Speed/accuracy trade-offs for modern convolutional object detectors

Jonathan Huang Vivek Rathod Chen Sun Menglong Zhu Anoop Korattikara
Alireza Fathi Ian Fischer Zbigniew Wojna Yang Song Sergio Guadarrama
Kevin Murphy

Abstract

The goal of this paper is to serve as a guide for selecting a detection architecture that achieves the right speed/memory/accuracy balance for a given application and platform. To this end, we investigate various ways to trade accuracy for speed and memory usage in modern convolutional object detection systems. A number of successful systems have been proposed in recent years, but apples-to-apples comparisons are difficult due to different base feature extractors (e.g., VGG, Residual Networks), different default image resolutions, as well as different hardware and software platforms. We present a unified implementation of the Faster R-CNN [30], R-FCN [6] and SSD [25] systems, which we view as "meta-architectures" and trace out the speed/accuracy trade-off curve created by using alternative feature extractors and varying other critical parameters such as image size within each of these meta-architectures. On one extreme end of this spectrum where speed and memory are critical, we present a detector that achieves real time speeds and can be deployed on a mobile device. On the opposite end in which accuracy is critical, we present a detector that achieves state-of-the-art performance measured on the COCO detection task.

1. Introduction

A lot of progress has been made in recent years on object detection due to the use of convolutional neural networks (CNNs). Modern object detectors based on these networks — such as Faster R-CNN [30], R-FCN [6], Multibox [39], SSD [25] and YOLO [28] — are now good enough to be deployed in consumer products (e.g., Google Photos, Pinterest Visual Search) and some have been shown to be fast enough to be run on mobile devices.

However, it can be difficult for practitioners to decide what architecture is best suited to their application. Standard metrics, such as mean average precision (mAP), do not tell the entire story, since for real deployments of computer vision systems, running time and memory usage are also critical. For example, mobile devices often require a small memory footprint, and self driving cars require real time performance. Server-side production systems, like those used in Google, Facebook or Snapchat, have more leeway to optimize for accuracy, but are still subject to throughput constraints. While the methods that win competitions, such as the COCO challenge [24], are optimized for accuracy, they often rely on model ensembling and multicrop methods which are too slow for practical usage.

Unfortunately, only a small subset of papers (e.g., R-FCN [6], SSD [25] YOLO [28]) discuss running time in any detail. Furthermore, these papers typically only state that they achieve some frame-rate, but do not give a full picture of the speed/accuracy trade-off, which depends on many other factors, such as which feature extractor is used, input image sizes, etc.

In this paper, we seek to explore the speed/accuracy trade-off of modern detection systems in an exhaustive and fair way. While this has been studied for full image classification((e.g., [3]), detection models tend to be significantly more complex. We primarily investigate single-model/single-pass detectors, by which we mean models that do not use ensembling, multi-crop methods, or other "tricks" such as horizontal flipping. In other words, we only pass a single image through a single network. For simplicity (and because it is more important for users of this technology), we focus only on test-time performance and not on how long these models take to train.

Though it is impractical to compare every recently proposed detection system, we are fortunate that many of the leading state-of-the-art approaches have, at a high level, converged on a common methodology. This has allowed us to implement and compare a large number of detection systems in a unified way. In particular, we have created implementations of the Faster R-CNN, R-FCN and SSD meta-architectures, which all consist of a single convolutional network, trained with a mixed regression and classification objective, and use sliding window style predictions.

To summarize, our main contributions are as follows:

- We provide a concise survey of modern convolutional detection systems, and describe how the leading ones follow very similar designs.
- We describe our flexible and unified implementation of three meta-architectures (Faster R-CNN, R-FCN

and SSD) in Tensorflow which we use to do extensive experiments that trace the accuracy/speed trade-off curve for different detection systems, varying meta-architecture, feature extractor, image resolution, etc.

- Our findings show that using fewer proposals for Faster R-CNN can speed it up significantly without a big loss in accuracy, making it competitive with its faster cousins, SSD and RFCN. We show that SSD's performance is less sensitive to the quality of the feature extractor than Faster R-CNN and R-FCN. And we identify "sweet spots" on the accuracy/speed trade-off curve where gains in accuracy are only possible by sacrificing speed (within our family of detectors).
- Several of the meta-architecture and feature-extractor combinations that we report have never appeared before in literature. We discuss how we used some of these novel combinations to train the winning entry of the 2016 COCO object detection challenge.

2. Convolutional detection meta-architectures

Neural nets have become the leading method for high quality object detection in recent years. In this section we survey some of the highlights of this literature. The R-CNN paper by Girshick et al. [11] was among the first modern incarnations of convolutional network based detection. Inspired by recent successes on image classification [20], the R-CNN method took the straightforward approach of cropping externally computed box proposals out of an input image and running a neural net classifier on these crops. This approach can be expensive however because many crops are necessary, leading to significant duplicated computation from overlapping crops. Fast R-CNN [10] alleviated this problem by pushing the entire image once through a feature extractor then cropping from an intermediate layer so that crops share the computation load of feature extraction.

While both R-CNN and Fast R-CNN relied on an external proposal generator, recent works have shown that it is possible to generate box proposals using neural networks as well [40, 39, 8, 30]. In these works, it is typical to have a collection of boxes overlaid on the image at different spatial locations, scales and aspect ratios that act as "anchors" (sometimes called "priors" or "default boxes"). A model is then trained to make two predictions for each anchor: (1) a discrete class prediction for each anchor, and (2) a continuous prediction of an offset by which the anchor needs to be shifted to fit the groundtruth bounding box.

Papers that follow this anchors methodology then minimize a combined classification and regression loss that we now describe. For each anchor a, we first find the best matching groundtruth box b (if one exists). If such a match can be found, we call a a "positive anchor", and assign it (1) a class label $y_a \in \{1 \dots K\}$ and (2) a vector encoding of box b with respect to anchor a (called the box encoding $\phi(b_a; a)$). If no match is found, we call a a "negative anchor" and we set the class label to be $y_a = 0$. If for the anchor a we predict box encoding $f_{loc}(\mathcal{I}; a, \theta)$ and corresponding class $f_{cls}(\mathcal{I}; a, \theta)$, where $\mathcal{I}$ is the image and θ the model parameters, then the loss for a is measured as a weighted sum of a location loss and a classification loss:

$$\begin{aligned}\mathcal{L}(a, \mathcal{I}; \theta) = \alpha \cdot \mathbb{1}[a \text{ is positive}] \cdot \ell_{loc}(\phi(b_a; a) - f_{loc}(\mathcal{I}; a, \theta)) \\ + \beta \cdot \ell_{cls}(y_a, f_{cls}(\mathcal{I}; a, \theta)),\end{aligned} \tag{1}$$

where α, β are weights balancing localization and classification losses. To train the model, Equation 1 is averaged over anchors and minimized with respect to parameters θ.

The choice of anchors has significant implications both for accuracy and computation. In the (first) Multibox paper [8], these anchors (called "box priors" by the authors) were generated by clustering groundtruth boxes in the dataset. In more recent works, anchors are generated by tiling a collection of boxes at different scales and aspect ratios regularly across the image. The advantage of having a regular grid of anchors is that predictions for these boxes can be written as tiled predictors on the image with shared parameters (i.e., convolutions) and are reminiscent of traditional sliding window methods, e.g. [43]. The Faster R-CNN [30] paper and the (second) Multibox paper [39] (which called these tiled anchors "convolutional priors") were the first papers to take this new approach.

2.1. Meta-architectures

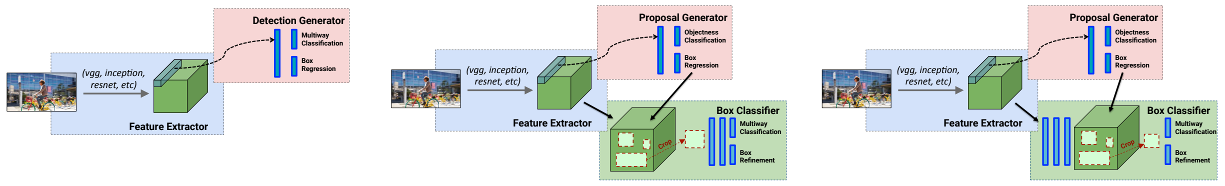
In our paper we focus primarily on three recent (meta)-architectures: SSD (Single Shot Multibox Detector [25]), Faster R-CNN [30] and R-FCN (Region-based Fully Convolutional Networks [6]). While these papers were originally presented with a particular feature extractor (e.g., VGG, Resnet, etc), we now review these three methods, decoupling the choice of meta-architecture from feature extractor so that conceptually, any feature extractor can be used with SSD, Faster R-CNN or R-FCN.

Single Shot Detector (SSD). Though the *SSD* paper was published only recently (Liu et al., [25]), we use the term SSD to refer broadly to architectures that use a single feed-forward convolutional network to directly predict classes and anchor offsets without requiring a second stage per-proposal classification operation (Figure 1(a)). Under this definition, the SSD meta-architecture has been explored in a number of precursors to [25]. Both Multibox and the Region Proposal Network (RPN) stage of Faster R-CNN [39, 30] use this approach to predict class-agnostic box proposals. [32, 28, 29, 9] use SSD-like architectures to predict final (1 of K) class labels. And Poirson et al., [27] extended this idea to predict boxes, classes and pose.

Faster R-CNN. In the Faster R-CNN setting, detection happens in two stages (Figure 1(b)). In the first stage, called the *region proposal network* (RPN), images are processed

Paper	Meta-architecture	Feature Extractor	Matching	Box Encoding $\phi(b_a, a)$	Location Loss functions
Szegedy et al. [39]	SSD	InceptionV3	Bipartite	$[x_0, y_0, x_1, y_1]$	L_2
Redmon et al. [28]	SSD	Custom (GoogLeNet inspired)	Box Center	$[x_c, y_c, \sqrt{w}, \sqrt{h}]$	L_2
Ren et al. [30]	Faster R-CNN	VGG	Argmax	$[\frac{x_c}{w_a}, \frac{y_c}{h_a}, \log w, \log h]$	SmoothL_1
He et al. [13]	Faster R-CNN	ResNet-101	Argmax	$[\frac{x_c}{w_a}, \frac{y_c}{h_a}, \log w, \log h]$	SmoothL_1
Liu et al. [25] (v1)	SSD	InceptionV3	Argmax	$[x_0, y_0, x_1, y_1]$	L_2
Liu et al. [25] (v2, v3)	SSD	VGG	Argmax	$[\frac{x_c}{w_a}, \frac{y_c}{h_a}, \log w, \log h]$	SmoothL_1
Dai et al [6]	R-FCN	ResNet-101	Argmax	$[\frac{x_c}{w_a}, \frac{y_c}{h_a}, \log w, \log h]$	SmoothL_1

Table 1. Convolutional detection models that use one of the meta-architectures described in Section 2. Boxes are encoded with respect to a matching anchor a via a function ϕ (Equation 1), where $[x_0, y_0, x_1, y_1]$ are min/max coordinates of a box, x_c, y_c are its center coordinates, and w, h its width and height. In some cases, w_a, h_a, width and height of the matching anchor are also used. **Notes**: (1) We include an early arXiv version of [25], which used a different configuration from that published at ECCV 2016; (2) [28] uses a fast feature extractor described as being inspired by GoogLeNet [38], which we do not compare to; (3) YOLO matches a groundtruth box to an anchor if its center falls inside the anchor (we refer to this as *BoxCenter*).



(a) SSD (b) Faster R-CNN (c) R-FCN

Figure 1. High level diagrams of the detection meta-architectures compared in this paper.

by a feature extractor (e.g., VGG-16), and features at some selected intermediate level (e.g., "conv5") are used to predict class-agnostic box proposals. The loss function for this first stage takes the form of Equation 1 using a grid of anchors tiled in space, scale and aspect ratio.

In the second stage, these (typically 300) box proposals are used to crop features from the same intermediate feature map which are subsequently fed to the remainder of the feature extractor (e.g., "fc6" followed by "fc7") in order to predict a class and class-specific box refinement for each proposal. The loss function for this second stage *box classifier* also takes the form of Equation 1 using the proposals generated from the RPN as anchors. Notably, one does *not* crop proposals directly from the image and re-run crops through the feature extractor, which would be duplicated computation. However there is part of the computation that must be run once per region, and thus the running time depends on the number of regions proposed by the RPN.

Since appearing in 2015, Faster R-CNN has been particularly influential, and has led to a number of follow-up works [2, 34, 33, 45, 13, 5, 19, 44, 23, 46] (including SSD and R-FCN). Notably, half of the submissions to the COCO object detection server as of November 2016 are reported to be based on the Faster R-CNN system in some way.

R-FCN. While Faster R-CNN is an order of magnitude faster than Fast R-CNN, the fact that the region-specific component must be applied several hundred times per image led Dai et al. [6] to propose the *R-FCN* (Region-based Fully Convolutional Networks) method which is like Faster R-CNN, but instead of cropping features from the same layer where region proposals are predicted, crops are taken from the last layer of features prior to prediction (Figure 1(c)). This approach of pushing cropping to the last layer minimizes the amount of per-region computation that must be done. Dai et al. argue that the object detection task needs localization representations that respect translation variance and thus propose a position-sensitive cropping mechanism that is used instead of the more standard ROI pooling operations used in [10, 30] and the differentiable crop mechanism of [5]. They show that the R-FCN model (using Resnet 101) could achieve comparable accuracy to Faster R-CNN often at faster running times. Recently, the R-FCN model was also adapted to do instance segmentation in the recent *TA-FCN model* [21], which won the 2016 COCO instance segmentation challenge.

3. Experimental platform for detection

The introduction of standard benchmarks such as Imagenet [31] and COCO [24] has made it easier in recent years to compare detection methods with respect to accuracy. However, when it comes to speed and memory, apples-to-apples comparisons have been harder to come by. Prior works have relied on different deep learning frameworks (e.g., DistBelief [7], Caffe [18], Torch [4]) and different hardware. Some papers have optimized for accuracy; others for speed. And finally, in some cases, metrics are reported using slightly different training sets (e.g., COCO training set vs. combined training+validation sets).

In order to better perform apples-to-apples comparisons, we have created a detection platform in Tensorflow [1] and have recreated training pipelines for SSD, Faster R-CNN and R-FCN meta-architectures on this platform. Having a unified framework has allowed us to easily swap feature extractor architectures, loss functions, and having it in Tensorflow allows for easy portability to diverse platforms for deployment. In the following we discuss ways to configure model architecture, loss function and input on our platform — knobs that can be used to trade speed and accuracy.

3.1. Architectural configuration

Feature extractors. In all of the meta-architectures, we first apply a convolutional *feature extractor* to the input image to obtain high-level features. The choice of feature extractor is crucial as the number of parameters and types of layers directly affect memory, speed, and performance of the detector. We have selected six representative feature extractors to compare in this paper and, with the exception of MobileNet [14], all have open source Tensorflow implementations and have had sizeable influence on the vision community.

In more detail, we consider the following six feature extractors. We use **VGG-16** [36] and **Resnet-101** [13], both of which have won many competitions such as ILSVRC and COCO 2015 (classification, detection and segmentation). We also use **Inception v2** [16], which set the state of the art in the ILSVRC 2014 classification and detection challenges, as well as its successor **Inception v3** [41]. Both of the Inception networks employ 'Inception units' which make it possible to increase the depth and width of a network without increasing its computational budget. Recently, Szegedy et al. [37] proposed **Inception Resnet (v2)**, which combines the optimization benefits conferred by residual connections with the computation efficiency of Inception units. Finally, we compare against the new **MobileNet** network [14], which has been shown to achieve VGG-16 level accuracy on Imagenet with only $1/30$ of the computational cost and model size. MobileNet is designed for efficient inference in various mobile vision applications. Its building blocks are depthwise separable convolutions which factorize a standard convolution into a depthwise convolution and a 1×1 convolution, effectively reducing both computational cost and number of parameters.

For each feature extractor, there are choices to be made in order to use it within a meta-architecture. For both Faster R-CNN and R-FCN, one must choose which layer to use for predicting region proposals. In our experiments, we use the choices laid out in the original papers when possible. For example, we use the 'conv5' layer from VGG-16 [30] and the last layer of conv_4_x layers in Resnet-101 [13]. For other feature extractors, we have made analogous choices. See supplementary materials for more details.

Liu et al. [25] showed that in the SSD setting, using multiple feature maps to make location and confidence predictions at multiple scales is critical for good performance. For VGG feature extractors, they used conv4_3, fc7 (converted to a convolution layer), as well as a sequence of added layers. In our experiments, we follow their methodology closely, always selecting the topmost convolutional feature map and a higher resolution feature map at a lower level, then adding a sequence of convolutional layers with spatial resolution decaying by a factor of 2 with each additional layer used for prediction. However unlike [25], we use batch normalization in all additional layers.

For comparison, feature extractors used in previous works are shown in Table 1. In this work, we evaluate all combinations of meta-architectures and feature extractors, most of which are novel. Notably, Inception networks have never been used in Faster R-CNN frameworks and until recently were not open sourced [35]. Inception Resnet (v2) and MobileNet have not appeared in the detection literature to date.

Number of proposals. For Faster R-CNN and R-FCN, we can also choose the number of region proposals to be sent to the box classifier at test time. Typically, this number is 300 in both settings, but an easy way to save computation is to send fewer boxes potentially at the risk of reducing recall. In our experiments, we vary this number of proposals between 10 and 300 in order to explore this trade-off.

Output stride settings for Resnet and Inception Resnet. Our implementation of Resnet-101 is slightly modified from the original to have an effective output stride of 16 instead of 32; we achieve this by modifying the conv5_1 layer to have stride 1 instead of 2 (and compensating for reduced stride by using atrous convolutions in further layers) as in [6]. For Faster R-CNN and R-FCN, in addition to the default stride of 16, we also experiment with a (more expensive) stride 8 Resnet-101 in which the conv4_1 block is additionally modified to have stride 1. Likewise, we experiment with stride 16 and stride 8 versions of the Inception Resnet network. We find that using stride 8 instead of 16 improves the mAP by a factor of 5%[1], but increased running time by a factor of 63%.

3.2. Loss function configuration

Beyond selecting a feature extractor, there are choices in configuring the loss function (Equation 1) which can impact training stability and final performance. Here we describe the choices that we have made in our experiments and Table 1 again compares how similar loss functions are configured in other works.

Matching. Determining classification and regression targets for each anchor requires matching anchors to groundtruth instances. Common approaches include greedy bipartite matching (e.g., based on Jaccard overlap) or many-to-one matching strategies in which bipartite-ness is not required, but matchings are discarded if Jaccard overlap between an anchor and groundtruth is too low. We refer to these strategies as *Bipartite* or *Argmax*, respectively. In our experiments we use *Argmax* matching throughout with thresholds set as suggested in the original paper for each

[1] i.e., (map8 - map16) / map16 = 0.05.

meta-architecture. After matching, there is typically a sampling procedure designed to bring the number of positive anchors and negative anchors to some desired ratio. In our experiments, we also fix these ratios to be those recommended by the paper for each meta-architecture.

Box encoding. To encode a groundtruth box with respect to its matching anchor, we use the box encoding function $\phi(b_a; a) = [10 \cdot \frac{x_c}{w_a}, 10 \cdot \frac{y_c}{h_a}, 5 \cdot \log w, 5 \cdot \log h]$ (also used by [11, 10, 30, 25]). Note that the scalar multipliers 10 and 5 are typically used in all of these prior works, even if not explicitly mentioned.

Location loss (ℓ_{loc}). Following [10, 30, 25], we use the Smooth L1 (or Huber [15]) loss function in all experiments.

3.3. Input size configuration.

In Faster R-CNN and R-FCN, models are trained on images scaled to M pixels on the shorter edge whereas in SSD, images are always resized to a fixed shape $M \times M$. We explore evaluating each model on downscaled images as a way to trade accuracy for speed. In particular, we have trained high and low-resolution versions of each model. In the "high-resolution" settings, we set $M = 600$, and in the "low-resolution" setting, we set $M = 300$. In both cases, this means that the SSD method processes fewer pixels on average than a Faster R-CNN or R-FCN model with all other variables held constant.

3.4. Training and hyperparameter tuning

We jointly train all models end-to-end using asynchronous gradient updates on a distributed cluster [7]. For Faster RCNN and R-FCN, we use SGD with momentum with batch sizes of 1 (due to these models being trained using different image sizes) and for SSD, we use RMSProp [42] with batch sizes of 32 (in a few exceptions we reduced the batch size for memory reasons). Finally we manually tune learning rate schedules individually for each feature extractor. For the model configurations that match works in literature ([30, 6, 13, 25]), we have reproduced or surpassed the reported mAP results.[2]

Note that for Faster R-CNN and R-FCN, this end-to-end approach is slightly different from the 4-stage training procedure that is typically used. Additionally, instead of using the ROI Pooling layer and Position-sensitive ROI Pooling layers used by [30, 6], we use Tensorflow's "crop_and_resize" operation which uses bilinear interpolation to resample part of an image onto a fixed sized grid. This is similar to the differentiable cropping mechanism of [5], the attention model of [12] as well as the Spatial Transformer Network [17]. However we disable backpropagation with respect to bounding box coordinates as we have found this to be unstable during training.

Our networks are trained on the COCO dataset, using all training images as well as a subset of validation images, holding out 8000 examples for validation.[3] Finally at test time, we post-process detections with non-max suppression using an IOU threshold of 0.6 and clip all boxes to the image window. To evaluate our final detections, we use the official COCO API [22], which measures mAP averaged over IOU thresholds in [0.5 : 0.05 : 0.95], amongst other metrics.

4. Results and discussion

In this section we analyze the data that we have collected by training and benchmarking detectors, sweeping over model configurations as described above. Each configuration includes a choice of meta-architecture, feature extractor, stride (for Resnet, Inception Resnet), resolution and number of proposals (for Faster R-CNN and R-FCN).

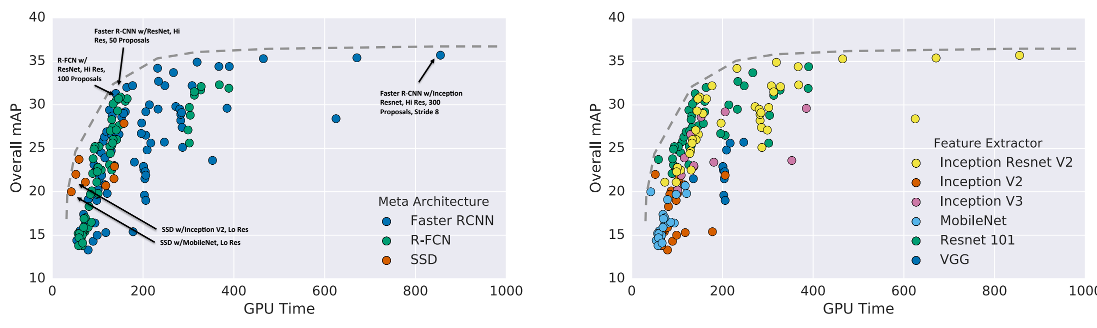
For each such model configuration, we measure timings on GPU, memory demand, number of parameters and floating point operations as described below. We make the entire table of results available in the supplementary material, noting that as of the time of this submission, we have include 147 model configurations; models for a small subset of experimental configurations (namely some of the high resolution SSD models) have yet to converge, so we have for now omitted them from analysis.

Benchmarking procedure. To time our models, we use a machine with 32GB RAM, Intel Xeon E5-1650 v2 processor and an Nvidia GeForce GTX Titan X GPU card. Timings are reported on GPU for a batch size of one. The images used for timing are COCO images resized so that the smallest size is at least k pixels in length and then cropped to $k \times k$ where k is either 300 or 600 based on the model. We average the timings over 500 images.

We include postprocessing in our timing which includes non-max suppression (NMS) and currently runs only on the CPU. NMS can take up the bulk of the running time for the fastest models at $\sim$ 40ms and currently caps our maximum framerate at 25 fps. Among other things, this means that while our timing results are comparable amongst each other, they may not be directly comparable to other reported speeds in the literature. Other potential differences include hardware, software drivers, framework (Tensorflow in our case), and batch size (e.g., the Liu et al. [25] report timings using batch sizes of 8). Finally, we use tfprof [26] to measure the total memory demand of the models during inference; this gives a more platform independent measure of memory demand. We average the memory measurements over three images.

[2]In the case of SSD with VGG, we have reproduced the number reported in the ECCV version of the paper, but the most recent version on ArXiv uses an improved data augmentation scheme to obtain somewhat higher numbers, which we have not yet experimented with.

[3]We remark that this dataset is similar but slightly smaller than the trainval35k set that has been used in several papers, e.g., [2, 25].



(a) (b)

Figure 2. mAP vs gpu wallclock time colored by (a) meta-architecture and (b) feature extractor. Each (meta-architecture, feature extractor) pair can correspond to multiple points on this plot due to changing input sizes, stride, etc.

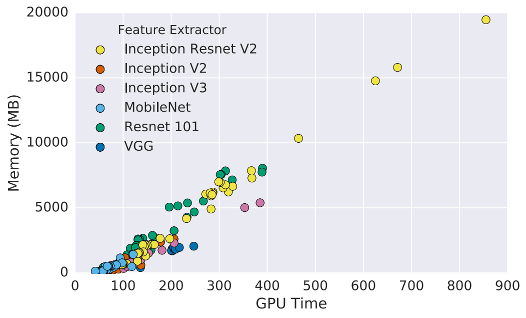


Figure 3. Memory vs GPU time for different feature extractors.

4.1. Analyses

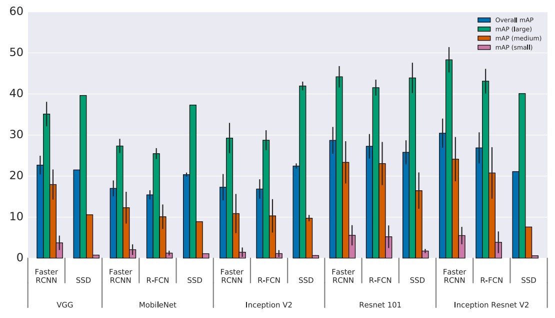
Figure 2(a) is a scatterplot visualizing the mAP of each of our model configurations colored by meta-architecture and Figure 2(b) shows the same points colored by feature extractor. Running time per image ranges from tens of milliseconds to almost 1 second. Generally we observe that R-FCN and SSD models are faster on average while Faster R-CNN tends to lead to slower but more accurate models, requiring at least 100 ms per image. However, as we discuss below, Faster R-CNN models can be just as fast if we limit the number of regions proposed. We have also overlaid an imaginary "optimality frontier" representing points at which better accuracy can only be attained within this family of detectors by sacrificing speed. Figure 3 is a plot of memory demand versus inference time with different feature extractors. Not surprisingly larger feature extractors are both slower and demand more memory. In the following, we highlight some of the key points along the optimality frontier as the best detectors to use and discuss the effect of the various model configuration options in isolation.

Critical points on the optimality frontier. *(Fastest: SSD w/MobileNet)*: On the fastest end of this optimality frontier, we see that SSD models with Inception v2 and Mobilenet are most accurate of the fastest models. Note that if we ignore postprocessing, Mobilenet seems to be roughly twice as fast as Inception v2 while being slightly worse in accuracy. *(Sweet Spot: R-FCN w/Resnet or Faster R-CNN w/Resnet and only 50 proposals)*: There is an "elbow" in the middle of the optimality frontier occupied by R-FCN models using Residual Network feature extractors which seem to strike the best balance between speed and accuracy among our model configurations. As we discuss below, Faster R-CNN w/Resnet models can attain similar speeds if we limit the number of proposals to 50. *(Most Accurate: Faster R-CNN w/Inception Resnet at stride 8)*: Finally Faster R-CNN with dense output Inception Resnet models attain the best possible accuracy on our optimality frontier, achieving (at time of submission) the state-of-the-art single model performance. However these models are slow, requiring nearly a second of processing time.

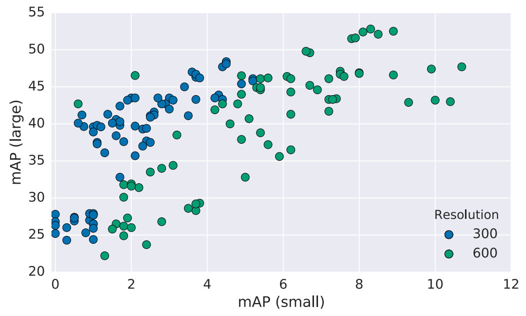
The effect of adjusting the Feature Extractor. Intuitively, stronger performance on classification should be positively correlated with stronger performance on COCO detection. To verify this, we investigate the relationship between overall mAP of different models and the Top-1 Imagenet classification accuracy attained by the pretrained feature extractor used to initialize each model. Figure 5 indicates that there is indeed an overall correlation between classification and detection performance. However this correlation appears to only be significant for Faster R-CNN and R-FCN while the performance of SSD appears to be less reliant on its feature extractor's classification accuracy.

While Figure 5 suggests that SSD is apparently unable to fully leverage the power of the ResNet and Inception ResNet feature extractors, it also suggests that using cheaper feature extractors does not hurt SSD too much. In fact when we partition performance of the same models by object size (Figure 4(a)), we see that even though SSD models typically have (very) poor performance on small objects, they are competitive with Faster RCNN and R-FCN on large objects, even outperforming these meta-architectures for the faster and more lightweight feature extractors.

The effect of adjusting image size. It has been observed by other authors that input resolution can significantly impact detection accuracy. From our experiments, we observe



(a)



(b)

Figure 4. (a) mAP for each object size by meta-architecture and feature extractor (for size 300 inputs). Interestingly, SSD shows strong performance on large objects across all feature extractors, while Faster RCNN and R-FCN both have decreased accuracy in this regime for Inception v2 and Mobilenet; (b) mAP on small objects vs mAP on large objects colored by input resolution. The two green points in the upper left hand corner are SSD models, where using high resolution inputs helps with large object performance, but can only do so much to help its already very poor performance with small objects.

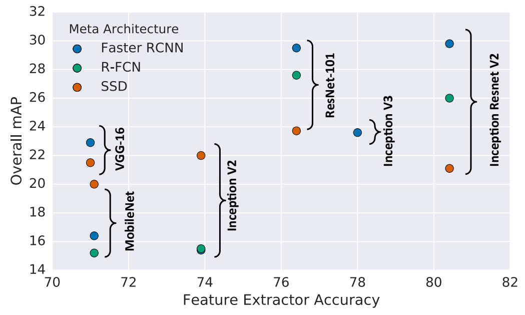
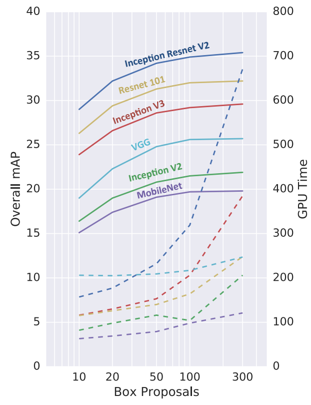
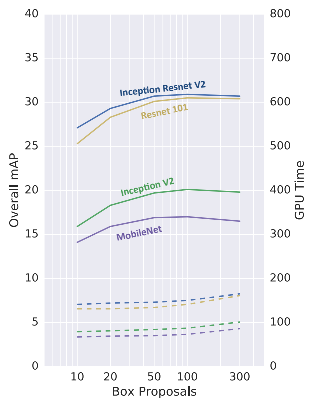


Figure 5. mAP vs. Top-1 Accuracy of the Feature Extractor on Imagenet (to avoid crowding the plot, we show only the low resolution models).



(a)



(b)

Figure 6. Effect of proposing fewer regions in (a) Faster R-CNN and (b) R-FCN on mAP (solid) and inference time (dotted). Surprisingly, for Faster R-CNN with Inception Resnet, we obtain 96% of the accuracy of using 300 proposals by using only 50 proposals, which reduces running time by a factor of 3.

that decreasing resolution by a factor of two in both dimensions consistently lowers accuracy (by 15.88% on average) but also reduces inference time by a relative factor of 27.4% on average. One reason for this effect is that high resolution inputs allow for small objects to be resolved. Figure 4(b), which compares detector performance on large objects against that on small objects, confirms that high resolution models lead to significantly better mAP results on small objects (by a factor of 2 in many cases) and somewhat better mAP results on large objects as well. We also see that strong performance on small objects implies strong performance on large objects in our models, (but not vice-versa as SSD models do well on large objects but not small).

The effect of adjusting the number of proposals. For Faster R-CNN and R-FCN, we can adjust the number of proposals computed by the region proposal network. The authors in both papers use 300 boxes, however, our experiments suggest that this number can be significantly reduced without harming mAP (by much). In some feature extractors where the "box classifier" portion of Faster R-CNN is expensive, this can lead to significant computational savings. Figure 6(a) visualizes this trade-off curve for Faster R-CNN models with high resolution inputs for different feature extractors. We see that Inception Resnet, which has 35.4% mAP with 300 proposals can still have surprisingly high accuracy (29% mAP) with only 10 proposals. The sweet spot is probably at 50 proposals, where we are able to obtain 96% of the accuracy of using 300 proposals while reducing running time by a factor of 3. While the computational savings are most pronounced for Inception Resnet, we see that similar tradeoffs hold for all feature extractors.

Figure 6(b) visualizes the same trade-off curves for R-FCN models and shows that savings from using fewer proposals in the R-FCN setting are minimal — this is not surprising as the box classifier (the expensive part) is only run once per image. We see in fact that at 100 proposals, the speed and accuracy for Faster R-CNN models with ResNet becomes comparable to that of equivalent R-FCN models which use 300 proposals in both mAP and GPU speed.

4.2. State-of-the-art detection on COCO

Finally, we briefly describe how we ensembled some of our models to achieve the current state of the art performance on the 2016 COCO object detection challenge. Our

	AP	AP@.50IOU	AP@.75IOU	AP_{small}	AP_{med}	AP_{large}	AR@100	AR_{small}	AR_{med}	AR_{large}
Ours	**0.413**	**0.62**	**0.45**	**0.231**	**0.436**	**0.547**	**0.604**	**0.424**	**0.641**	**0.748**
MSRA2015	0.371	0.588	0.398	0.173	0.415	0.525	0.489	0.267	0.552	0.679
Trimps-Soushen	0.359	0.58	0.383	0.158	0.407	0.509	0.497	0.269	0.557	0.683

Table 2. test-challenge numbers from the 2016 detection challenge. AP and AR refer to (mean) average precision and average recall respectively. Most notably, our model achieves a relative improvement of nearly 60% on small objects recall over the previous state-of-the-art COCO detector.

AP	Feature Extractor	Output stride	loss ratio	Location loss function
32.93	Resnet 101	8	3:1	SmoothL1
33.3	Resnet 101	8	1:1	SmoothL1
34.75	Inception Resnet (v2)	16	1:1	SmoothL1
35.0	Inception Resnet (v2)	16	2:1	SmoothL1
35.64	Inception Resnet (v2)	8	1:1	SmoothL1 + IOU

Table 3. Summary of single models that were automatically selected to be part of the diverse ensemble and their individual level mAP performance. Loss ratio refers to the multipliers α, β for location and classification losses, respectively.

	AP	AP@.50IOU	AP@.75IOU	AP_{small}	AP_{med}	AP_{large}
Faster RCNN with Inception Resnet (v2)	0.347	0.555	0.367	0.135	0.381	0.52
Hand selected Faster RCNN ensemble w/multicrop	0.41	0.617	0.449	0.236	0.43	0.542
Diverse Faster RCNN ensemble w/multicrop	0.416	0.619	0.454	0.239	0.435	0.549

Table 4. Effects of ensembling and multicrop inference. Numbers reported on COCO test-dev dataset. Second row (hand selected ensemble) consists of 6 Faster RCNN models with 3 Resnet 101 (v1) and 3 Inception Resnet (v2) and the third row (diverse ensemble) is described in detail in Table 4.1.

model attains 41.3% mAP@[.5, .95] on the COCO test set and is an ensemble of five Faster R-CNN models based on Resnet and Inception Resnet feature extractors and outperforms the previous best result (37.1% mAP@[.5, .95]) by MSRA which used an ensemble of three Resnet-101 models [13]. Table 4.1 summarizes the performance of our model and highlights how our model has improved on the state-of-the-art across all COCO metrics. Most notably, our model achieves a relative improvement of nearly 60% on small object recall over the previous best result. Even though this ensemble with state-of-the-art numbers could be viewed as an extreme point on the speed/accuracy tradeoff curves (requires ~50 end-to-end network evaluations per image), we have chosen to present this model in isolation since it is not comparable to the "single model" results that we focused our analysis on.

To construct our ensemble, we selected a set of five models from our collection of Faster R-CNN models. Each of the models was based on Resnet and Inception Resnet feature extractors with varying output stride configurations, retrained using variations on the loss functions, and different random orderings of the training data. Models were selected greedily using their performance on a held-out validation set. However, in order to take advantage of models with complementary strengths, we also explicitly encourage diversity by pruning away models that are too similar to previously selected models. To do this, we computed the vector of average precision results across each COCO category for each model and declared two models to be too similar if their category-wise AP vectors had cosine distance greater than some threshold.

Table 4.1 summarizes the final selected model specifications as well as their individual performance on COCO as single models.[4] Ensembling these five models using the procedure described in [13] (Appendix A) and using multicrop inference then yielded our final model. Note that we do not use multiscale training, horizontal flipping, box refinement, box voting, or global context which are sometimes used in the literature. Table 4 compares a single model's performance against two ways of ensembling, and shows that (1) encouraging for diversity did help against a hand selected ensemble, and (2) ensembling/multicrop were responsible for a $\sim$ 7 point improvement over a single model.

5. Conclusion

We have performed an experimental comparison of some of the main aspects that influence the speed and accuracy of modern object detectors. We hope this will help practitioners choose an appropriate method when deploying object detection in the real world. We have also identified some new techniques for improving speed without sacrificing much accuracy, such as using many fewer proposals than is usual for Faster R-CNN.

Acknowledgements

We would like to thank the following people for their advice and support throughout this project: Tom Duerig, Dumitru Erhan, Jitendra Malik, George Papandreou, Dominik Roblek, Chuck Rosenberg, Nathan Silberman, Abhinav Srivastava, Rahul Sukthankar, Christian Szegedy, Jasper Uijlings, Jay Yagnik, Xiangxin Zhu.

[4]Note that these numbers were computed on a held-out validation set and are not strictly comparable to the official COCO test-dev data results (though they are expected to be very close).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*, 1, 2015. 3

[2] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. *arXiv preprint arXiv:1512.04143*, 2015. 3, 5

[3] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016. 1

[4] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011. 3

[5] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. *arXiv preprint arXiv:1512.04412*, 2015. 3, 5

[6] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. *arXiv preprint arXiv:1605.06409*, 2016. 1, 2, 3, 4, 5

[7] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012. 3, 5

[8] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2147–2154, 2014. 2

[9] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017. 2

[10] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015. 2, 3, 5

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 2, 5

[12] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1462–1471, 2015. 5

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 3, 4, 5, 8

[14] A. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 4

[15] P. J. Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964. 5

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[17] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 5

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014. 3

[19] K.-H. Kim, S. Hong, B. Roh, Y. Cheon, and M. Park. Pvanet: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021*, 2016. 3

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 2

[21] Y. Li, H. Qi, J. Dai, X. Ji, and W. Yichen. Translation-aware fully convolutional instance segmentation. `https://github.com/daijifeng001/TA-FCN`, 2016. 3

[22] T. Y. Lin and P. Dollar. Ms coco api. `https://github.com/pdollar/coco`, 2016. 5

[23] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. *arXiv preprint arXiv:1612.03144*, 2016. 3

[24] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 1 May 2014. 1, 3

[25] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016. 1, 2, 3, 4, 5

[26] X. Pan. tfprof: A profiling tool for tensorflow models. `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/tools/tfprof`, 2016. 5

[27] P. Poirson, P. Ammirato, C.-Y. Fu, W. Liu, J. Kosecka, and A. C. Berg. Fast single shot detection and pose estimation. *arXiv preprint arXiv:1609.05590*, 2016. 2

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640*, 2015. 1, 2, 3

[29] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016. 2

[30] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1, 2, 3, 4, 5

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 3

[32] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization

and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 2

[33] A. Shrivastava and A. Gupta. Contextual priming and feedback for faster r-cnn. In *European Conference on Computer Vision*, pages 330–348. Springer, 2016. 3

[34] A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. *arXiv preprint arXiv:1604.03540*, 2016. 3

[35] N. Silberman and S. Guadarrama. Tf-slim: A high level library to define complex models in tensorflow. `https://research.googleblog.com/2016/08/tf-slim-high-level-library-to-define.html`, 2016. [Online; accessed 6-November-2016]. 4

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4

[37] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv preprint arXiv:1602.07261*, 2016. 4

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 3

[39] C. Szegedy, S. Reed, D. Erhan, and D. Anguelov. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014. 1, 2, 3

[40] C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013. 2

[41] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015. 4

[42] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012. 5

[43] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004. 2

[44] B. Yang, J. Yan, Z. Lei, and S. Z. Li. Craft objects from images. *arXiv preprint arXiv:1604.03239*, 2016. 3

[45] S. Zagoruyko, A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár. A multipath network for object detection. *arXiv preprint arXiv:1604.02135*, 2016. 3

[46] A. Zhai, D. Kislyuk, Y. Jing, M. Feng, E. Tzeng, J. Donahue, Y. L. Du, and T. Darrell. Visual discovery at pinterest. *arXiv preprint arXiv:1702.04680*, 2017. 3