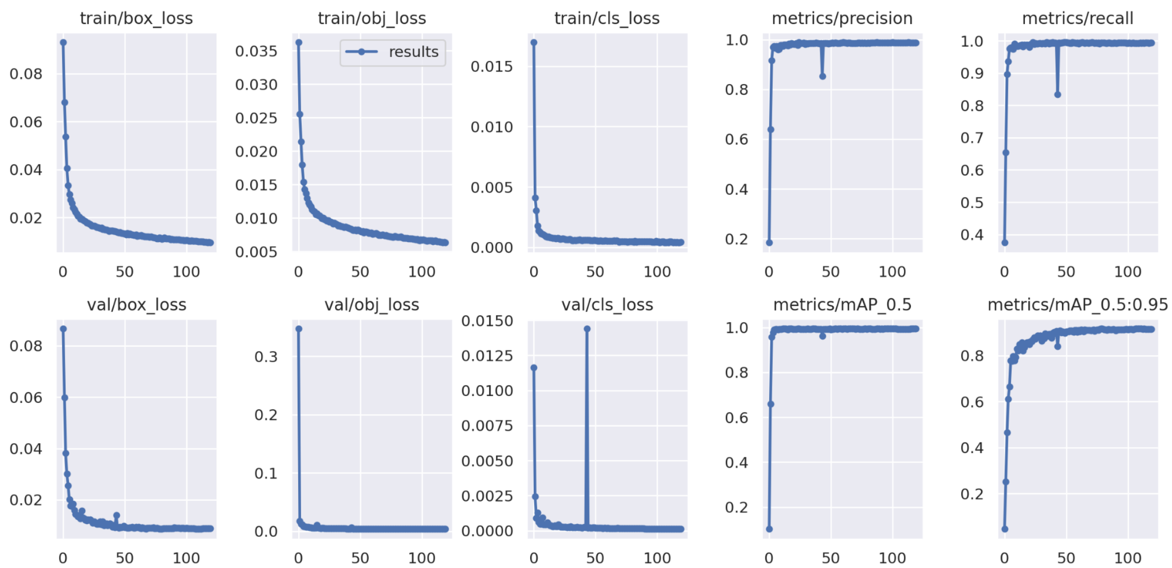
2. Training Processes

2.1. First Training

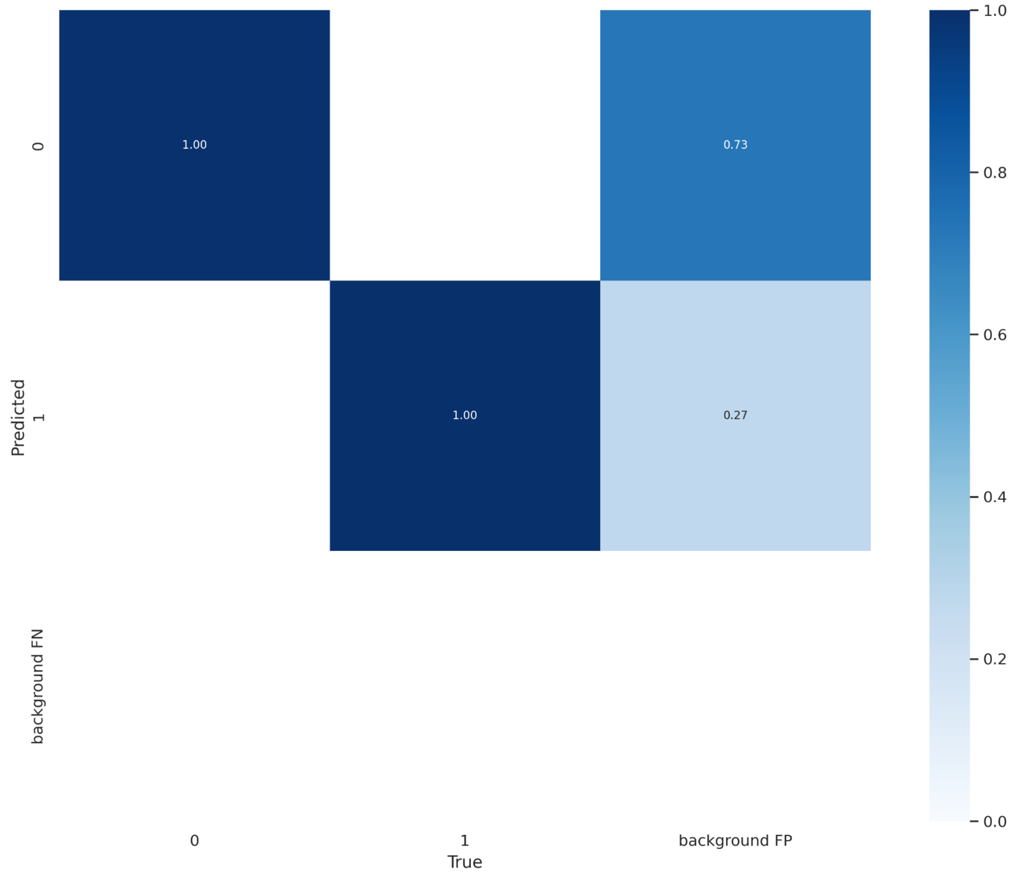
- Started given challenges dataset and used train and val folders.
- Image size is 640 x 640, batch size is 32.
- Hyperparameters, used yolov5 default (“hyp.scratch-low.yaml”):

```
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.5
cls_pw: 1.0
obj: 1.0
obj_pw: 1.0
iou_t: 0.2
anchor_t: 4.0
fl_gamma: 0.0
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
```

- Training results :
  metrics/mAP_0.5 = 0.99382
  metrics/mAP_0.5:0.95 = 0.91516



- Confusion Matrix (0: bolt, 1: nut) :



- Lesson learned from first training: Training was successful but when I tested in the test folder in the challenge folder, I got some misdetections. These misdetections are caused by objects in front of white or bright areas. For that reason I decided to change some hyperparameters and add some data which includes white or bright areas.
- Learning rate decreased 0.01 to 0.001 by changing optimizer SGD to AdamW. I used AdamW because it is a more advanced optimization algorithm compared to Adam, as it combines the benefits of weight decay regularization with the adaptive learning rates of Adam. And AdamW reduces overfitting.
- Increased final OneCycleLR learning rate 0.01 to 0.1 for fast finetuning.
- Decreased cls_loss 0.5 to 0.2 for reducing false background positives.
- Focal loss gamma increased 0.0 to 0.3 for reducing false positives.
- Image shear, mixup and scale increased for better accuracy.
- These parameters will be used for second training.

2.2. Second Training (Finetune)

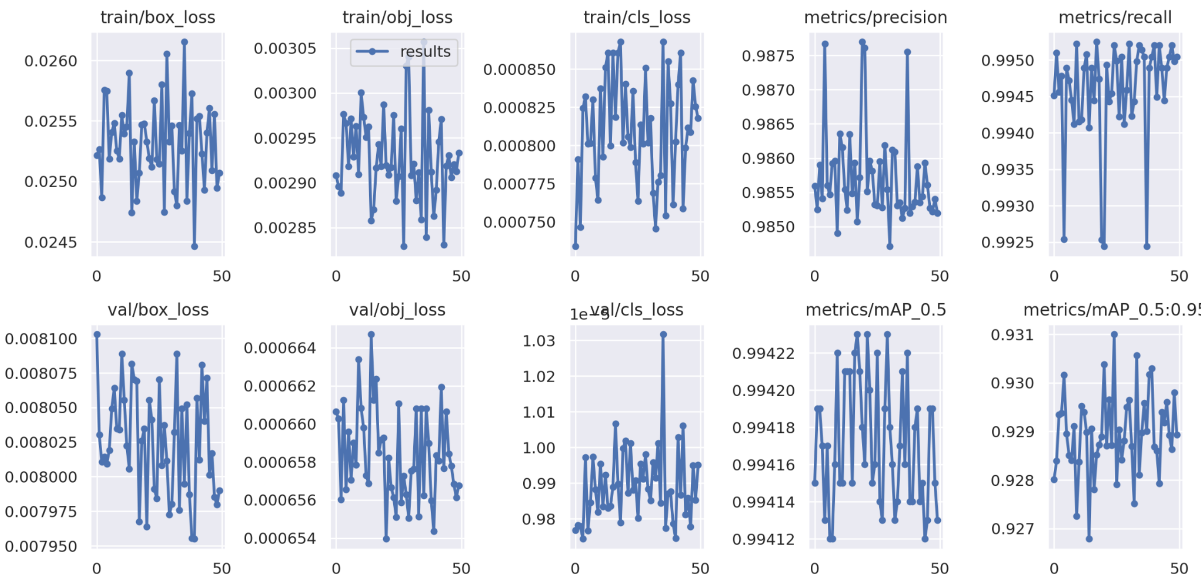
- Added new images from MVTec Screws Dataset and XMI dataset taken from roboflow.
- Image size is 640 x 640, batch size is 32.
- Hyperparameters, changed according to first training:

```
lr0: 0.001
lrf: 0.1
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.2
cls_pw: 0.8
obj: 0.6
obj_pw: 1.0
iou_t: 0.2
anchor_t: 4.0
fl_gamma: 0.3
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.5
translate: 0.1
scale: 0.9
shear: 0.5
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.1
copy_paste: 0.0
```

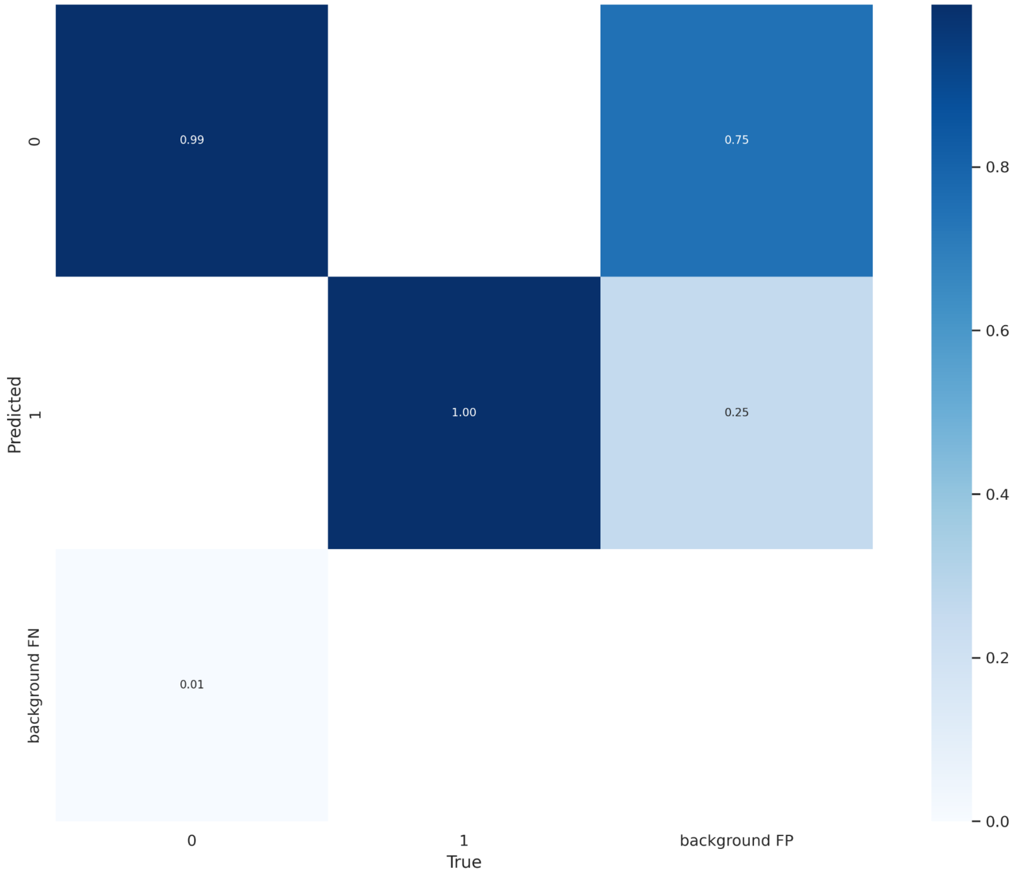
- Training results :

  metrics/mAP_0.5 = 0.99413

  metrics/mAP_0.5:0.95 = 0.92893

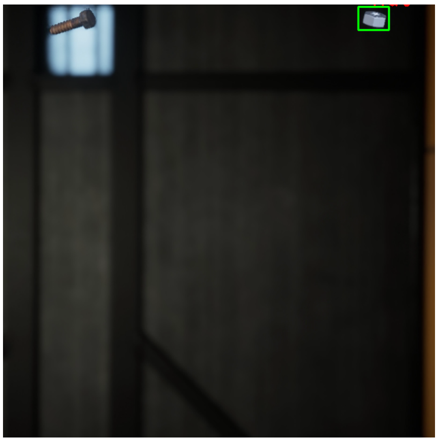
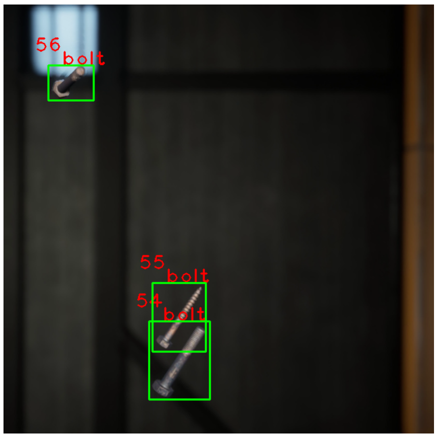


- Confusion Matrix (0: bolt, 1: nut) :



- Lesson learned from first training: Training was successful but object detection on white or bright areas could be better. For that reason I decided to finetune again with 50 epochs and set patience parameter 10 to avoid overtrain.
- Final OneCycleLR learning rate returned to default value 0.01.
- Decreased classification loss (cls_loss) 0.2 to 0.1 reducing false background positives.
- Momentum and weight decay decreased for a stable optimization process.
- Focal loss gamma increased 0.3 to 0.9 for reducing false positives.
- Hsv parameters increased for image augmentation process in case it could be better for bright areas.
- Other augmentation parameters for example degrees, scale, translate, shear increased.
- These parameters will be used for third training.

Table 1. Training example images.

<table>
<tr><th>Misdetection on bright areas example</th><th>First detection on bright areas</th></tr>
<tr><td></td><td>
</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td>Added example image for better detection in bright areas</td><td>Added example image for better detection in bright areas</td></tr>
<tr><td></td><td></td></tr>
</table>

2.3. Third Training (Finetune)

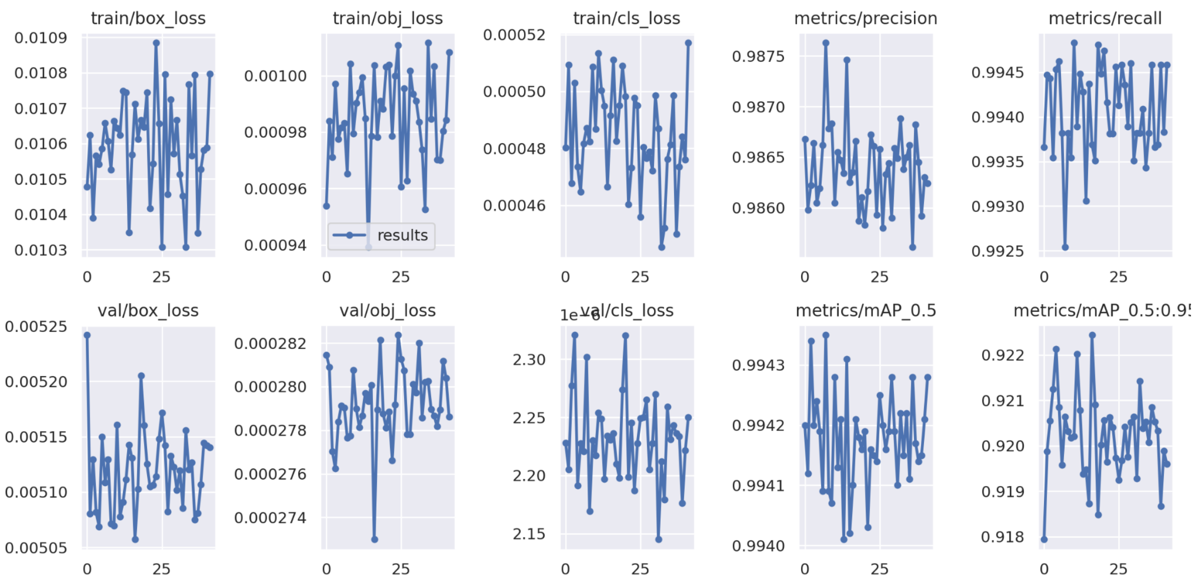
- Dataset not changed. Only hyperparameters and batch size changed.
- Image size is 640 x 640, batch size is 16.
- Hyperparameters, changed according to first training:

```
lr0: 0.001
lrf: 0.01
momentum: 0.637
weight_decay: 0.0003
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.03
cls: 0.1
cls_pw: 0.5
obj: 0.4
obj_pw: 0.8
iou_t: 0.05
anchor_t: 4.0
fl_gamma: 0.9
hsv_h: 0.05
hsv_s: 0.8
hsv_v: 0.4
degrees: 0.3
translate: 0.3
scale: 0.6
shear: 0.3
perspective: 0.0
flipud: 0.0
fliplr: 0.3
mosaic: 1.0
mixup: 0.1
copy_paste: 0.0
```

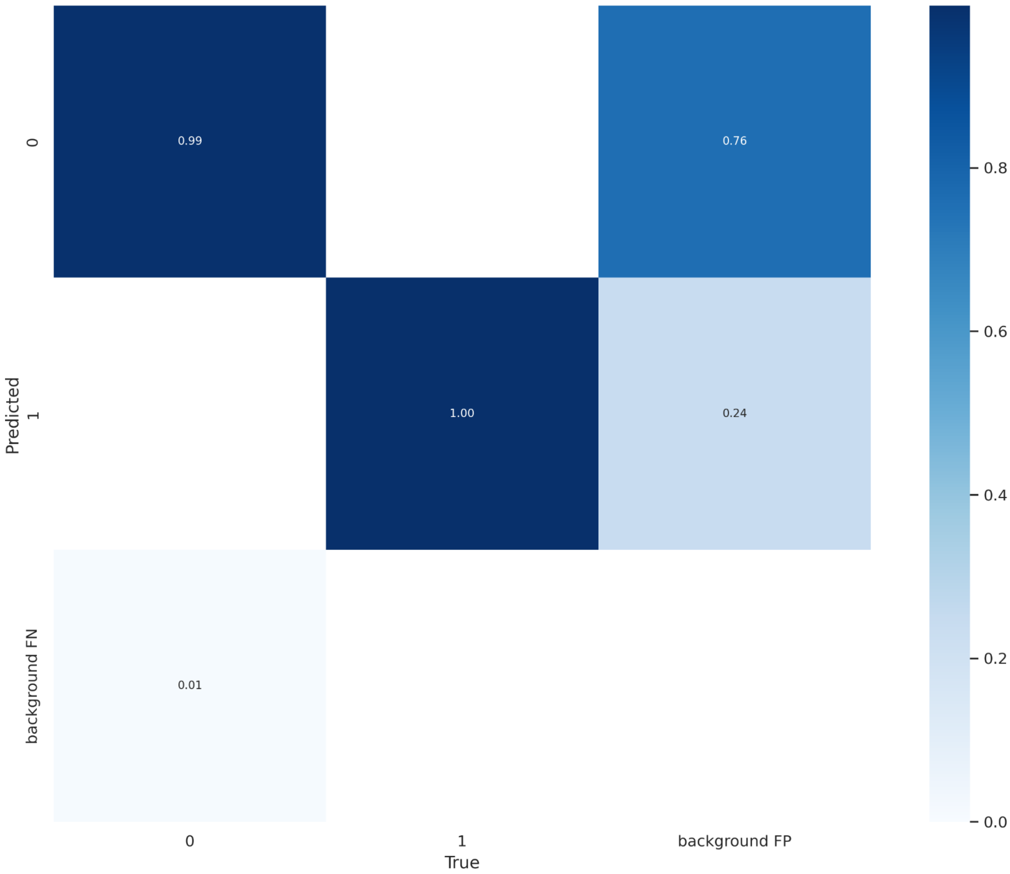
- Training results :

  metrics/mAP_0.5 = 0.9941

  metrics/mAP_0.5:0.95 = 0.92244



- Confusion Matrix (0: bolt, 1: nut) :



- Lesson learned from first training: Object detection on white or bright areas in some cases is getting better results.
- I do not have time for new training processes for that reason I stopped training. To get the best results I could have done the following steps to get the best results.

    - Using larger datasets that include objects with bright areas.
    - Use a smaller anchor box size.
    - Using larger models for example medium model could be more accurate. To run jetsons I choose yolov5 small model.

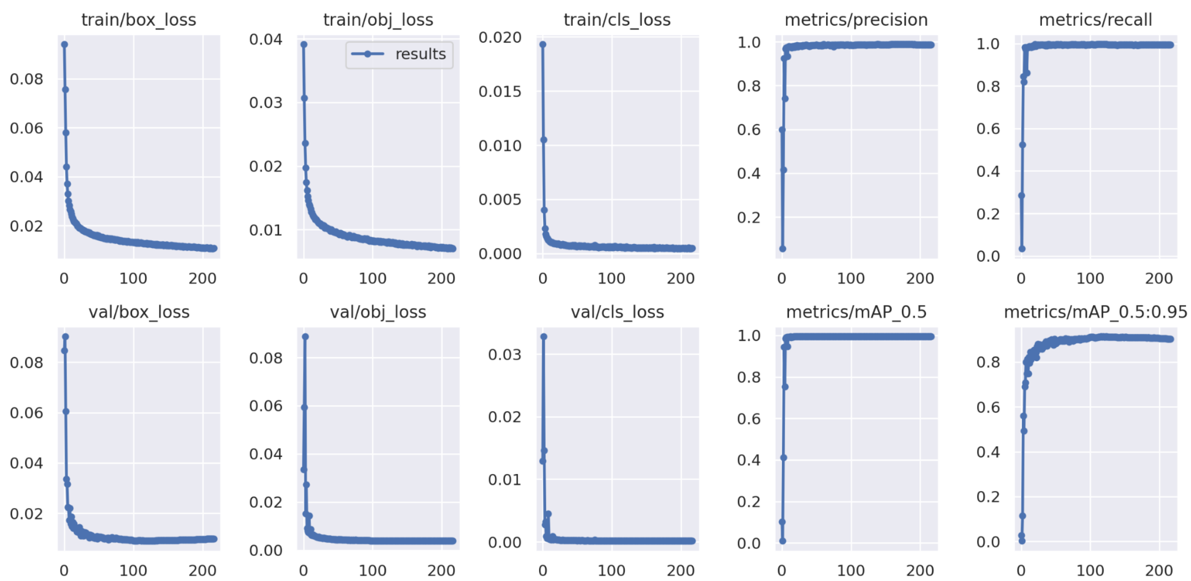
3. Yolov5 Nano Model Training (For Jetson Nano)

- Image size is 640 x 640, batch size is 32.
- Hyperparameters:

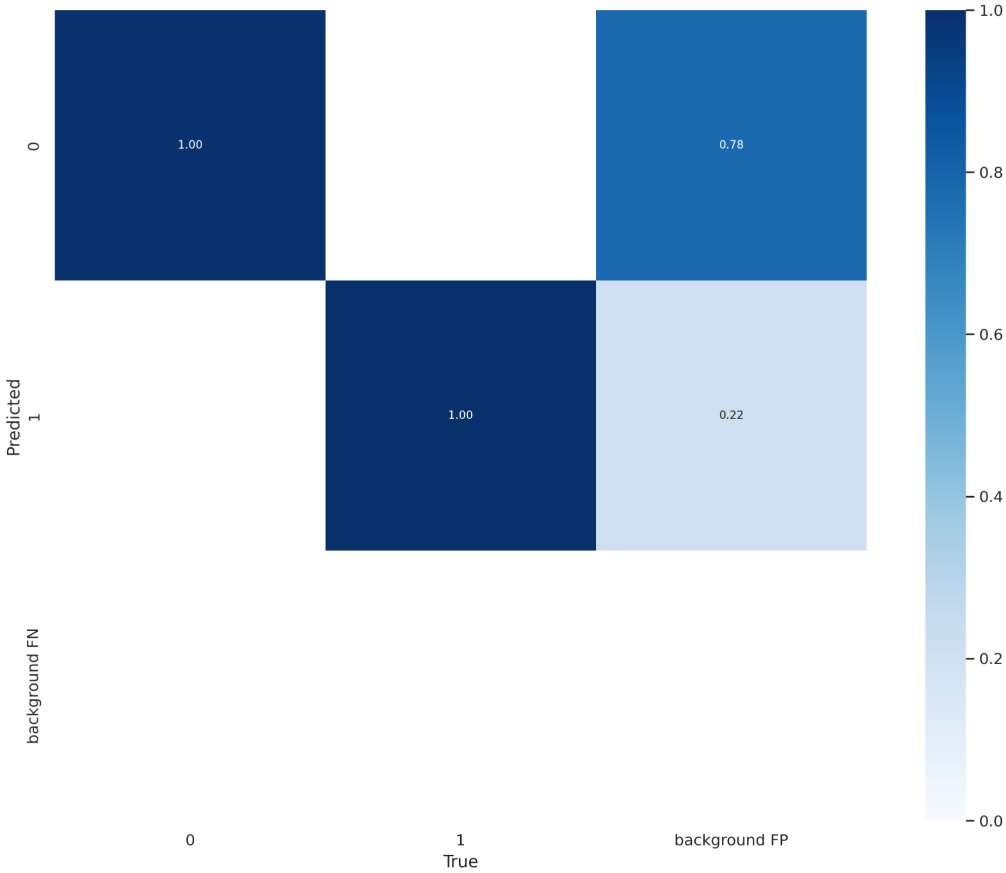
```
lr0: 0.01
lrf: 0.01
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.5
cls_pw: 1.0
obj: 1.0
obj_pw: 1.0
iou_t: 0.2
anchor_t: 4.0
fl_gamma: 0.0
```

```
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.5
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.0
copy_paste: 0.0
```

- Training results :
  metrics/mAP_0.5 = 0.99463
  metrics/mAP_0.5:0.95 = 0.91387



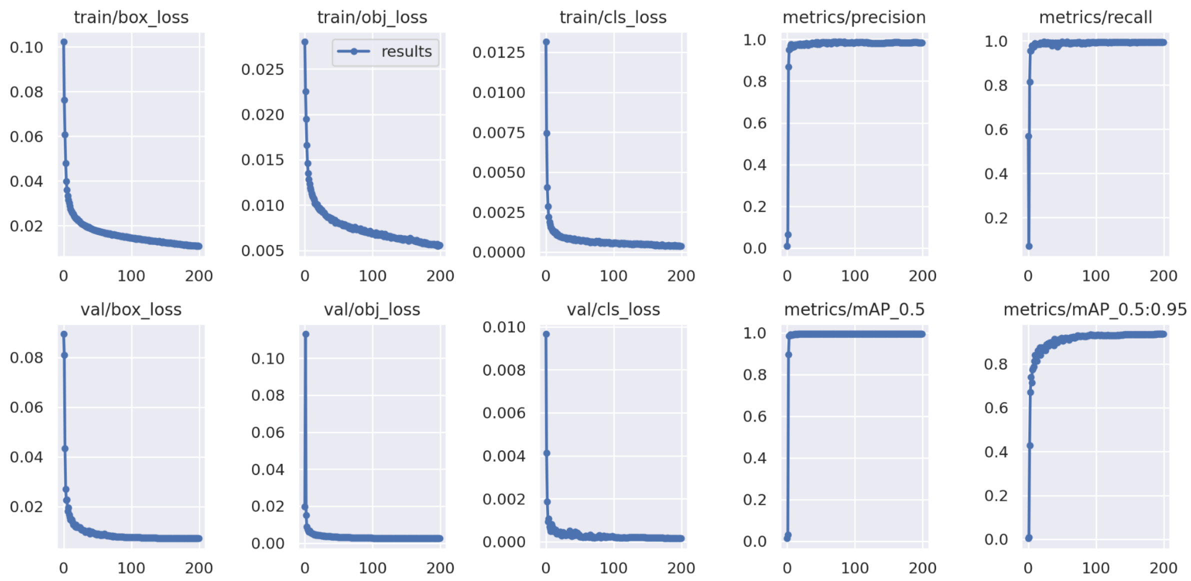
- Confusion matrix :



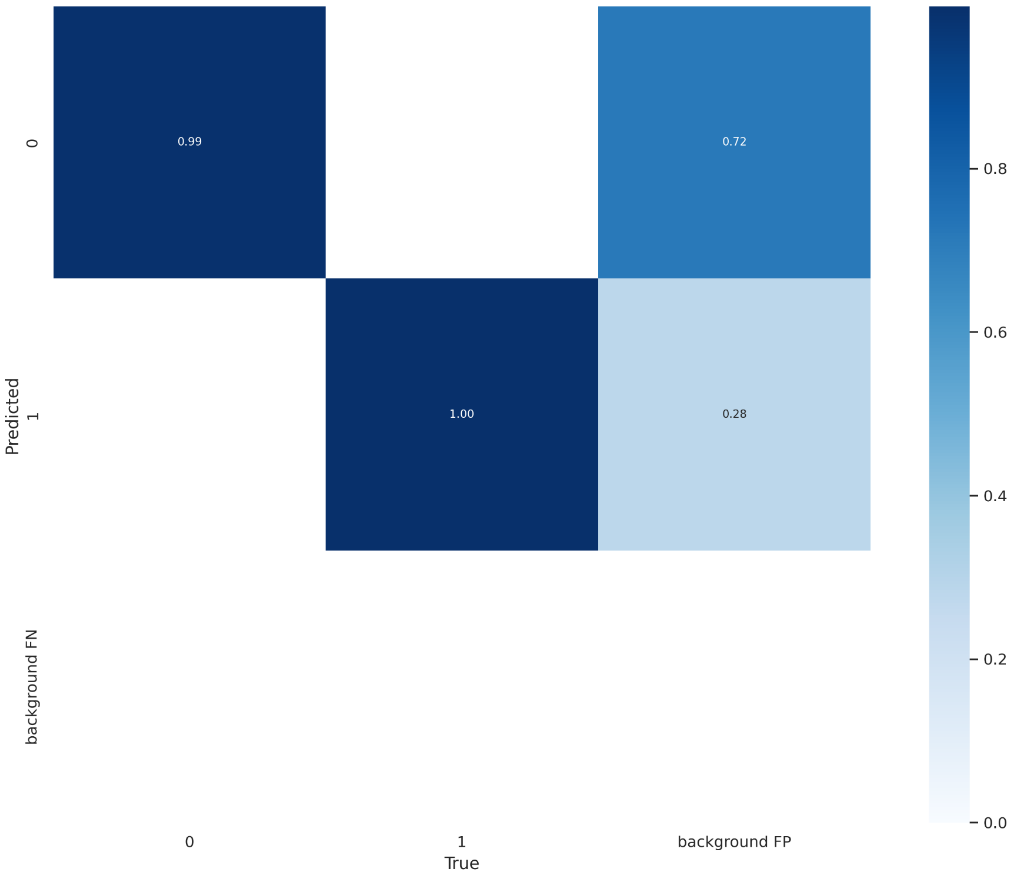
4. Yolov5 Small Model Training (For Better Results)

- Image size is 640 x 640, batch size is 32.
- Hyperparameters:

```
lr0: 0.01
lrf: 0.1
momentum: 0.937
weight_decay: 0.0005
warmup_epochs: 3.0
warmup_momentum: 0.8
warmup_bias_lr: 0.1
box: 0.05
cls: 0.3
cls_pw: 1.0
obj: 0.7
obj_pw: 1.0
iou_t: 0.2
anchor_t: 4.0
fl_gamma: 0.0
hsv_h: 0.015
hsv_s: 0.7
hsv_v: 0.4
degrees: 0.0
translate: 0.1
scale: 0.9
shear: 0.0
perspective: 0.0
flipud: 0.0
fliplr: 0.5
mosaic: 1.0
mixup: 0.1
copy_paste: 0.0
```



- Confusion matrix :



5. Yolov5 Pruning

Yolov5 supports pruning in utils folder. To pruning I added "prune(model, 0.3)" in ultralytics/yolov5/val.py folder. It is possible to add pruning function to inference code. It will be added in my repository. After pruning process the result is in below. We do not need pruning because we are using TensorRT model conversion for acceleration.

```
val: data=dataset_bolt.yaml, weights=['/home/berkay/Desktop/traffic-sign/yolov5/runs/train/exp23/weights/best.pt'], batch_size=32, imgsz=640, con
f_thres=0.001, iou_thres=0.65, task=val, device=, workers=8, single_cls=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, s
ave_conf=False, save_json=False, project=runs/val, name=exp, exist_ok=False, half=True, dnn=False
YOLOv5 🚀 v6.1-289-g526e6505 Python-3.7.15 torch-1.13.1+cu117 CUDA:0 (NVIDIA GeForce RTX 3070, 7979MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
Pruning model...  0.3 global sparsity
Model summary: 214 layers, 7015519 parameters, 0 gradients
val: Scanning '/home/berkay/Desktop/bolt-nuts-dataset/val.cache' images and labels... 1612 found, 188 missing, 0 empty, 0 corrupt: 100%|
               Class     Images     Labels          P          R      mAP@.5 mAP@.5:.95: 100%|          | 57/57 [00:15<00:00,  3.75it/s]
                 all       1800       4017      0.984      0.993      0.995      0.857
                   0       1800       3258       0.99      0.987      0.995      0.838
                   1       1800        759      0.978      0.999      0.995      0.876
Speed: 0.2ms pre-process, 1.8ms inference, 0.7ms NMS per image at shape (32, 3, 640, 640)
```

6. TensorRT build process

To get fast results I developed tensorrtx/yolov5 repository. This repository includes C++ inference generating "pt" to "wts" and "wts".

- **Generate wts file:** Run gen_wts.py in python training and inference code (**yolov5train** folder).
  $ python gen_wts.py -w model_file.pt -o converted_model.wts
- **Build TensorRT code:**
  Open terminal in yolv5trt directory.

In terminal run this commands:
$ mkdir build
$ cmake build ..
$ make -j(device_core_number+1)

- **Convert wts to TensorRT engine:**
  ./yolov5 -s [.wts] [.engine] [n/s/m/l/x/n6/s6/m6/l6/x6 c/c6 gd gw]
- **Run code:**
  ./yolov5 -d [.engine] [video_folder]
- **config.json explaining:**

```
{
  "printProcessMs": true,
  "confThresh": 0.6,
  "nmsThresh": 0.1,
  "batchSize": 1,
  "showImage":true,
  "tracker":
  {
    "maxAge": 10
  },
  "videoRecord":
  {
    "isRecordingEnable" : false,
    "isAvi": true,
    "isH264": false
  }
}
```

  - printProcessMs : Prints inference time ms.
  - confThresh : confidence threshold.
  - nmsThresh : NMS threshold.
  - batchSize : Batch Size (Use 1)
  - showImage : Shows results
  - tracker -> maxAge : Tracker max age value.
  - videoRecord -> isRecordingEnable : records video if true.
  - videoRecord -> isAvi : records video avi format.
  - videoRecord -> isH264 : records video h264 format.

7. Yolov2-FastestDet Training and NCNN Inference For Low Power Edge Devices

Build [ncnn is a high-performance neural network inference framework optimized for the mobile platform]:

Main dependency : OpenCV

```
# install dependencies
$ sudo apt-get install cmake wget
$ sudo apt-get install libprotobuf-dev protobuf-compiler libvulkan-dev
# download ncnn
$ git clone --depth=1 https://github.com/Tencent/ncnn.git
# download glslang
$ cd ncnn
$ git submodule update --depth=1 --init
# prepare folders
$ mkdir build
$ cd build
# build 64-bit ncnn for Jetson Nano
$ cmake -D CMAKE_TOOLCHAIN_FILE=../toolchains/jetson.toolchain.cmake \
    -D NCNN_DISABLE_RTTI=OFF \
    -D NCNN_BUILD_TOOLS=ON \
    -D NCNN_VULKAN=ON \
    -D CMAKE_BUILD_TYPE=Release ..
$ make -j4
$ make install
# copy output to dirs
$ sudo mkdir /usr/local/lib/ncnn
$ sudo cp -r install/include/ncnn /usr/local/include/ncnn
$ sudo cp -r install/lib/*.a /usr/local/lib/ncnn/
# once you've placed the output in your /usr/local directory,
# you can delete the ncnn directory if you wish
$ cd ~
$ sudo rm -rf ncnn
```

- Using VULCAN will get %57 performance boost which is even better MNN with CUDA for that reason NCNN is a good library for NVIDIA Jetson Nano.
- Some benchmark results are below while Jetson Nano CPU was overclocked to 2014.5 MHz and the GPU to 998.4 MHz.

Model	CPU (mSec)	Vulkan (mSec)
SqueezeNet	57.0	14.0
MobileNetV1	32.6	15.9
MobileNetV2	26.9	27.8
ResNet	128.7	45.0
GoogleNet	85.4	43.8
ShuffleNet	27.8	14.4

- NCNN Yolov2FastestDet is similar to ShuffleNet. For that reason we will get more than 30FPS.

- **Train Results:**

```
Total params: 238,642
Trainable params: 238,642
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.42
Forward/backward pass size (MB): 45459284107159.84
Params size (MB): 0.91
Estimated Total Size (MB): 45459284107162.17
----------------------------------------------------------------
computer mAP...
Evaluation model:: 100%|                                        | 17/17 [00:06<00:00,  2.49it/s]
computer PR...
Evaluation model:: 100%|                                        | 17/17 [00:06<00:00,  2.44it/s]
Precision:0.930946 Recall:0.975711 AP:0.970763 F1:0.952638
```

Precision: 0.930946
Recall: 0.975711
AP: 0.970763
F1: 0.952638
Image Size: 352x352
epochs: 600
steps: 120,240,360,480
batch_size: 96
learning_rate: 0.001

- **Second Train Results:**

```
================================================================
Total params: 238,642
Trainable params: 238,642
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.42
Forward/backward pass size (MB): 45459284107159.84
Params size (MB): 0.91
Estimated Total Size (MB): 45459284107162.17
----------------------------------------------------------------
computer mAP...
Evaluation model:: 100%|██████████████████████████████████████████████████| 202/202 [00:08<00:00, 23.00it/s]
computer PR...
Evaluation model:: 100%|██████████████████████████████████████████████████| 202/202 [00:08<00:00, 24.80it/s]
Precision:0.909978 Recall:0.980457 AP:0.976483 F1:0.943472
```

Precision: 0.909978
Recall: 0.980457
AP: 0.976483
F1: 0.943472
epochs=300
steps=70,140,210,280
batch_size=4
subdivisions=2
learning_rate=0.001


```
{
  "videoPath": "test.mp4",
  "modelBinFile": "boltnut-sim-opt.bin",
  "modelParamFile": "boltnut-sim-opt.param",
  "printProcessMs": true,
  "confThresh": 0.95,
  "iouThresh": 0.3,
  "showImage":true,
  "tracker":
  {
    "maxAge": 10
  },
  "videoRecord":
  {
    "isRecordingEnable" : false,
    "isAvi": true,
    "isH264": false
  }
}
```

    - videoPath : Video path for inference.
    - modelBinFile : model bin file path
    - modelParamFile : model param file path
    - printProcessMs : Prints inference time ms.
    - confThresh : confidence threshold.
    - iouThresh : IOU threshold.
    - showImage : Shows results
    - tracker -> maxAge : Tracker max age value.
    - videoRecord -> isRecordingEnable : records video if true.
    - videoRecord -> isAvi : records video avi format.
    - videoRecord -> isH264 : records video h264 format.

Building steps for NCNN inference code:

- Configure CMakeList.txt. Change ncnn_DIR path to your ncnn build path.
- In ncnn_cpp_infer folder, open terminal and run this commands:
    - $ mkdir build
    - $ cd build
    - $ cmake ..
    - $ make -j5
    - $ ./ncnn_demo

NOTE: NCNN and TensorRT Result videos are stored in the repository results folder.