

Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

YoloV3 Implemented in Tensorflow 2.0

#tensorflow #tf2 #yolo #yolov3 #object-detection #deep-learning #machine-learning #neural-network #tensorflow-tutorials #tensorflow-examples

60 commits

3 branches

0 packages

0 releases

5 contributors

MIT

Branch: master

New pull request

Find file

Clone or download

zzh8829 Fix cv2 color space		Latest commit 78ff113 11 days ago
checkpoints	Fix epochs	9 months ago
data	Add training instruction	11 days ago
docs	Update dataset	11 days ago
tools	Fix cv2 color space	11 days ago
yolov3_tf2	Fix training for custom size	11 days ago
.gitignore	Add training instruction	11 days ago
LICENSE	Initial commit	9 months ago
README.md	Add max boxes	11 days ago
conda-cpu.yml	Add dataset visualizer	11 days ago
conda-gpu.yml	Add training instruction	11 days ago
convert.py	Add num classes and threshold flags	5 months ago
detect.py	Fix cv2 color space	11 days ago
detect_video.py	Fix cv2 color space	11 days ago
requirements.txt	Update tensorflow	3 months ago
setup.py	Add training instruction	11 days ago
train.py	Fix training for custom size	11 days ago

README.md

YoloV3 Implemented in TensorFlow 2.0

This repo provides a clean implementation of YoloV3 in TensorFlow 2.0 using all the best practices.

Key Features

- TensorFlow 2.0
- yolov3 with pre-trained Weights
- yolov3-tiny with pre-trained Weights
- Inference example
- Transfer learning example
- Eager mode training with `tf.GradientTape`
- Graph mode training with `model.fit`


```
# Tensorflow CPU
conda env create -f conda-cpu.yml
conda activate yolov3-tf2-cpu

# Tensorflow GPU
conda env create -f conda-gpu.yml
conda activate yolov3-tf2-gpu
```

Pip

```
pip install -r requirements.txt
```

Nvidia Driver (For GPU)

```
# Ubuntu 18.04
sudo apt-add-repository -r ppa:graphics-drivers/ppa
sudo apt install nvidia-driver-430
# Windows/Other
https://www.nvidia.com/Download/index.aspx
```

Convert pre-trained Darknet weights

```
# yolov3
wget https://pjreddie.com/media/files/yolov3.weights -O data/yolov3.weights
python convert.py

# yolov3-tiny
wget https://pjreddie.com/media/files/yolov3-tiny.weights -O data/yolov3-tiny.weights
python convert.py --weights ./data/yolov3-tiny.weights --output ./checkpoints/yolov3-tiny.tf --tiny
```

Detection

```
# yolov3
python detect.py --image ./data/meme.jpg

# yolov3-tiny
python detect.py --weights ./checkpoints/yolov3-tiny.tf --tiny --image ./data/street.jpg

# webcam
python detect_video.py --video 0

# video file
python detect_video.py --video path_to_file.mp4 --weights ./checkpoints/yolov3-tiny.tf --tiny

# video file with output
python detect_video.py --video path_to_file.mp4 --output ./output.avi
```

Training

I have created a complete tutorial on how to train from scratch using the VOC2012 Dataset. See the documentation here https://github.com/zzh8829/yolov3-tf2/blob/master/docs/training_voc.md

For customized training, you need to generate tfrecord following the TensorFlow Object Detection API. For example you can use [Microsoft VOTT](#) to generate such dataset. You can also use this [script](#) to create the pascal voc dataset.

Example command line arguments for training

```
python train.py --batch_size 8 --dataset ~/Data/voc2012.tfrecord --val_dataset ~/Data/voc2012_val.tfrecord --epochs
python train.py --batch_size 8 --dataset ~/Data/voc2012.tfrecord --val_dataset ~/Data/voc2012_val.tfrecord --epochs
python train.py --batch_size 8 --dataset ~/Data/voc2012.tfrecord --val_dataset ~/Data/voc2012_val.tfrecord --epochs
python train.py --batch_size 8 --dataset ~/Data/voc2012.tfrecord --val_dataset ~/Data/voc2012_val.tfrecord --epochs
```

Tensorflow Serving

You can export the model to tf serving

```
python export_tfserving.py --output serving/yolov3/1/
# verify tf-serving graph
saved_model_cli show --dir serving/yolov3/1/ --tag_set serve --signature_def serving_default
```

The inputs are preprocessed images (see `dataset.transform_images`)

outputs are

```
yolo_nms_0: bounding boxes
yolo_nms_1: scores
yolo_nms_2: classes
yolo_nms_3: numbers of valid detections
```

Benchmark (No Training Yet)

Numbers are obtained with rough calculations from `detect_video.py`

Macbook Pro 13 (2.7GHz i5)

Detection	416x416	320x320	608x608
YoloV3	1000ms	500ms	1546ms
YoloV3-Tiny	100ms	58ms	208ms

Desktop PC (GTX 970)

Detection	416x416	320x320	608x608
YoloV3	74ms	57ms	129ms
YoloV3-Tiny	18ms	15ms	28ms

AWS g3.4xlarge (Tesla M60)

Detection	416x416	320x320	608x608
YoloV3	66ms	50ms	123ms
YoloV3-Tiny	15ms	10ms	24ms

RTX 2070 (credit to @AnaRhisT94)

Detection	416x416
YoloV3 predict_on_batch	29-32ms
YoloV3 predict_on_batch + TensorRT	22-28ms

Darknet version of YoloV3 at 416x416 takes 29ms on Titan X. Considering Titan X has about double the benchmark of Tesla M60, Performance-wise this implementation is pretty comparable.

Implementation Details

Eager execution

Great addition for existing TensorFlow experts. Not very easy to use without some intermediate understanding of TensorFlow graphs. It is annoying when you accidentally use incompatible features like `tensor.shape[0]` or some sort of python control flow that works fine in eager mode, but totally breaks down when you try to compile the model to graph.

model(x) vs. model.predict(x)

When calling `model(x)` directly, we are executing the graph in eager mode. For `model.predict`, tf actually compiles the graph on the first run and then execute in graph mode. So if you are only running the model once, `model(x)` is faster since there is no compilation needed. Otherwise, `model.predict` or using exported SavedModel graph is much faster (by 2x). For non real-time usage, `model.predict_on_batch` is even faster as tested by @AnaRhisT94

GradientTape

Extremely useful for debugging purpose, you can set breakpoints anywhere. You can compile all the keras fitting functionalities with gradient tape using the `run_eagerly` argument in `model.compile`. From my limited testing, all training methods including GradientTape, `keras.fit`, eager or not yeilds similar performance. But graph mode is still preferred since it's a tiny bit more efficient.

@tf.function

@tf.function is very cool. It's like an in-between version of eager and graph. You can step through the function by disabling tf.function and then gain performance when you enable it in production. Important note, you should not pass any non-tensor parameter to @tf.function, it will cause re-compilation on every call. I am not sure whats the best way other than using globals.

absl.py (abseil)

Absolutely amazing. If you don't know already, absl.py is officially used by internal projects at Google. It standardizes application interface for Python and many other languages. After using it within Google, I was so excited to hear abseil going open source. It includes many decades of best practices learned from creating large size scalable applications. I literally have nothing bad to say about it, strongly recommend absl.py to everybody.

Loading pre-trained Darknet weights

very hard with pure functional API because the layer ordering is different in tf.keras and darknet. The clean solution here is creating sub-models in keras. Keras is not able to save nested model in h5 format properly, TF Checkpoint is recommended since its offically supported by TensorFlow.

tf.keras.layers.BatchNormalization

It doesn't work very well for transfer learning. There are many articles and github issues all over the internet. I used a simple hack to make it work nicer on transfer learning with small batches.

What is the output of transform_targets ???

I know it's very confusion but the output is tuple of shape

```
(
  [N, 13, 13, 3, 6],
  [N, 26, 26, 3, 6],
  [N, 52, 52, 3, 6]
)
```

where N is the number of labels in batch and the last dimension "6" represents `[x, y, w, h, obj, class]` of the bounding boxes.

IOU and Score Threshold

the default threshold is 0.5 for both IOU and score, you can adjust them according to your need by setting `--yolo_iou_threshold` and `--yolo_score_threshold` flags

Maximum number of boxes

By default there can be maximum 100 bounding boxes per image, if for some reason you would like to have more boxes you can use the `--yolo_max_boxes` flag.

NAN Loss / Training Failed / Doesn't Converge

Many people including me have succeeded in training, so the code definitely works @LongxingTan in <https://github.com/zzh8829/yolov3-tf2/issues/128> provided some of his insights summarized here:

1. For nan loss, try to make learning rate smaller
2. Double check the format of your input data. Data input labelled by `vott` and `labellmg` is different. so make sure the input box is the right, and check carefully the format is `x1/width,y1/height,x2/width,y2/height` and **NOT** `x1,y1,x2,y2`, or `x,y,w,h`

Make sure to visualize your custom dataset using this tool

```
python tools/visualize_dataset.py --classes=./data/voc2012.names
```

It will output one random image from your dataset with label to `output.jpg` Training definitely won't work if the rendered label doesn't look correct

Command Line Args Reference

`convert.py:`

```
--output: path to output
          (default: './checkpoints/yolov3.tf')
--[no]tiny: yolov3 or yolov3-tiny
          (default: 'false')
--weights: path to weights file
          (default: './data/yolov3.weights')
--num_classes: number of classes in the model
          (default: '80')
          (an integer)
```

`detect.py:`

```
--classes: path to classes file
          (default: './data/coco.names')
--image: path to input image
          (default: './data/girl.png')
--output: path to output image
          (default: './output.jpg')
--[no]tiny: yolov3 or yolov3-tiny
          (default: 'false')
--weights: path to weights file
          (default: './checkpoints/yolov3.tf')
--num_classes: number of classes in the model
          (default: '80')
          (an integer)
```

`detect_video.py:`

```
--classes: path to classes file
          (default: './data/coco.names')
--video: path to input video (use 0 for cam)
          (default: './data/video.mp4')
--output: path to output video (remember to set right codec for given format. e.g. XVID for .avi)
          (default: None)
--output_format: codec used in VideoWriter when saving video to file
          (default: 'XVID')
--[no]tiny: yolov3 or yolov3-tiny
          (default: 'false')
--weights: path to weights file
          (default: './checkpoints/yolov3.tf')
--num_classes: number of classes in the model
          (default: '80')
          (an integer)
```

`train.py:`

```
--batch_size: batch size
          (default: '8')
          (an integer)
--classes: path to classes file
          (default: './data/coco.names')
```

```

--dataset: path to dataset
  (default: '')
--epochs: number of epochs
  (default: '2')
  (an integer)
--learning_rate: learning rate
  (default: '0.001')
  (a number)
--mode: <fit|eager_fit|eager_tf>: fit: model.fit, eager_fit: model.fit(run_eagerly=True), eager_tf: custom GradientDescent
  (default: 'fit')
--num_classes: number of classes in the model
  (default: '80')
  (an integer)
--size: image size
  (default: '416')
  (an integer)
--[no]tiny: yolov3 or yolov3-tiny
  (default: 'false')
--transfer: <none|darknet|no_output|frozen|fine_tune>: none: Training from scratch, darknet: Transfer darknet, no_fine_tune: Transfer all and freeze darknet only
  (default: 'none')
--val_dataset: path to validation dataset
  (default: '')
--weights: path to weights file
  (default: './checkpoints/yolov3.tf')

```

Change Log

October 1, 2019

- Updated to Tensorflow to v2.0.0 Release

References

It is pretty much impossible to implement this from the yolov3 paper alone. I had to reference the official (very hard to understand) and many un-official (many minor errors) repos to piece together the complete picture.

- <https://github.com/pjreddie/darknet>
 - official yolov3 implementation
- <https://github.com/AlexeyAB>
 - explanations of parameters
- <https://github.com/qqwweee/keras-yolo3>
 - models
 - loss functions
- <https://github.com/YunYang1994/tensorflow-yolov3>
 - data transformations
 - loss functions
- <https://github.com/ayooshkathuria/pytorch-yolo-v3>
 - models
- <https://github.com/broadinstitute/keras-resnet>
 - batch normalization fix