Detectron2 Vs EfficientDet

Detectron2 and EfficientDet both are object detection model, but has some significant differences in feature extraction step, the base network and many other things taken from web are briefly listed below.

**Detectron2:**

1. It is designed to be flexible in order to support rapid implementation and evaluation, mainly designed for on-premise development
2. It is a code base , support different type of computer vision algorithm algorithm (object detection , semantic segmentation etc) with different option for their backbone architecture.
3. Backbone network are ResNet , DenseNet , FPN etc
4. Accuracy of backbone network is lower than EfficientNet
5. Slower and has large parameter
6. Only available in Pytorch
7. These models scale in only dimension either depth or width or resolution
8. design makes it easy to implement cutting-edge research projects without having to fork the entire codebase.

**EfficientDet**

1. Its flexibility doesn’t lies on rapid implementation but in FLOPS(floating point operation per second). It is designed to be used in production environment
2. Only support object detection.
3. Backbone Network are EfficientNet ( and their variation)
4. Proven accuracy of EfficientNet on 5 benchmark dataset is 3-5% higher than backbone of Detectron2
5. Less parameter and verly less FLOP compared to Detectron backbone (uses normalised fusion instead of softmax)
6. Available in tensorflow and pytorch
7. Cclaculated scaling is performed in all the three dimension
8. Use compound scaling method for architecture design hence makes the network scalable.

What is novice in EfficientDet

EfficientDet = BiFPN+ EfficientNet + compound scaling method.  
Every module is

1. **BiFPN**: A weighted *bidirectional feature network* for easy and fast multi-scale feature fusion. Weighted sum of features at different resolutions, where in Detectron all the model were using simple FPN.
2. **Compound scaling**: A new method, which jointly scales up backbone, feature network, box/class network, and resolution, in a principled way.
3. **EfficientDet**: A new family of detectors with significantly better accuracy and efficiency across a wide spectrum of resource constraints.

Note:

**1>compound scaling method**

This method uniformly scales network width, depth,and resolution with a set of fixed scaling coefficients. For example, if we want to use 2 ^ N times more computational resources, then we can simply increase the network depth by α^ N , width by β N , and image size by γ N , where α, β, γ are constant coefficients determined by a small grid search on the original small model.

Different phi values are used to get different version of EfficientNet

(<https://towardsdatascience.com/googles-efficientdet-an-overview-8d010fa15860>)

2> paper snaps

Combining EfficientNet backbones with our propose BiFPN and compound scaling, we have developed a new family of object detectors, named EfficientDet, which consistently achieve better accuracy with much fewer parameters and FLOPs than previous object detectors

3> Platform Independence by ONXX is shown in the repository linked below.

EfficientNet Repository in Pytorch is saving the final model using ONXX. ONXX model support a long list of Frameworks.

(<https://github.com/lukemelas/EfficientNet-PyTorch/blob/master/README.md#example-export>)

Training of EfficientDet :

We trained 3 different implementation of Efficientdet

1. Tensorflow : <https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet>
2. Pytorch : <https://github.com/toandaominh1997/EfficientDet.Pytorch>

Both models are very deep and are tough to train , both model has used gradient clipping and extensive regularizer. The confidence score for even any class regression box never even reached 30% for any of the class after 2-3 days of continuous training.

|  |  |
| --- | --- |
| Tensorflow | Pytorch |
| Base network : Efficientdet1 | Base Network : EfficientDet0 |
| Train\_backbone : True | Train\_backbone = False |
| Gradient Descent Used | Gradient Descent Used |
| Data first Transformed to Tf records | Directly reading from json |
| Loss function is sum of three components  box loss , cls loss and det loss | Loss function is sum of 2 components  Box loss and class loss |
| Total loss toggles from 0.3 to 0.6  never converge even after very small learning rate | Same case , loss used to toggle about 1 |
| Trained for 93200 steps with 200 steps in each epochs , loss remain 0.5 which was reached in 400th step | Trained with 7 epochs with 400 iteration each , loss always toggles all the time instead to converge |
| This repo is benchmark repo |  |
| Refer image 1 | Refer image2 |

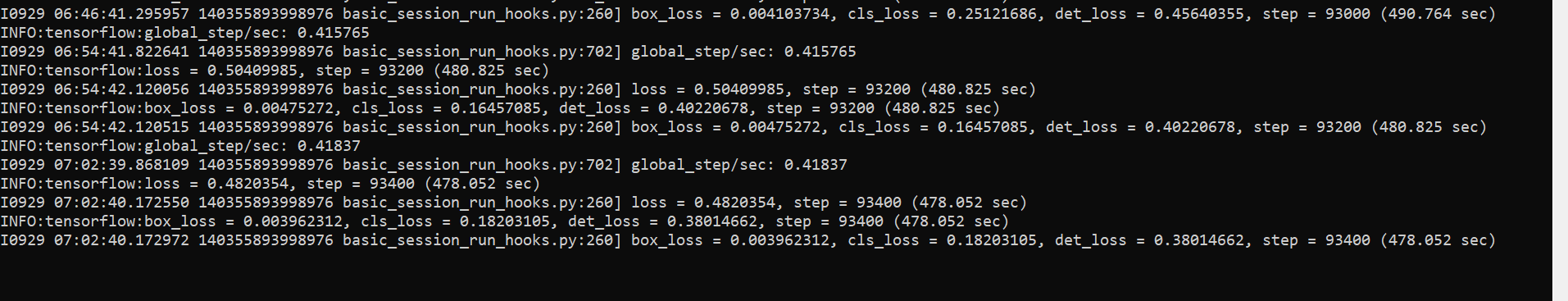


Image1

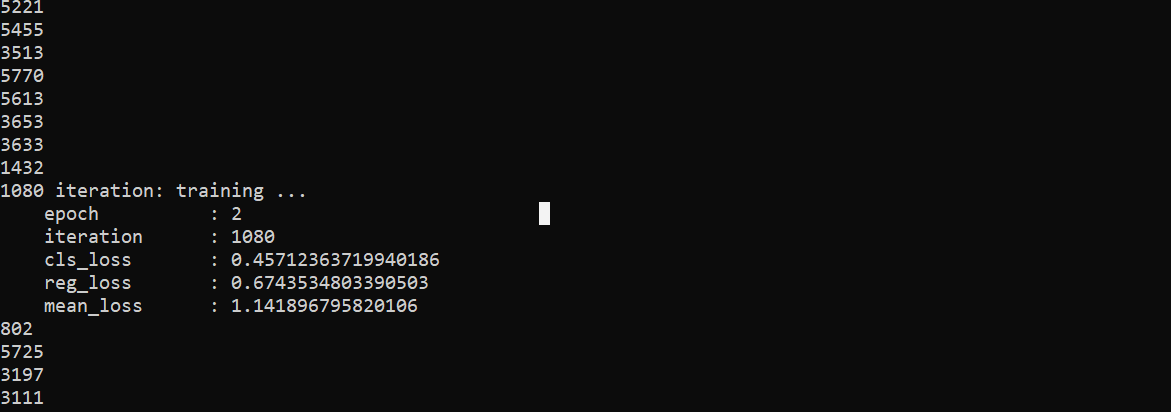


Image2

Public repositories are full of bugs, they take parse data from pycocotools and keep the class number as 80 always that too hardcoded.

There is no feature based comparision for efficientdet/net while it is available for farster rcnn and other benchmark models.

Will wait for new updated repository to come

####$$$$$$$$######