NN for object detection project

In this project we demonstrate chess pieces detection using yolov3 model implemented in Pytorch.

We cloned Ultralytics's yolov3 model and preformed transfer learning on the last three layers of yolov3.

DATASET

The dataset we used for this project available for free on roboflow.

We added more augmentations making the dataset x3 times bigger

And changed the number of classes from 13 to 6 (detecting without color).

Total of: 1818 train images | 174 validation images | image size 614x614

- Architecture: for each jpg there is corresponding txt file with information about the label and bounding boxes of that jpg image. [label x y w h] (x,y,w,h are normalized).
- Augmentation: used Albumentation library
  1. First augmentation: gaussianNoise + MotionBlur + Rotation
  2. Second augmentation: HorizontalFlip + Verticalflip + Brightness + Rotation

  For each image augmentation we created a new txt file corresponding to the new image name (oldname + A/B + .jpg/.txt) containing the new bounding boxes.

- Making mapping files: "images_train" and "images_valid"
  (for convenience we made it all organized into folders)
  Each file is a list paths for all the images on train/valid dataset
- Makes special .yaml file - "chess_yolov3.yaml" :

```
# parent
# ├── yolov3
# └── datasets
#     └── chess_yolov3.yaml

# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list
path: ../  # dataset root dir
train: train/images/images_train.txt  # train images (relative to 'path')
val: valid/images/images_valid.txt  # val images (relative to 'path') 128 images
test:  # test images (optional)

# Classes
nc: 13  # number of classes
names: ['bishop', 'king', 'knight', 'pawn', 'queen', 'rook'] # class names
```

Training

By running 'train.py' we will train our model.

Train.py accepts argument… the one we used:

- " --data chess_yolov3 " gives the model the new metadata configurations such as number of classes and essential paths.
- "--weights yolov3.pt" initiate the model with pretrained weights. This specific weights were trained on COCO dataset with 80 different classes.
- " --epochs 50" : number of epochs.
- "--img 614" : image input shape. → updated to 640.
  `--img-size 614 must be multiple of max stride 32, updating.`
- "--freeze 24" : The tranfer-learning part.
  The yolov3 architecture of 106 layers is divided to 28 sublayers (wrap).
  1. 0-9 layers are the backbone darknet network.
  2. 10 – 27 layers are the additional yolov3 layers.
  3. The last layer is the final detection layer.

  For transfer learning we will try different freezing values:

  a. 10 – run very slow and crushes due to space allocation errors.
  b. 24 – runs faster, don't make any errors, training last detection layer.
  c. 28 – fastest, train only the the last output detection layer.

Detecting

For detection we will use a home made video of us playing chess.

The detection takes please at "detect.py" and we pass the following arguments:

1. "--weights best_weight" - the weight with the best results from all our runs.
2. "--conf 0.25" : detection box threshold, changed the number for better precision.
3. "--source input.mp4 " – input video to detect on.

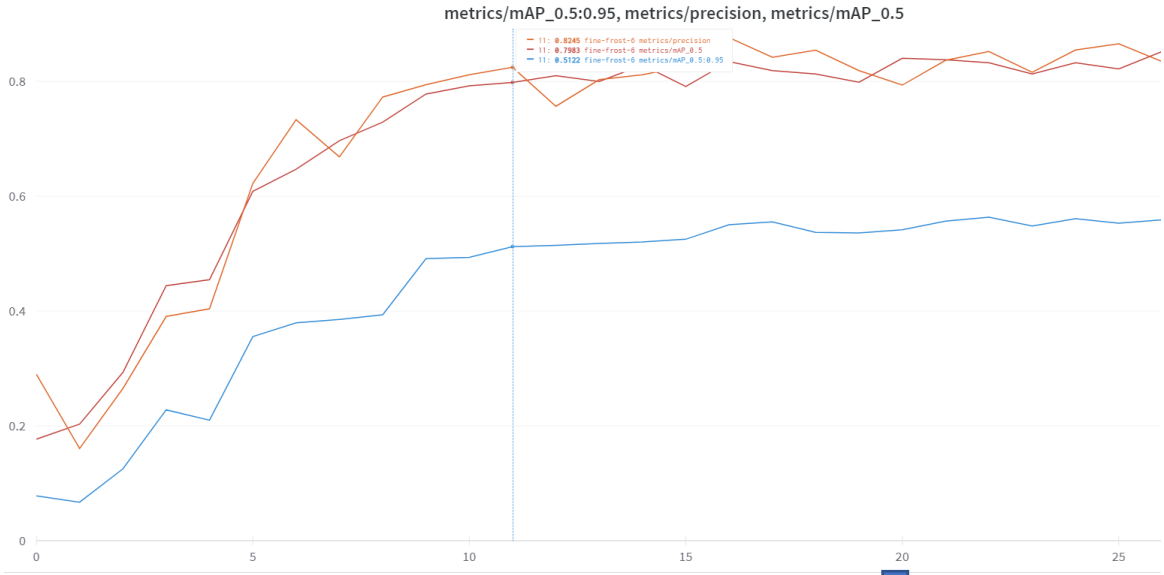
The output detections saved in yolov3/rund/detect/… in the same format given.

Results

We tried 5 different trainings:

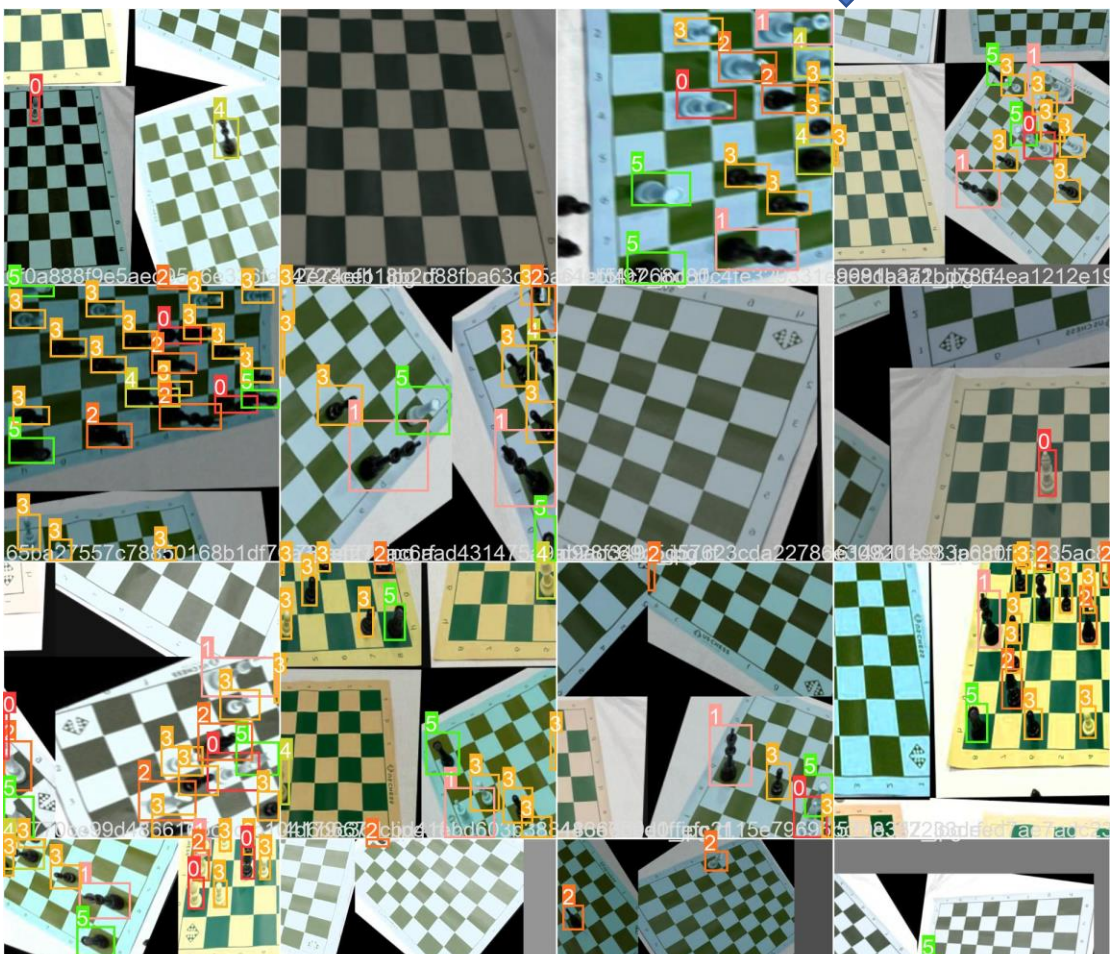
#	Epochs	Freeze	Size	Batch	classes
1	10	10	Hd images	16	13
2	50	24	614*614	16	13
3	50	28	614*614	16	13
4	50	24	614*614	16	6
5	50	24	614*614	1	13

Training info for best result: Freeze 24 | epochs 50 | image size 614 | batches 16



Precision/epoch_num

16 batch representation



Box, cls and obj LOSS