©Copyright 2021
Joseph Redmon

Practical Computer Vision

Joseph Redmon

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Ali Farhadi, Chair

Steven Seitz

Zachary Tatlock

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Practical Computer Vision

Joseph Redmon

Chair of the Supervisory Committee:
Associate Professor Ali Farhadi
Computer Science & Engineering

The last half-decade ushered in a new era of vision research. Computer vision now works on real images, in natural environments, solving hard problems. But the technology is far from ubiquitous and many researchers are most concerned with getting the best performance on a handful of datasets. This hyperfocus on accuracy has largely turned vision into a numbers game and research tends toward complex, finely-tuned systems that are brittle and impractical in the real world.

I focus on aspects of research that are often neglected in vision: speed, scalability, usability. I design new vision systems and algorithms from the ground up with the goal of making them useful in the real world. This involves high-level algorithm improvements, mid-level architectural design, and low-level optimization and approximation. It also involves educating the next generation of vision experts and giving them the tools to solve the problems they care about.

TABLE OF CONTENTS

LIST OF FIGURES

DEDICATION

to Kelp

INTRODUCTION

Computer vision research has exploded over the last few years. Powered by convolutional neural networks and the latest gaming GPUs, the wildest imaginings of vision researchers just a decade ago are the ”Getting Started” tutorials of modern day vision frameworks. Want to tell a dog from a cat? Child’s play. My networks can tell you the specific breeds of multiple dogs in the same image, where they are, and everything else in the image to boot. Just give me 14 million labelled training examples and a computer that costs as much as a sports car.

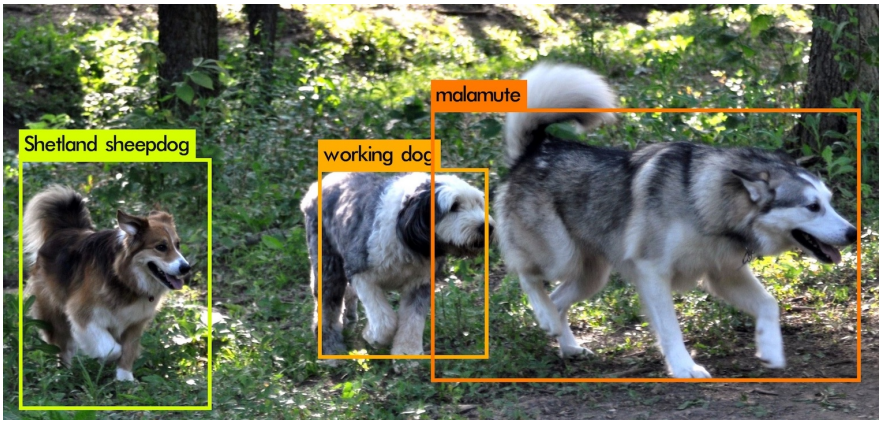


Figure 1: YOLO9000 can detect a wide variety of object classes in real-time [99].

Companies with $25 million of hardware to devote to a problem can get state-of-the-art results [112]. However, most companies and researchers want simple systems they can get working on their own problems with minimal hassle and fuss. The breakneck pace of advancement in vision research has left behind a lot of people who could benefit the most.

My research focuses on making computer vision faster, better, and more useful for people who want to use it. I build complete, robust systems for solving core problems in vision. Then I put them online, free and open-source for anyone to use.

And people do! Researchers and companies around the world use my neural network framework Darknet [96] and detection system YOLO [98] to power their own projects. Researchers apply my software to myriad domains including medicine [94], underwater robotics [65], assistive technology [89], and ecology [92], just to name a few.

This focus on the applicability of my research entails far more than just engineering challenges. I design new vision systems and algorithms from the ground up with the goal of making them useful in the real world. This thesis is the result of that work.

The remainder of the Introduction lays out the gaps between research solutions and practical needs, and where my research fills those gaps. Chapters 1-3 describe the development of YOLO, a real-time object detection system. Chapter 4 presents XNOR nets, an approximation technique to increase the speed and lower the resource consumption of arbitrary neural networks. Chapter 5 places this work in context with related trends in computer vision and concludes with possible future research directions.

0.1 The Problem

Current techniques in computer vision require enormous computing resources, are tuned for particular tasks, and are hard to adapt to new domains or datasets. Training a state-of-the-art detector requires days of time and thousands of dollars of GPU hardware. If you want to add a new category to your detector, you have to label hundreds or thousands of images and start training all over again. At test time these detectors might perform well on expensive GPUs but are prohibitively slow on CPUs, phones, or embedded devices.

Researchers in core vision algorithms primarily focus on achieving the highest accuracies on a few specific datasets. This hyper focus on accuracy leads to complex, multi-stage pipelines for processing images as each new project or paper adds another incremental improvement. Accuracies are measured on a handful of datasets: ImageNet, Pascal VOC, or COCO [21] [30] [79]. The resulting models require tens or hundreds of billions of operations to process single images and can only recognize, in the case of object detection, between 20 and 80 objects categories.

Meanwhile, other scientists who want to apply these models face an uphill struggle. If they

want to look for particular classes of object in an image—anything outside of the standard 80 COCO classes, for example—they have to manually label their own dataset and retrain the model from scratch. Simply installing the necessary software to run these models is complex. The installation instructions for Tensorflow begin with 5 paragraphs waxing poetic about the benefits and pitfalls of virtual Python environments and application containerization. Even if you get a model working it can take seconds or minutes to process single images, especially without high performance GPU hardware. This makes a large portion of research unsuitable for tasks like robotics, autonomous driving, or video processing.

My research bridges the gap between state-of-the-art computer vision and practical solutions to real problems. I strive for the following research goals:

- *Fast core vision algorithms* for real-time applications.
- *Scalable methods* to harness disparate data sources and easily extend models to new data.
- *Open technologies* that anyone can easily apply to their own data and tasks.

0.2 Speed

Robots, autonomous vehicles, and assistive devices need to understand the visual world in order to function. Object detection is the task of identifying what's in an image and where it is. Prior work relies on running an image classifier on numerous sub-windows of an image and looking for regions with strong responses from the classifier [33] [39]. When I began work on detection, state-of-the-art algorithms took more than 20 seconds to process an image, even with a $3,000 GPU [39]. Twenty seconds is prohibitively expensive, especially for tasks like autonomous driving, where every millisecond counts.

We re-frame object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, "you only look once" (YOLO) at an image to predict what objects are present and where they are. YOLO processes images in real-time with near state-of-the-art accuracy [98] [99] [101]. YOLO advances object detection

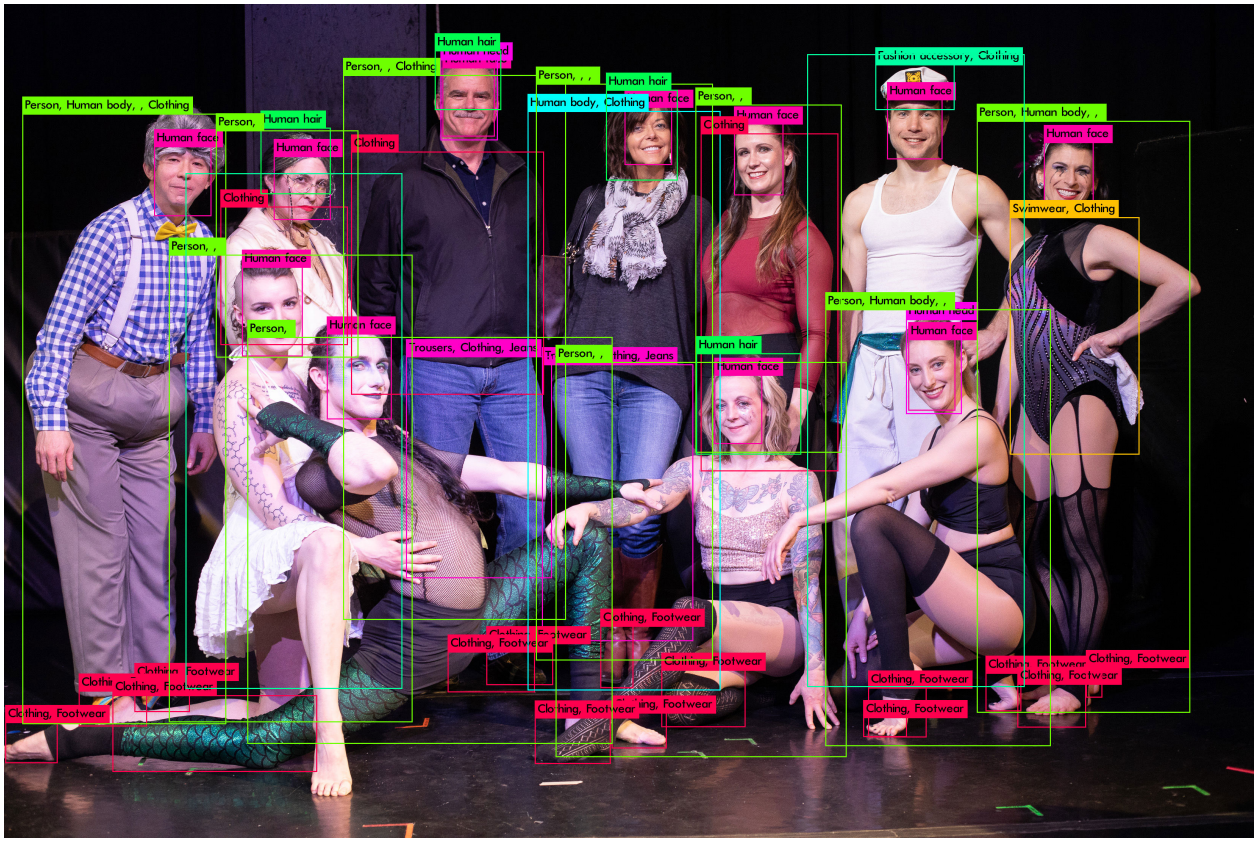


Figure 2: YOLO trained on the Open Images dataset. The model can recognize discrete objects like 'Person' but also their component parts like 'Face', 'Clothing', 'Footwear', etc.

from 20 seconds per image to 20 milliseconds per image, a thousand times faster. This speed up is due both to fundamental algorithmic advances and careful design of the underlying neural network architecture.

Algorithmically, we train a single neural network to replace the standard object detection pipeline, with numerous technical contributions to ensure high performance. YOLO introduces a novel parameterization and loss function for object detection. We jointly optimize loss terms for bounding box coordinates, objectness, and conditional class probabilities while training on full images. Improvements like multi-scale training and dimensional cluster priors decrease overfitting and make our objective easier to learn, improving model performance. Architecturally, we design

new network architectures specifically crafted for low latency and high performance. Our final network structure, Darknet-53, has comparable classification performance to ResNet-152 but is twice as fast [101].

YOLO, and most state-of-the-art techniques in vision, rely on convolutional neural networks to build a statistical model of visual data. While fast in massively parallel settings, these networks can be prohibitively expensive on CPUs or embedded devices. Researchers at the Allen Institute for Artificial Intelligence and I show that instead of using full precision floating point operations we can approximate the weights and values in a network as single bits. Our approximation reduces the problem of matrix multiplication to a series of XNOR and bit-counting operations [95].

These XNOR-nets use significantly less processing power and memory than their full-precision counterparts. This makes them much faster on lower cost hardware like CPUs, mobile phones, and embedded devices. Using XNOR-nets we can put advanced vision algorithms onto cheap hardware and still maintain good performance.

Scalability

Core computer vision research generally evaluate on a few standard datasets. Typically, detection algorithms are evaluated on Pascal VOC or COCO which have 20/80 classes and about 20,000/100,000 images respectively. These benchmarks have been critical in advancing the state of computer vision because they provide well-labelled training data and consistent standards for evaluation. However, researchers tend to focus their efforts on these particular datasets without considering the generalizability of their approaches.

For example, a detector that takes linear time based on the number of object categories will be fine on datasets with 20 or 80 categories but won't scale well to detecting thousands of different categories. Furthermore, researchers who want to use these detection systems find that the pre-trained models work well on the specific classes in the dataset but getting the detector to find new object categories requires labelling new data and retraining the network. We really need a fast, powerful detector that can detect thousands of different object classes, out-of-the-box, without the need for retraining.

The most common detection datasets contain thousands to hundreds of thousands of images with dozens to hundreds of tags [30] [79] [21]. Classification datasets have millions of images with tens or hundreds of thousands of categories [122] [21].

We would like detection to scale to level of object classification. However, labelling images for detection is far more expensive than labelling for classification or tagging (tags are often user-supplied for free). Thus we are unlikely to see detection datasets on the same scale as classification datasets in the near future.

Our work on YOLO9000 proposes a new method to harness the large amount of classification data we already have and use it to expand the scope of current detection systems. Our method uses a hierarchical view of object classification that allows us to combine distinct datasets together. We also propose a joint training algorithm that allows us to train object detectors on both detection and classification data. Our method leverages labeled detection images to learn to precisely localize objects while it uses classification images to increase its vocabulary and robustness.

Using this method we train YOLO9000, a real-time object detector that can detect over 9000 different object categories. We use our dataset combination method and joint training algorithm to train on more than 9000 classes from ImageNet as well as detection data from COCO.

While it's just a first step into this domain, YOLO9000 provides a real-time detector that can detect a wide variety of categories in different visual domains. Researchers in other fields can use YOLO9000 out-of-the-box for many common tasks. Furthermore, it lowers the burden of labelling data for new categories because a performant detector can be trained with mostly classification data.

0.3 Openness

My research has impact both because it's useful and because it's available. All of my research code is open source and free for any use on GitHub: `https://github.com/pjreddie/darknet`. I try to make my research easy for others to use and apply to their own problems. Other neural network frameworks can take hours to install, with Darknet it only takes 5 shell commands to run a detector from scratch:

```
git clone https://github.com/pjreddie/darknet
cd darknet
make
wget https://pjreddie.com/media/files/yolo.weights
./darknet detect cfg/yolo.cfg yolo.weights data/dog.jpg
```

Darknet has no external libraries or dependencies, making it easy to install on a variety of devices and systems. These details are not simply an added bonus on top of my research—they are part of a broader approach to research in general.

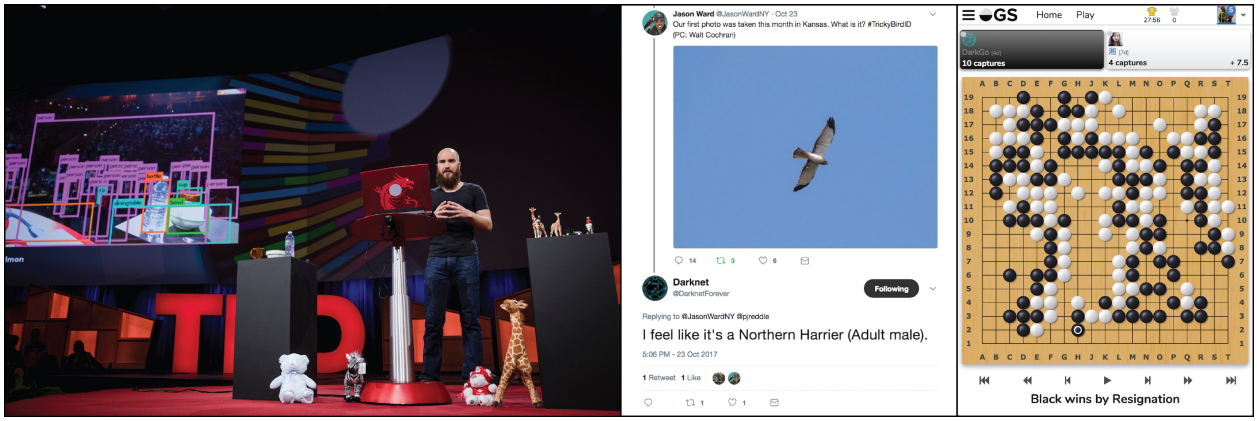


Figure 3: I'm always excited to reach out to the greater scientific community and share my research, be it through TED talks, Twitter, or online board games. The Darknet Twitter API competes regularly in the bird recognition challenges #CrowOrNo and #TrickyBirdID (and often wins!) while DarkGo has played over 45,000 games of Go with humans online at a 5 dan level.

As researchers, it's not enough to simply state that something is possible, or even show that it's possible. Our job is to enable others to do things that used to be impossible.

Along those lines, I try to connect with fellow scientists and the greater public in a variety of ways. I was fortunate to be invited to give a TED talk about my research last year in Vancouver and share with people around the world how far we've come in computer vision. My neural network framework has a Twitter API that runs classifiers or detectors on images that other users tweet at

it. Darknet even comes with it's own Go playing engine, DarkGo, which has played more than 45,000 games on the KGS and OGS game servers.

This openness and outreach has created a community of users around my research that magnify its effect. They help each other install the software, train new models, and debug issues even if I'm unavailable. All of this means that more people have access to the tools they need to solve the problems they care about.

Chapter 1

YOU ONLY LOOK ONCE: UNIFIED, REAL-TIME OBJECT DETECTION

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

1.1 Introduction

Humans glance at an image and instantly know what objects are in the image, where they are, and how they interact. The human visual system is fast and accurate, allowing us to perform complex tasks like driving with little conscious thought. Fast, accurate algorithms for object detection would allow computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information to human users, and unlock the potential for general purpose, responsive robotic systems.

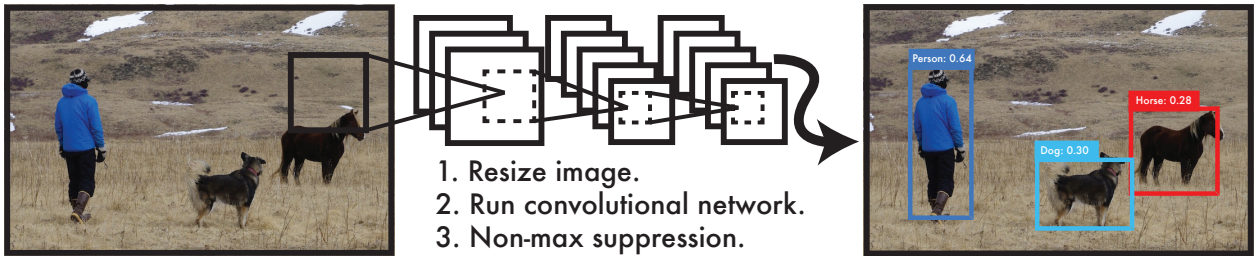


Figure 1.1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448×448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

Current detection systems repurpose classifiers to perform detection. To detect an object, these systems take a classifier for that object and evaluate it at various locations and scales in a test image. Systems like deformable parts models (DPM) use a sliding window approach where the classifier is run at evenly spaced locations over the entire image [32].

More recent approaches like R-CNN use region proposal methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [40]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only look once (YOLO) at an image to predict what objects are present and where they are.

YOLO is refreshingly simple: see Figure 1.1. A single convolutional network simultaneously predicts multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images and directly optimizes detection performance. This unified model has several benefits over traditional methods of object detection.

First, YOLO is extremely fast. Since we frame detection as a regression problem we don't need a complex pipeline. We simply run our neural network on a new image at test time to predict

detections. Our base network runs at 45 frames per second with no batch processing on a Titan X GPU and a fast version runs at more than 150 fps. This means we can process streaming video in real-time with less than 25 milliseconds of latency. Furthermore, YOLO achieves more than twice the mean average precision of other real-time systems. For a demo of our system running in real-time on a webcam please see our project webpage: http://pjreddie.com/yolo/.

Second, YOLO reasons globally about the image when making predictions. Unlike sliding window and region proposal-based techniques, YOLO sees the entire image during training and test time so it implicitly encodes contextual information about classes as well as their appearance. Fast R-CNN, a top detection method [41], mistakes background patches in an image for objects because it can't see the larger context. YOLO makes less than half the number of background errors compared to Fast R-CNN.

Third, YOLO learns generalizable representations of objects. When trained on natural images and tested on artwork, YOLO outperforms top detection methods like DPM and R-CNN by a wide margin. Since YOLO is highly generalizable it is less likely to break down when applied to new domains or unexpected inputs.

YOLO still lags behind state-of-the-art detection systems in accuracy. While it can quickly identify objects in images it struggles to precisely localize some objects, especially small ones. We examine these tradeoffs further in our experiments.

All of our training and testing code is open source. A variety of pretrained models are also available to download.

1.2 Unified Detection

We unify the separate components of object detection into a single neural network. Our network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. This means our network reasons globally about the full image and all the objects in the image. The YOLO design enables end-to-end training and real-time speeds while maintaining high average precision.

Our system divides the input image into an $S \times S$ grid. If the center of an object falls into a

grid cell, that grid cell is responsible for detecting that object.

Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally we define confidence as $\Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}}$. If no object exists in that cell, the confidence scores should be zero. Otherwise we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box.

Each grid cell also predicts C conditional class probabilities, $\Pr(\text{Class}_i|\text{Object})$. These probabilities are conditioned on the grid cell containing an object. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B.

At test time we multiply the conditional class probabilities and the individual box confidence predictions,

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(\text{Class}_i) * \text{IOU}^{\text{truth}}_{\text{pred}} \tag{1.1}$$

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

For evaluating YOLO on PASCAL VOC, we use $S = 7$, $B = 2$. PASCAL VOC has 20 labelled classes so $C = 20$. Our final prediction is a $7 \times 7 \times 30$ tensor.

1.2.1 Network Design

We implement this model as a convolutional neural network and evaluate it on the PASCAL VOC detection dataset [29]. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates.

Our network architecture is inspired by the GoogLeNet model for image classification [119]. Our network has 24 convolutional layers followed by 2 fully connected layers. Instead of the

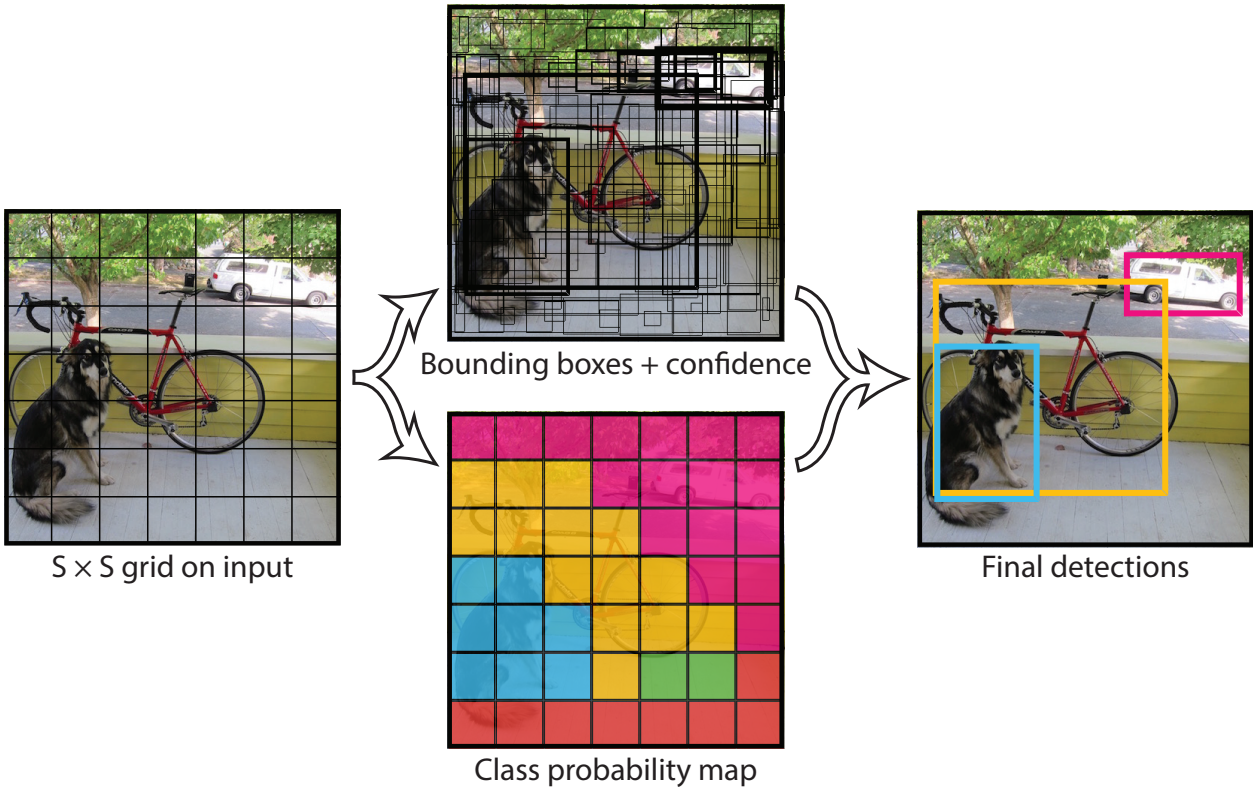


Figure 1.2: The Model. Our system models detection as a regression problem. It divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

inception modules used by GoogLeNet, we simply use 1×1 reduction layers followed by 3×3 convolutional layers, similar to Lin et al [75]. The full network is shown in Figure 3.1.

We also train a fast version of YOLO designed to push the boundaries of fast object detection. Fast YOLO uses a neural network with fewer convolutional layers (9 instead of 24) and fewer filters in those layers. Other than the size of the network, all training and testing parameters are the same between YOLO and Fast YOLO.

The final output of our network is the $7 \times 7 \times 30$ tensor of predictions.

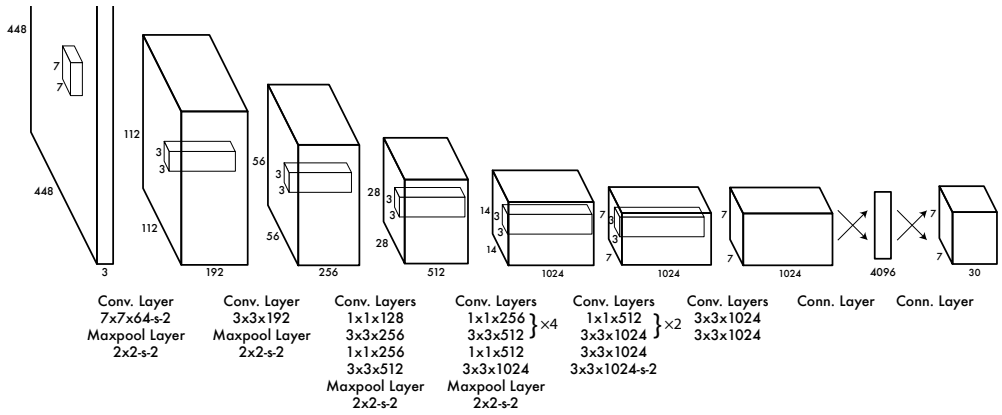


Figure 1.3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

1.2.2 Training

We pretrain our convolutional layers on the ImageNet 1000-class competition dataset [104]. For pretraining we use the first 20 convolutional layers from Figure 3.1 followed by a average-pooling layer and a fully connected layer. We train this network for approximately a week and achieve a single crop top-5 accuracy of 88% on the ImageNet 2012 validation set, comparable to the GoogLeNet models in Caffe's Model Zoo [87]. We use the Darknet framework for all training and inference [96].

We then convert the model to perform detection. Ren et al. show that adding both convolutional and connected layers to pretrained networks can improve performance [103]. Following their example, we add four convolutional layers and two fully connected layers with randomly initialized weights. Detection often requires fine-grained visual information so we increase the input resolution of the network from 224×224 to 448×448.

Our final layer predicts both class probabilities and bounding box coordinates. We normalize the bounding box width and height by the image width and height so that they fall between 0 and 1.

We parametrize the bounding box x and y coordinates to be offsets of a particular grid cell location so they are also bounded between 0 and 1.

We use a linear activation function for the final layer and all other layers use the following leaky rectified linear activation:

$$\phi(x) = \begin{cases} x, & \text{if } x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \tag{1.2}$$

We optimize for sum-squared error in the output of our model. We use sum-squared error because it is easy to optimize, however it does not perfectly align with our goal of maximizing average precision. It weights localization error equally with classification error which may not be ideal. Also, in every image many grid cells do not contain any object. This pushes the "confidence" scores of those cells towards zero, often overpowering the gradient from cells that do contain objects. This can lead to model instability, causing training to diverge early on.

To remedy this, we increase the loss from bounding box coordinate predictions and decrease the loss from confidence predictions for boxes that don't contain objects. We use two parameters, λ_{coord} and λ_{noobj} to accomplish this. We set $\lambda_{\text{coord}} = 5$ and $\lambda_{\text{noobj}} = .5$.

Sum-squared error also equally weights errors in large boxes and small boxes. Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.

YOLO predicts multiple bounding boxes per grid cell. At training time we only want one bounding box predictor to be responsible for each object. We assign one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors. Each predictor gets better at predicting certain sizes, aspect ratios, or classes of object, improving overall recall.

During training we optimize the following, multi-part loss function:

$$\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (1.3)
\end{aligned}$$

where $\mathbb{1}_i^{\text{obj}}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{\text{obj}}$ denotes that the jth bounding box predictor in cell i is "responsible" for that prediction.

Note that the loss function only penalizes classification error if an object is present in that grid cell (hence the conditional class probability discussed earlier). It also only penalizes bounding box coordinate error if that predictor is "responsible" for the ground truth box (i.e. has the highest IOU of any predictor in that grid cell).

We train the network for about 135 epochs on the training and validation data sets from PASCAL VOC 2007 and 2012. When testing on 2012 we also include the VOC 2007 test data for training. Throughout training we use a batch size of 64, a momentum of 0.9 and a decay of 0.0005.

Our learning rate schedule is as follows: For the first epochs we slowly raise the learning rate from 10^{-3} to 10^{-2}. If we start at a high learning rate our model often diverges due to unstable gradients. We continue training with 10^{-2} for 75 epochs, then 10^{-3} for 30 epochs, and finally 10^{-4} for 30 epochs.

To avoid overfitting we use dropout and extensive data augmentation. A dropout layer with rate = .5 after the first connected layer prevents co-adaptation between layers [56]. For data augmentation we introduce random scaling and translations of up to 20% of the original image size. We also randomly adjust the exposure and saturation of the image by up to a factor of 1.5 in the HSV color space.

1.2.3 Inference

Just like in training, predicting detections for a test image only requires one network evaluation. On PASCAL VOC the network predicts 98 bounding boxes per image and class probabilities for each box. YOLO is extremely fast at test time since it only requires a single network evaluation, unlike classifier-based methods.

The grid design enforces spatial diversity in the bounding box predictions. Often it is clear which grid cell an object falls in to and the network only predicts one box for each object. However, some large objects or objects near the border of multiple cells can be well localized by multiple cells. Non-maximal suppression can be used to fix these multiple detections. While not critical to performance as it is for R-CNN or DPM, non-maximal suppression adds 2-3% in mAP.

1.2.4 Limitations of YOLO

YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. Our model struggles with small objects that appear in groups, such as flocks of birds.

Since our model learns to predict bounding boxes from data, it struggles to generalize to objects in new or unusual aspect ratios or configurations. Our model also uses relatively coarse features for predicting bounding boxes since our architecture has multiple downsampling layers from the input image.

Finally, while we train on a loss function that approximates detection performance, our loss function treats errors the same in small bounding boxes versus large bounding boxes. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU. Our main source of error is incorrect localizations.

1.3 Comparison to Other Detection Systems

Object detection is a core problem in computer vision. Detection pipelines generally start by extracting a set of robust features from input images (Haar [91], SIFT [85], HOG [18], convolutional features [25]). Then, classifiers [126, 74, 40, 32] or localizers [8, 109] are used to identify objects in the feature space. These classifiers or localizers are run either in sliding window fashion over the whole image or on some subset of regions in the image [123, 46, 130]. We compare the YOLO detection system to several top detection frameworks, highlighting key similarities and differences.

Deformable parts models. Deformable parts models (DPM) use a sliding window approach to object detection [32]. DPM uses a disjoint pipeline to extract static features, classify regions, predict bounding boxes for high scoring regions, etc. Our system replaces all of these disparate parts with a single convolutional neural network. The network performs feature extraction, bounding box prediction, non-maximal suppression, and contextual reasoning all concurrently. Instead of static features, the network trains the features in-line and optimizes them for the detection task. Our unified architecture leads to a faster, more accurate model than DPM.

R-CNN. R-CNN and its variants use region proposals instead of sliding windows to find objects in images. Selective Search [123] generates potential bounding boxes, a convolutional network extracts features, an SVM scores the boxes, a linear model adjusts the bounding boxes, and non-max suppression eliminates duplicate detections. Each stage of this complex pipeline must be precisely tuned independently and the resulting system is very slow, taking more than 40 seconds per image at test time [41].

YOLO shares some similarities with R-CNN. Each grid cell proposes potential bounding boxes and scores those boxes using convolutional features. However, our system puts spatial constraints on the grid cell proposals which helps mitigate multiple detections of the same object. Our system also proposes far fewer bounding boxes, only 98 per image compared to about 2000 from Selective Search. Finally, our system combines these individual components into a single, jointly optimized model.

Other Fast Detectors Fast and Faster R-CNN focus on speeding up the R-CNN framework

by sharing computation and using neural networks to propose regions instead of Selective Search [41] [102]. While they offer speed and accuracy improvements over R-CNN, both still fall short of real-time performance.

Many research efforts focus on speeding up the DPM pipeline [106] [129] [20]. They speed up HOG computation, use cascades, and push computation to GPUs. However, only 30Hz DPM [106] actually runs in real-time.

Instead of trying to optimize individual components of a large detection pipeline, YOLO throws out the pipeline entirely and is fast by design.

Detectors for single classes like faces or people can be highly optimized since they have to deal with much less variation [127]. YOLO is a general purpose detector that learns to detect a variety of objects simultaneously.

Deep MultiBox. Unlike R-CNN, Szegedy et al. train a convolutional neural network to predict regions of interest [27] instead of using Selective Search. MultiBox can also perform single object detection by replacing the confidence prediction with a single class prediction. However, MultiBox cannot perform general object detection and is still just a piece in a larger detection pipeline, requiring further image patch classification. Both YOLO and MultiBox use a convolutional network to predict bounding boxes in an image but YOLO is a complete detection system.

OverFeat. Sermanet et al. train a convolutional neural network to perform localization and adapt that localizer to perform detection [109]. OverFeat efficiently performs sliding window detection but it is still a disjoint system. OverFeat optimizes for localization, not detection performance. Like DPM, the localizer only sees local information when making a prediction. OverFeat cannot reason about global context and thus requires significant post-processing to produce coherent detections.

MultiGrasp. Our work is similar in design to work on grasp detection by Redmon et al [97]. Our grid approach to bounding box prediction is based on the MultiGrasp system for regression to grasps. However, grasp detection is a much simpler task than object detection. MultiGrasp only needs to predict a single graspable region for an image containing one object. It doesn't have to estimate the size, location, or boundaries of the object or predict it's class, only find a region

suitable for grasping. YOLO predicts both bounding boxes and class probabilities for multiple objects of multiple classes in an image.

1.4 Experiments

First we compare YOLO with other real-time detection systems on PASCAL VOC 2007. To understand the differences between YOLO and R-CNN variants we explore the errors on VOC 2007 made by YOLO and Fast R-CNN, one of the highest performing versions of R-CNN [41]. Based on the different error profiles we show that YOLO can be used to rescore Fast R-CNN detections and reduce the errors from background false positives, giving a significant performance boost. We also present VOC 2012 results and compare mAP to current state-of-the-art methods. Finally, we show that YOLO generalizes to new domains better than other detectors on two artwork datasets.

1.4.1 Comparison to Other Real-Time Systems

Many research efforts in object detection focus on making standard detection pipelines fast. [20] [129] [106] [41] [53] [102] However, only Sadeghi et al. actually produce a detection system that runs in real-time (30 frames per second or better) [106]. We compare YOLO to their GPU implementation of DPM which runs either at 30Hz or 100Hz. While the other efforts don't reach the real-time milestone we also compare their relative mAP and speed to examine the accuracy-performance tradeoffs available in object detection systems.

Fast YOLO is the fastest object detection method on PASCAL; as far as we know, it is the fastest extant object detector. With 52.7% mAP, it is more than twice as accurate as prior work on real-time detection. YOLO pushes mAP to 63.4% while still maintaining real-time performance.

We also train YOLO using VGG-16. This model is more accurate but also significantly slower than YOLO. It is useful for comparison to other detection systems that rely on VGG-16 but since it is slower than real-time the rest of the paper focuses on our faster models.

Fastest DPM effectively speeds up DPM without sacrificing much mAP but it still misses real-time performance by a factor of 2 [129]. It also is limited by DPM's relatively low accuracy on

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [106]	2007	16.0	100
30Hz DPM [106]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [129]	2007	30.4	15
R-CNN Minus R [73]	2007	53.5	6
Fast R-CNN [41]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[102]	2007+2012	73.2	7
Faster R-CNN ZF [102]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Table 1.1: Real-Time Systems on PASCAL VOC 2007. Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

detection compared to neural network approaches.

R-CNN minus R replaces Selective Search with static bounding box proposals [73]. While it is much faster than R-CNN, it still falls short of real-time and takes a significant accuracy hit from not having good proposals.

Fast R-CNN speeds up the classification stage of R-CNN but it still relies on selective search which can take around 2 seconds per image to generate bounding box proposals. Thus it has high mAP but at 0.5 fps it is still far from real-time.

The recent Faster R-CNN replaces selective search with a neural network to propose bounding boxes, similar to Szegedy et al. [27] In our tests, their most accurate model achieves 7 fps while a

smaller, less accurate one runs at 18 fps. The VGG-16 version of Faster R-CNN is 10 mAP higher but is also 6 times slower than YOLO. The Zeiler-Fergus Faster R-CNN is only 2.5 times slower than YOLO but is also less accurate.

1.4.2 VOC 2007 Error Analysis

To further examine the differences between YOLO and state-of-the-art detectors, we look at a detailed breakdown of results on VOC 2007. We compare YOLO to Fast R-CNN since Fast R-CNN is one of the highest performing detectors on PASCAL and it's detections are publicly available.

We use the methodology and tools of Hoiem et al. [57] For each category at test time we look at the top N predictions for that category. Each prediction is either correct or it is classified based on the type of error:

- Correct: correct class and IOU $> .5$
- Localization: correct class, $.1 <$ IOU $< .5$
- Similar: class is similar, IOU $> .1$
- Other: class is wrong, IOU $> .1$
- Background: IOU $< .1$ for any object

Figure 1.4 shows the breakdown of each error type averaged across all 20 classes.

YOLO struggles to localize objects correctly. Localization errors account for more of YOLO's errors than all other sources combined. Fast R-CNN makes much fewer localization errors but far more background errors. 13.6% of it's top detections are false positives that don't contain any objects. Fast R-CNN is almost 3x more likely to predict background detections than YOLO.

1.4.3 Combining Fast R-CNN and YOLO

YOLO makes far fewer background mistakes than Fast R-CNN. By using YOLO to eliminate background detections from Fast R-CNN we get a significant boost in performance. For every

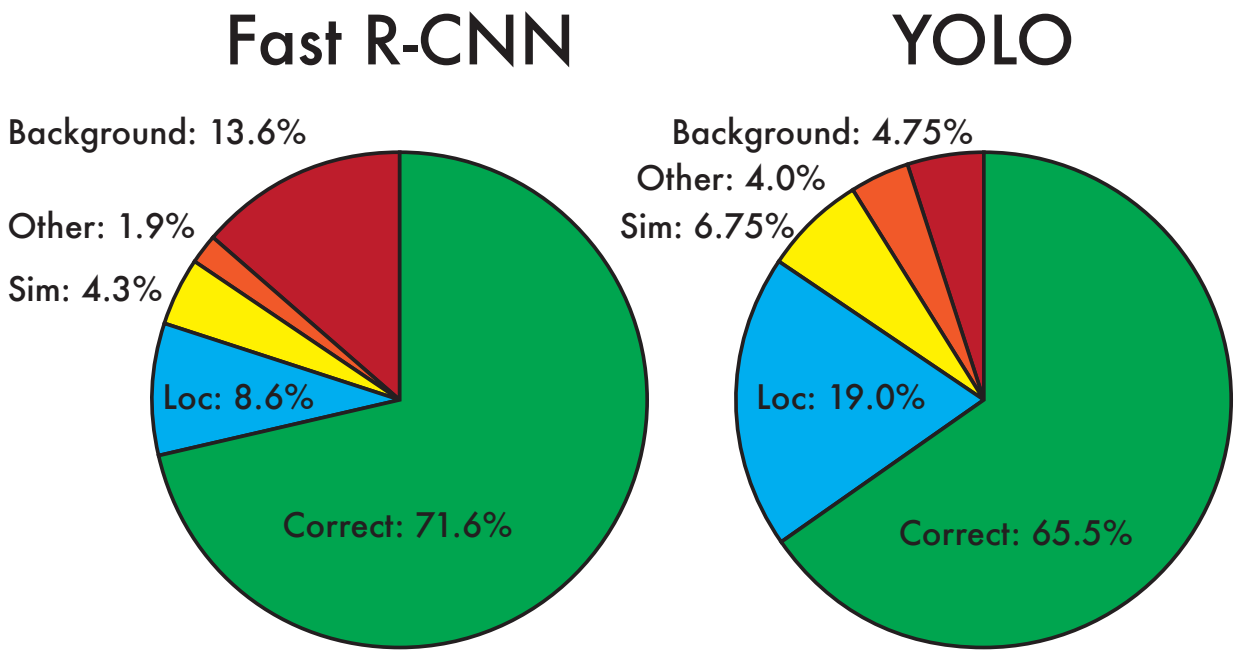


Figure 1.4: Error Analysis: Fast R-CNN vs. YOLO These charts show the percentage of localization and background errors in the top N detections for various categories (N = # objects in that category).

bounding box that R-CNN predicts we check to see if YOLO predicts a similar box. If it does, we give that prediction a boost based on the probability predicted by YOLO and the overlap between the two boxes.

The best Fast R-CNN model achieves a mAP of 71.8% on the VOC 2007 test set. When combined with YOLO, its mAP increases by 3.2% to 75.0%. We also tried combining the top Fast R-CNN model with several other versions of Fast R-CNN. Those ensembles produced small increases in mAP between .3 and .6%, see Table 1.2 for details.

The boost from YOLO is not simply a byproduct of model ensembling since there is little benefit from combining different versions of Fast R-CNN. Rather, it is precisely because YOLO makes different kinds of mistakes at test time that it is so effective at boosting Fast R-CNN's performance.

Unfortunately, this combination doesn't benefit from the speed of YOLO since we run each

model seperately and then combine the results. However, since YOLO is so fast it doesn't add any significant computational time compared to Fast R-CNN.

1.4.4 VOC 2012 Results

On the VOC 2012 test set, YOLO scores 57.9% mAP. This is lower than the current state of the art, closer to the original R-CNN using VGG-16, see Table 3.3. Our system struggles with small objects compared to its closest competitors. On categories like `bottle`, `sheep`, and `tv/monitor` YOLO scores 8-10% lower than R-CNN or Feature Edit. However, on other categories like `cat` and `train` YOLO achieves higher performance.

Our combined Fast R-CNN + YOLO model is one of the highest performing detection methods. Fast R-CNN gets a 2.3% improvement from the combination with YOLO, boosting it 5 spots up on the public leaderboard.

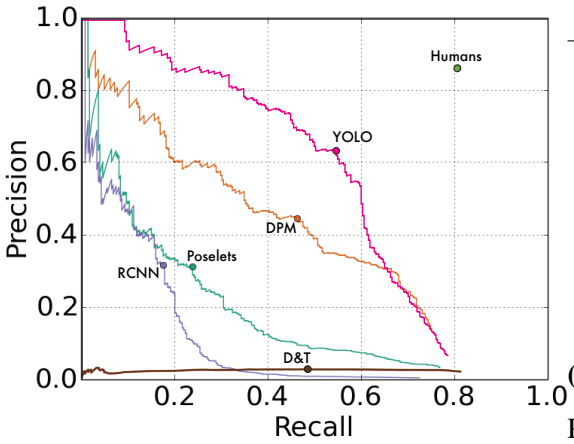
1.4.5 Generalizability: Person Detection in Artwork

Academic datasets for object detection draw the training and testing data from the same distribution. In real-world applications it is hard to predict all possible use cases and the test data can diverge from what the system has seen before [10]. We compare YOLO to other detection systems on the Picasso Dataset [37] and the People-Art Dataset [10], two datasets for testing person detection on artwork.

Figure 1.5 shows comparative performance between YOLO and other detection methods. For reference, we give VOC 2007 detection AP on `person` where all models are trained only on VOC 2007 data. On Picasso models are trained on VOC 2012 while on People-Art they are trained on VOC 2010.

R-CNN has high AP on VOC 2007. However, R-CNN drops off considerably when applied to artwork. R-CNN uses Selective Search for bounding box proposals which is tuned for natural images. The classifier step in R-CNN only sees small regions and needs good proposals.

DPM maintains its AP well when applied to artwork. Prior work theorizes that DPM performs



(a) Picasso Dataset precision-recall curves.

	VOC 2007	Picasso		People-Art
	AP	AP	Best F_1	AP
YOLO	**59.2**	**53.3**	**0.590**	**45**
R-CNN	54.2	10.4	0.226	26
DPM	43.2	37.8	0.458	32
Poselets [9]	36.5	17.8	0.271	
D&T [18]	-	1.9	0.051	

(b) Quantitative results on the VOC 2007, Picasso, and People-Art Datasets. The Picasso Dataset evaluates on both AP and best F_1 score.

Figure 1.5: Generalization results on Picasso and People-Art datasets.

well because it has strong spatial models of the shape and layout of objects. Though DPM doesn't degrade as much as R-CNN, it starts from a lower AP.

YOLO has good performance on VOC 2007 and its AP degrades less than other methods when applied to artwork. Like DPM, YOLO models the size and shape of objects, as well as relationships between objects and where objects commonly appear. Artwork and natural images are very different on a pixel level but they are similar in terms of the size and shape of objects, thus YOLO can still predict good bounding boxes and detections.

1.5 Real-Time Detection In The Wild

YOLO is a fast, accurate object detector, making it ideal for computer vision applications. We connect YOLO to a webcam and verify that it maintains real-time performance, including the time to fetch images from the camera and display the detections.

The resulting system is interactive and engaging. While YOLO processes images individually, when attached to a webcam it functions like a tracking system, detecting objects as they move

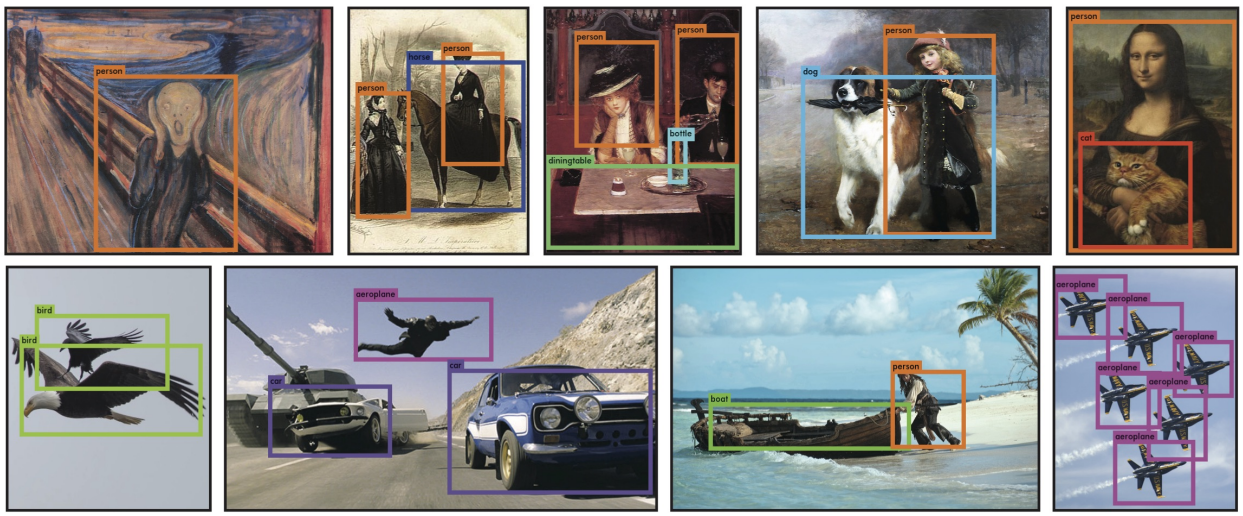


Figure 1.6: Qualitative Results. YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

around and change in appearance. A demo of the system and the source code can be found on our project website: http://pjreddie.com/yolo/.

1.6 Conclusion

We introduce YOLO, a unified model for object detection. Our model is simple to construct and can be trained directly on full images. Unlike classifier-based approaches, YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly.

Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection.

Acknowledgements: This work is partially supported by ONR N00014-13-1-0720, NSF IIS-1338054, and The Allen Distinguished Investigator Award.

	mAP	Combined	Gain
Fast R-CNN	71.8	-	-
Fast R-CNN (2007 data)	**66.9**	72.4	.6
Fast R-CNN (VGG-M)	59.2	72.4	.6
Fast R-CNN (CaffeNet)	57.1	72.1	.3
YOLO	63.4	**75.0**	**3.2**

Table 1.2: Model combination experiments on VOC 2007. We examine the effect of combining various models with the best version of Fast R-CNN. Other versions of Fast R-CNN provide only a small benefit while YOLO provides a significant performance boost.

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [36]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [36]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [102]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [103]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [41]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [26]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [26]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [40]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [40]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [110]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [40]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [50]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [40]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Table 1.3: PASCAL VOC 2012 Leaderboard. YOLO compared with the full `comp4` (outside data allowed) public leaderboard as of November 6th, 2015. Mean average precision and per-class average precision are shown for a variety of detection methods. YOLO is the only real-time detector. Fast R-CNN + YOLO is the forth highest scoring method, with a 2.3% boost over Fast R-CNN.

Chapter 2

YOLO9000: BETTER, FASTER, STRONGER

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. YOLO9000 predicts detections for more than 9000 different object categories, all in real-time.

2.1 Introduction

General purpose object detection should be fast, accurate, and able to recognize a wide variety of objects. Since the introduction of neural networks, detection frameworks have become increasingly fast and accurate. However, most detection methods are still constrained to a small set of objects.

Current object detection datasets are limited compared to datasets for other tasks like classification and tagging. The most common detection datasets contain thousands to hundreds of thousands

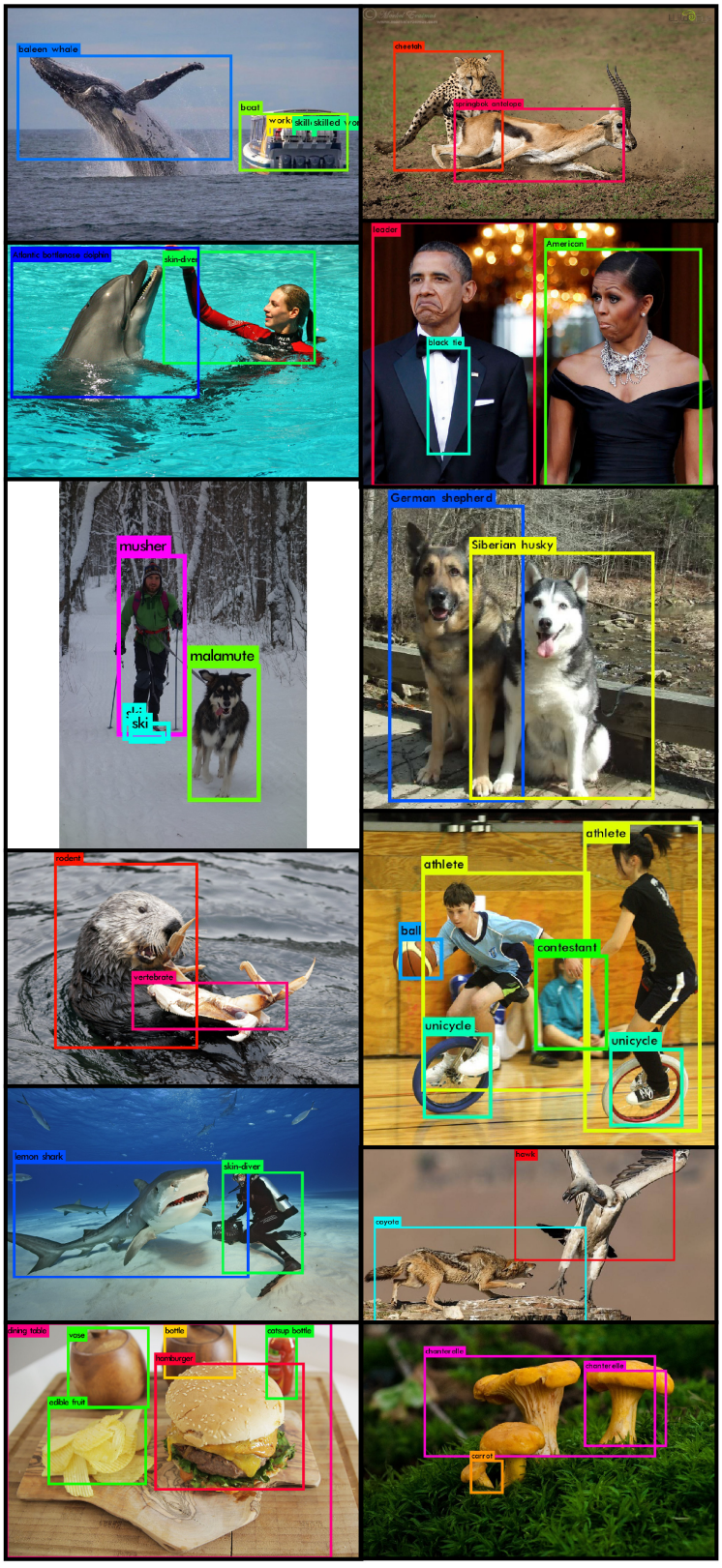


Figure 2.1: YOLO9000. YOLO9000 can detect a wide variety of object classes in real-time.

of images with dozens to hundreds of tags [30] [79] [22]. Classification datasets have millions of images with tens or hundreds of thousands of categories [122] [22].

We would like detection to scale to level of object classification. However, labelling images for detection is far more expensive than labelling for classification or tagging (tags are often user-supplied for free). Thus we are unlikely to see detection datasets on the same scale as classification datasets in the near future.

We propose a new method to harness the large amount of classification data we already have and use it to expand the scope of current detection systems. Our method uses a hierarchical view of object classification that allows us to combine distinct datasets together.

We also propose a joint training algorithm that allows us to train object detectors on both detection and classification data. Our method leverages labeled detection images to learn to precisely localize objects while it uses classification images to increase its vocabulary and robustness.

Using this method we train YOLO9000, a real-time object detector that can detect over 9000 different object categories. First we improve upon the base YOLO detection system to produce YOLOv2, a state-of-the-art, real-time detector. Then we use our dataset combination method and joint training algorithm to train a model on more than 9000 classes from ImageNet as well as detection data from COCO.

All of our code and pre-trained models are available online at `http://pjreddie.com/yolo9000/`.

2.2 Better

YOLO suffers from a variety of shortcomings relative to state-of-the-art detection systems. Error analysis of YOLO compared to Fast R-CNN shows that YOLO makes a significant number of localization errors. Furthermore, YOLO has relatively low recall compared to region proposal-based methods. Thus we focus mainly on improving recall and localization while maintaining classification accuracy.

Computer vision generally trends towards larger, deeper networks [55] [116] [113]. Better performance often hinges on training larger networks or ensembling multiple models together.

However, with YOLOv2 we want a more accurate detector that is still fast. Instead of scaling up our network, we simplify the network and then make the representation easier to learn. We pool a variety of ideas from past work with our own novel concepts to improve YOLO's performance. A summary of results can be found in Table 2.2.

Batch Normalization. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization [62]. By adding batch normalization on all of the convolutional layers in YOLO we get more than 2% improvement in mAP. Batch normalization also helps regularize the model. With batch normalization we can remove dropout from the model without overfitting.

High Resolution Classifier. All state-of-the-art detection methods use classifier pre-trained on ImageNet [104]. Starting with AlexNet most classifiers operate on input images smaller than 256×256 [69]. The original YOLO trains the classifier network at 224×224 and increases the resolution to 448 for detection. This means the network has to simultaneously switch to learning object detection and adjust to the new input resolution.

For YOLOv2 we first fine tune the classification network at the full 448×448 resolution for 10 epochs on ImageNet. This gives the network time to adjust its filters to work better on higher resolution input. We then fine tune the resulting network on detection. This high resolution classification network gives us an increase of almost 4% mAP.

Convolutional With Anchor Boxes. YOLO predicts the coordinates of bounding boxes directly using fully connected layers on top of the convolutional feature extractor. Instead of predicting coordinates directly Faster R-CNN predicts bounding boxes using hand-picked priors [102]. Using only convolutional layers the region proposal network (RPN) in Faster R-CNN predicts offsets and confidences for anchor boxes. Since the prediction layer is convolutional, the RPN predicts these offsets at every location in a feature map. Predicting offsets instead of coordinates simplifies the problem and makes it easier for the network to learn.

We remove the fully connected layers from YOLO and use anchor boxes to predict bounding boxes. First we eliminate one pooling layer to make the output of the network's convolutional layers higher resolution. We also shrink the network to operate on 416 input images instead of

448×448. We do this because we want an odd number of locations in our feature map so there is a single center cell. Objects, especially large objects, tend to occupy the center of the image so it's good to have a single location right at the center to predict these objects instead of four locations that are all nearby. YOLO's convolutional layers downsample the image by a factor of 32 so by using an input image of 416 we get an output feature map of 13×13.

When we move to anchor boxes we also decouple the class prediction mechanism from the spatial location and instead predict class and objectness for every anchor box. Following YOLO, the objectness prediction still predicts the IOU of the ground truth and the proposed box and the class predictions predict the conditional probability of that class given that there is an object.

Using anchor boxes we get a small decrease in accuracy. YOLO only predicts 98 boxes per image but with anchor boxes our model predicts more than a thousand. Without anchor boxes our intermediate model gets 69.5 mAP with a recall of 81%. With anchor boxes our model gets 69.2 mAP with a recall of 88%. Even though the mAP decreases, the increase in recall means that our model has more room to improve.

Dimension Clusters. We encounter two issues with anchor boxes when using them with YOLO. The first is that the box dimensions are hand picked. The network can learn to adjust the boxes appropriately but if we pick better priors for the network to start with we can make it easier for the network to learn to predict good detections.

Instead of choosing priors by hand, we run k-means clustering on the training set bounding boxes to automatically find good priors. If we use standard k-means with Euclidean distance larger boxes generate more error than smaller boxes. However, what we really want are priors that lead to good IOU scores, which is independent of the size of the box. Thus for our distance metric we use:

$$d(\text{box}, \text{centroid}) = 1 - \text{IOU}(\text{box}, \text{centroid})$$

We run k-means for various values of k and plot the average IOU with closest centroid, see Figure 2.2. We choose $k = 5$ as a good tradeoff between model complexity and high recall. The cluster centroids are significantly different than hand-picked anchor boxes. There are fewer short,

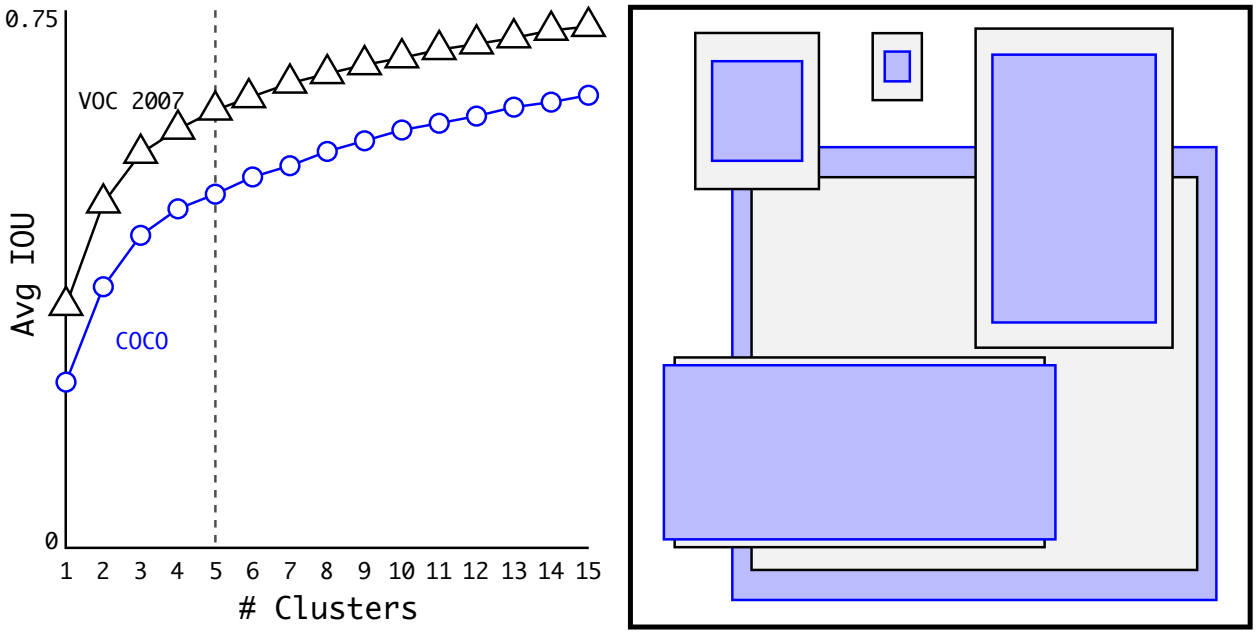


Figure 2.2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. COCO has greater variation in size than VOC.

wide boxes and more tall, thin boxes.

We compare the average IOU to closest prior of our clustering strategy and the hand-picked anchor boxes in Table 2.1. At only 5 priors the centroids perform similarly to 9 anchor boxes with an average IOU of 61.0 compared to 60.9. If we use 9 centroids we see a much higher average IOU. This indicates that using k-means to generate our bounding box starts the model off with a better representation and makes the task easier to learn.

Direct location prediction. When using anchor boxes with YOLO we encounter a second issue: model instability, especially during early iterations. Most of the instability comes from predicting the (x, y) locations for the box. In region proposal networks the network predicts values t_x and t_y and the (x, y) center coordinates are calculated as:

Box Generation	#	Avg IOU
Cluster SSE	5	58.7
Cluster IOU	5	61.0
Anchor Boxes [102]	9	60.9
Cluster IOU	9	67.2

Table 2.1: Average IOU of boxes to closest priors on VOC 2007. The average IOU of objects on VOC 2007 to their closest, unmodified prior using different generation methods. Clustering gives much better results than using hand-picked priors.

$$x = (t_x * w_a) - x_a$$
$$y = (t_y * h_a) - y_a$$

For example, a prediction of $t_x = 1$ would shift the box to the right by the width of the anchor box, a prediction of $t_x = -1$ would shift it to the left by the same amount.

This formulation is unconstrained so any anchor box can end up at any point in the image, regardless of what location predicted the box. With random initialization the model takes a long time to stabilize to predicting sensible offsets.

Instead of predicting offsets we follow the approach of YOLO and predict location coordinates relative to the location of the grid cell. This bounds the ground truth to fall between 0 and 1. We use a logistic activation to constrain the network's predictions to fall in this range.

The network predicts 5 bounding boxes at each cell in the output feature map. The network predicts 5 coordinates for each bounding box, t_x, t_y, t_w, t_h, and t_o. If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h} \\
Pr(\text{object}) * IOU(b, \text{object}) &= \sigma(t_o)
\end{aligned}
$$

Since we constrain the location prediction the parametrization is easier to learn, making the network more stable. Using dimension clusters along with directly predicting the bounding box center location improves YOLO by almost 5% over the version with anchor boxes.

Fine-Grained Features. This modified YOLO predicts detections on a 13×13 feature map. While this is sufficient for large objects, it may benefit from finer grained features for localizing smaller objects. Faster R-CNN and SSD both run their proposal networks at various feature maps in the network to get a range of resolutions. We take a different approach, simply adding a passthrough layer that brings features from an earlier layer at 26×26 resolution.

The passthrough layer concatenates the higher resolution features with the low resolution features by stacking adjacent features into different channels instead of spatial locations, similar to the identity mappings in ResNet. This turns the $26 \times 26 \times 512$ feature map into a $13 \times 13 \times 2048$ feature map, which can be concatenated with the original features. Our detector runs on top of this expanded feature map so that it has access to fine grained features. This gives a modest 1% performance increase.

Multi-Scale Training. The original YOLO uses an input resolution of 448×448. With the addition of anchor boxes we changed the resolution to 416×416. However, since our model only uses convolutional and pooling layers it can be resized on the fly. We want YOLOv2 to be robust to running on images of different sizes so we train this into the model.

Instead of fixing the input image size we change the network every few iterations. Every 10 batches our network randomly chooses new image dimensions. Since our model downsamples by

a factor of 32, we pull from the following multiples of 32: $\{320, 352, ..., 608\}$. Thus the smallest option is 320×320 and the largest is 608×608. We resize the network to that dimension and continue training.

This regime forces the network to learn to predict well across a variety of input dimensions. This means the same network can predict detections at different resolutions. The network runs faster at smaller sizes so YOLOv2 offers an easy tradeoff between speed and accuracy.

At low resolutions YOLOv2 operates as a cheap, fairly accurate detector. At 288×288 it runs at more than 90 FPS with mAP almost as good as Fast R-CNN. This makes it ideal for smaller GPUs, high framerate video, or multiple video streams.

At high resolution YOLOv2 is a state-of-the-art detector with 78.6 mAP on VOC 2007 while still operating above real-time speeds. See Table 2.3 for a comparison of YOLOv2 with other frameworks on VOC 2007. Figure 2.4

Further Experiments. We train YOLOv2 for detection on VOC 2012. Table 2.4 shows the comparative performance of YOLOv2 versus other state-of-the-art detection systems. YOLOv2 achieves 73.4 mAP while running far faster than other methods. We also train on COCO, see Table 2.5. On the VOC metric (IOU = .5) YOLOv2 gets 44.0 mAP, comparable to SSD and Faster R-CNN.

2.3 Faster

We want detection to be accurate but we also want it to be fast. Most applications for detection, like robotics or self-driving cars, rely on low latency predictions. In order to maximize performance we design YOLOv2 to be fast from the ground up.

Most detection frameworks rely on VGG-16 as the base feature extractor [113]. VGG-16 is a powerful, accurate classification network but it is needlessly complex. The convolutional layers of VGG-16 require 30.69 billion floating point operations for a single pass over a single image at 224×224 resolution.

The YOLO framework uses a custom network based on the Googlenet architecture [120]. This network is faster than VGG-16, only using 8.52 billion operations for a forward pass. However, it's

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Table 2.2: The path from YOLO to YOLOv2. Most of the listed design decisions lead to significant increases in mAP. Two exceptions are switching to a fully convolutional network with anchor boxes and using the new network. Switching to the anchor box style approach increased recall without changing mAP while using the new network cut computation by 33%.

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [42]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[102]	2007+2012	73.2	7
Faster R-CNN ResNet[55]	2007+2012	76.4	5
YOLO [98]	2007+2012	63.4	45
SSD300 [83]	2007+2012	74.3	46
SSD500 [83]	2007+2012	76.8	19
YOLOv2 288×288	2007+2012	69.0	91
YOLOv2 352×352	2007+2012	73.7	81
YOLOv2 416×416	2007+2012	76.8	67
YOLOv2 480×480	2007+2012	77.8	59
YOLOv2 544×544	2007+2012	**78.6**	40

Table 2.3: Detection frameworks on PASCAL VOC 2007. YOLOv2 is faster and more accurate than prior detection methods. It can also run at different resolutions for an easy tradeoff between speed and accuracy. Each YOLOv2 entry is actually the same trained model with the same weights, just evaluated at a different size. All timing information is on a Geforce GTX Titan X (original, not Pascal model).

accuracy is slightly worse than VGG-16. For single-crop, top-5 accuracy at 224×224, YOLO's custom model gets 88.0% ImageNet compared to 90.0% for VGG-16.

Darknet-19. We propose a new classification model to be used as the base of YOLOv2. Our model builds off of prior work on network design as well as common knowledge in the field. Similar to the VGG models we use mostly 3×3 filters and double the number of channels after every pooling step [113]. Following the work on Network in Network (NIN) we use global average pooling to make predictions as well as 1×1 filters to compress the feature representation between 3×3 convolutions [76]. We use batch normalization to stabilize training, speed up convergence, and regularize the model [62].

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [42]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [102]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [98]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [83]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [83]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [55]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

Table 2.4: PASCAL VOC2012 `test` detection results. YOLOv2 performs on par with state-of-the-art detectors like Faster R-CNN with ResNet and SSD512 and is $2-10\times$ faster.

Our final model, called Darknet-19, has 19 convolutional layers and 5 maxpooling layers. For a full description see Table 3.1. Darknet-19 only requires 5.58 billion operations to process an image yet achieves 72.9% top-1 accuracy and 91.2% top-5 accuracy on ImageNet.

Training for classification. We train the network on the standard ImageNet 1000 class classification dataset for 160 epochs using stochastic gradient descent with a starting learning rate of 0.1, polynomial rate decay with a power of 4, weight decay of 0.0005 and momentum of 0.9 using the Darknet neural network framework [96]. During training we use standard data augmentation tricks including random crops, rotations, and hue, saturation, and exposure shifts.

As discussed above, after our initial training on images at 224×224 we fine tune our network at a larger size, 448. For this fine tuning we train with the above parameters but for only 10 epochs and starting at a learning rate of 10^{-3}. At this higher resolution our network achieves a top-1 accuracy of 76.5% and a top-5 accuracy of 93.3%.

Training for detection. We modify this network for detection by removing the last convolutional layer and instead adding on three 3×3 convolutional layers with 1024 filters each followed by a final 1×1 convolutional layer with the number of outputs we need for detection. For VOC we predict 5 boxes with 5 coordinates each and 20 classes per box so 125 filters. We also add a passthrough layer from the final $3 \times 3 \times 512$ layer to the second to last convolutional layer so that our model can use fine grain features.

		0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Fast R-CNN [42]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast R-CNN[7]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster R-CNN[102]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [7]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster R-CNN[79]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300 [83]	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512 [83]	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**
YOLOv2 [83]	trainval35k	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4

Table 2.5: Results on COCO `test-dev2015`. Table adapted from [83]

We train the network for 160 epochs with a starting learning rate of 10^{-3}, dividing it by 10 at 60 and 90 epochs. We use a weight decay of 0.0005 and momentum of 0.9. We use a similar data augmentation to YOLO and SSD with random crops, color shifting, etc. We use the same training strategy on COCO and VOC.

2.4 Stronger

We propose a mechanism for jointly training on classification and detection data. Our method uses images labelled for detection to learn detection-specific information like bounding box coordinate prediction and objectness as well as how to classify common objects. It uses images with only class labels to expand the number of categories it can detect.

During training we mix images from both detection and classification datasets. When our network sees an image labelled for detection we can backpropagate based on the full YOLOv2 loss function. When it sees a classification image we only backpropagate loss from the classification-specific parts of the architecture.

This approach presents a few challenges. Detection datasets have only common objects and

general labels, like “dog” or “boat”. Classification datasets have a much wider and deeper range of labels. ImageNet has more than a hundred breeds of dog, including “Norfolk terrier”, “Yorkshire terrier”, and “Bedlington terrier”. If we want to train on both datasets we need a coherent way to merge these labels.

Most approaches to classification use a softmax layer across all the possible categories to compute the final probability distribution. Using a softmax assumes the classes are mutually exclusive. This presents problems for combining datasets, for example you would not want to combine ImageNet and COCO using this model because the classes “Norfolk terrier” and “dog” are not mutually exclusive.

We could instead use a multi-label model to combine the datasets which does not assume mutual exclusion. This approach ignores all the structure we do know about the data, for example that all of the COCO classes are mutually exclusive.

Hierarchical classification. ImageNet labels are pulled from WordNet, a language database that structures concepts and how they relate [86]. In WordNet, “Norfolk terrier” and “Yorkshire terrier” are both hyponyms of “terrier” which is a type of “hunting dog”, which is a type of “dog”, which is a “canine”, etc. Most approaches to classification assume a flat structure to the labels however for combining datasets, structure is exactly what we need.

WordNet is structured as a directed graph, not a tree, because language is complex. For example a “dog” is both a type of “canine” and a type of “domestic animal” which are both synsets in WordNet. Instead of using the full graph structure, we simplify the problem by building a hierarchical tree from the concepts in ImageNet.

To build this tree we examine the visual nouns in ImageNet and look at their paths through the WordNet graph to the root node, in this case “physical object”. Many synsets only have one path through the graph so first we add all of those paths to our tree. Then we iteratively examine the concepts we have left and add the paths that grow the tree by as little as possible. So if a concept has two paths to the root and one path would add three edges to our tree and the other would only add one edge, we choose the shorter path.

The final result is WordTree, a hierarchical model of visual concepts. To perform classifica-

tion with WordTree we predict conditional probabilities at every node for the probability of each hyponym of that synset given that synset. For example, at the "terrier" node we predict:

$$Pr(\text{Norfolk terrier}|\text{terrier})$$
$$Pr(\text{Yorkshire terrier}|\text{terrier})$$
$$Pr(\text{Bedlington terrier}|\text{terrier})$$
$$\ldots$$

If we want to compute the absolute probability for a particular node we simply follow the path through the tree to the root node and multiply to conditional probabilities. So if we want to know if a picture is of a Norfolk terrier we compute:

$$\begin{aligned} Pr(\text{Norfolk terrier}) = {} & Pr(\text{Norfolk terrier}|\text{terrier}) \\ & *Pr(\text{terrier}|\text{hunting dog}) \\ & *\ldots* \\ & *Pr(\text{mammal}|Pr(\text{animal}) \\ & *Pr(\text{animal}|\text{physical object}) \end{aligned}$$

For classification purposes we assume that the the image contains an object: $Pr(\text{physical object}) = 1$.

To validate this approach we train the Darknet-19 model on WordTree built using the 1000 class ImageNet. To build WordTree1k we add in all of the intermediate nodes which expands the label space from 1000 to 1369. During training we propagate ground truth labels up the tree so that if an image is labelled as a "Norfolk terrier" it also gets labelled as a "dog" and a "mammal", etc. To compute the conditional probabilities our model predicts a vector of 1369 values and we compute the softmax over all sysnsets that are hyponyms of the same concept, see Figure 2.5.

Using the same training parameters as before, our hierarchical Darknet-19 achieves 71.9% top-1 accuracy and 90.4% top-5 accuracy. Despite adding 369 additional concepts and having our network predict a tree structure our accuracy only drops marginally. Performing classification in this manner also has some benefits. Performance degrades gracefully on new or unknown object categories. For example, if the network sees a picture of a dog but is uncertain what type of dog it is, it will still predict "dog" with high confidence but have lower confidences spread out among the hyponyms.

This formulation also works for detection. Now, instead of assuming every image has an object, we use YOLOv2's objectness predictor to give us the value of $Pr(\text{physical object})$. The detector predicts a bounding box and the tree of probabilities. We traverse the tree down, taking the highest confidence path at every split until we reach some threshold and we predict that object class.

Dataset combination with WordTree. We can use WordTree to combine multiple datasets together in a sensible fashion. We simply map the categories in the datasets to synsets in the tree. Figure 2.6 shows an example of using WordTree to combine the labels from ImageNet and COCO. WordNet is extremely diverse so we can use this technique with most datasets.

Joint classification and detection. Now that we can combine datasets using WordTree we can train our joint model on classification and detection. We want to train an extremely large scale detector so we create our combined dataset using the COCO detection dataset and the top 9000 classes from the full ImageNet release. We also need to evaluate our method so we add in any classes from the ImageNet detection challenge that were not already included. The corresponding WordTree for this dataset has 9418 classes. ImageNet is a much larger dataset so we balance the dataset by oversampling COCO so that ImageNet is only larger by a factor of 4:1.

Using this dataset we train YOLO9000. We use the base YOLOv2 architecture but only 3 priors instead of 5 to limit the output size. When our network sees a detection image we backpropagate loss as normal. For classification loss, we only backpropagate loss at or above the corresponding level of the label. For example, if the label is "dog" we do assign any error to predictions further down in the tree, "German Shepherd" versus "Golden Retriever", because we do not have that information.

When it sees a classification image we only backpropagate classification loss. To do this we simply find the bounding box that predicts the highest probability for that class and we compute the loss on just its predicted tree. We also assume that the predicted box overlaps what would be the ground truth label by at least .3 IOU and we backpropagate objectness loss based on this assumption.

Using this joint training, YOLO9000 learns to find objects in images using the detection data in COCO and it learns to classify a wide variety of these objects using data from ImageNet.

We evaluate YOLO9000 on the ImageNet detection task. The detection task for ImageNet shares on 44 object categories with COCO which means that YOLO9000 has only seen classification data for the majority of the test categories. YOLO9000 gets 19.7 mAP overall with 16.0 mAP on the disjoint 156 object classes that it has never seen any labelled detection data for. This mAP is higher than results achieved by DPM but YOLO9000 is trained on different datasets with only partial supervision [33]. It also is simultaneously detecting 9000 other categories, all in real-time.

YOLO9000 learns new species of animals well but struggles with learning categories like clothing and equipment. New animals are easier to learn because the objectness predictions generalize well from the animals in COCO. Conversely, COCO does not have bounding box label for any type of clothing, only for person, so YOLO9000 struggles to model categories like “sunglasses” or “swimming trunks”.

2.5 Conclusion

We introduce YOLOv2 and YOLO9000, real-time detection systems. YOLOv2 is state-of-the-art and faster than other detection systems across a variety of detection datasets. Furthermore, it can be run at a variety of image sizes to provide a smooth tradeoff between speed and accuracy.

YOLO9000 is a real-time framework for detection more than 9000 object categories by jointly optimizing detection and classification. We use WordTree to combine data from various sources and our joint optimization technique to train simultaneously on ImageNet and COCO. YOLO9000 is a strong step towards closing the dataset size gap between detection and classification.

Many of our techniques generalize outside of object detection. Our WordTree representation of

ImageNet offers a richer, more detailed output space for image classification. Dataset combination using hierarchical classification would be useful in the classification and segmentation domains. Training techniques like multi-scale training could provide benefit across a variety of visual tasks.

For future work we hope to use similar techniques for weakly supervised image segmentation. We also plan to improve our detection results using more powerful matching strategies for assigning weak labels to classification data during training. Computer vision is blessed with an enormous amount of labelled data. We will continue looking for ways to bring different sources and structures of data together to make stronger models of the visual world.

Acknowledgements: We would like to thank Junyuan Xie for helpful discussions about constructing WordTree.

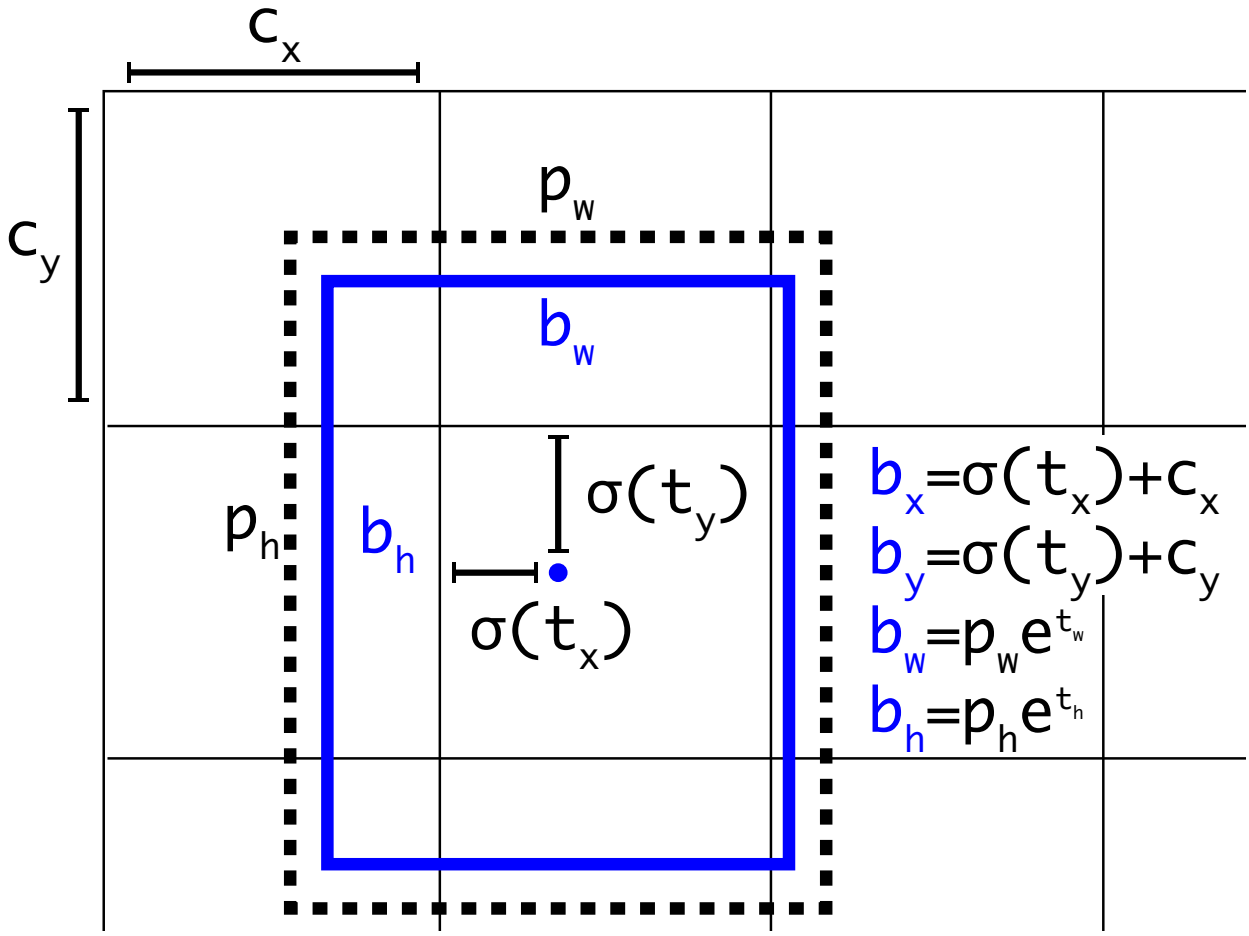


Figure 2.3: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function.

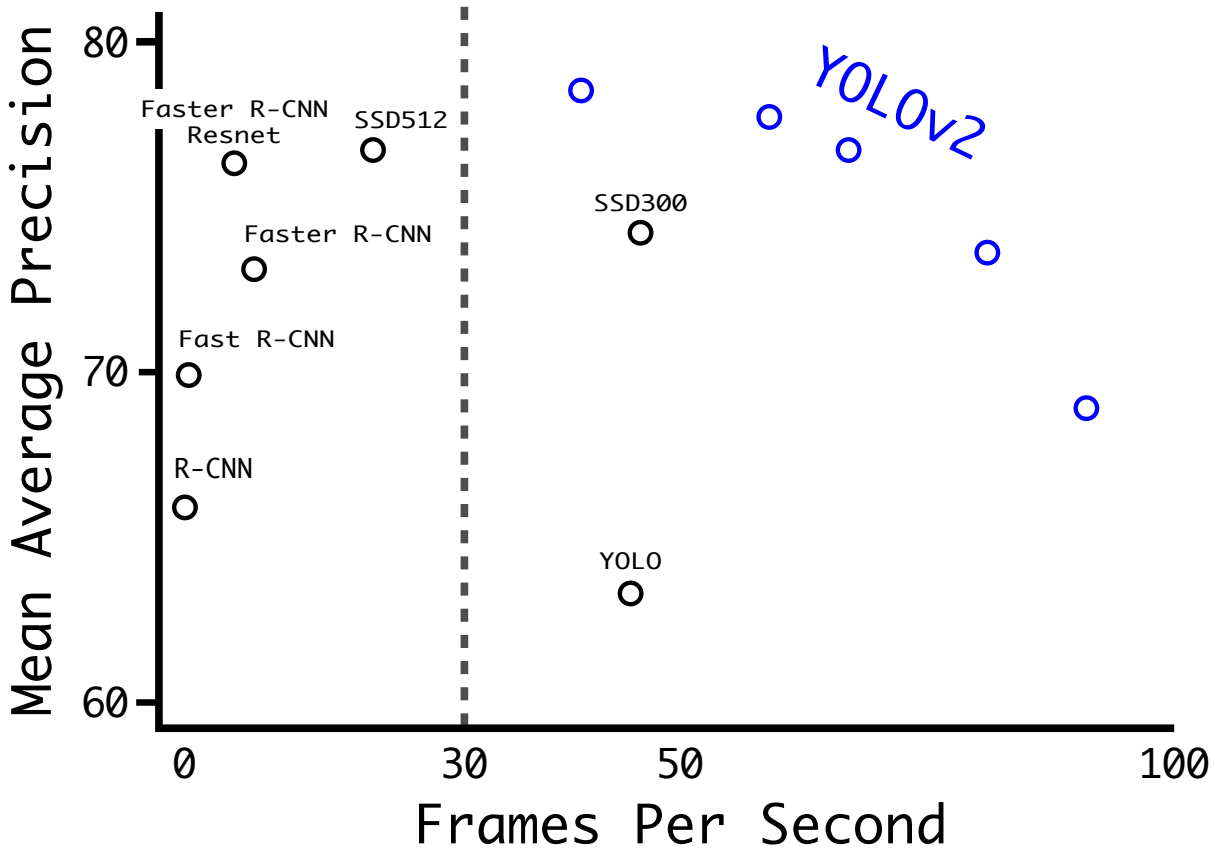


Figure 2.4: Accuracy and speed on VOC 2007.

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Table 2.6: Darknet-19.

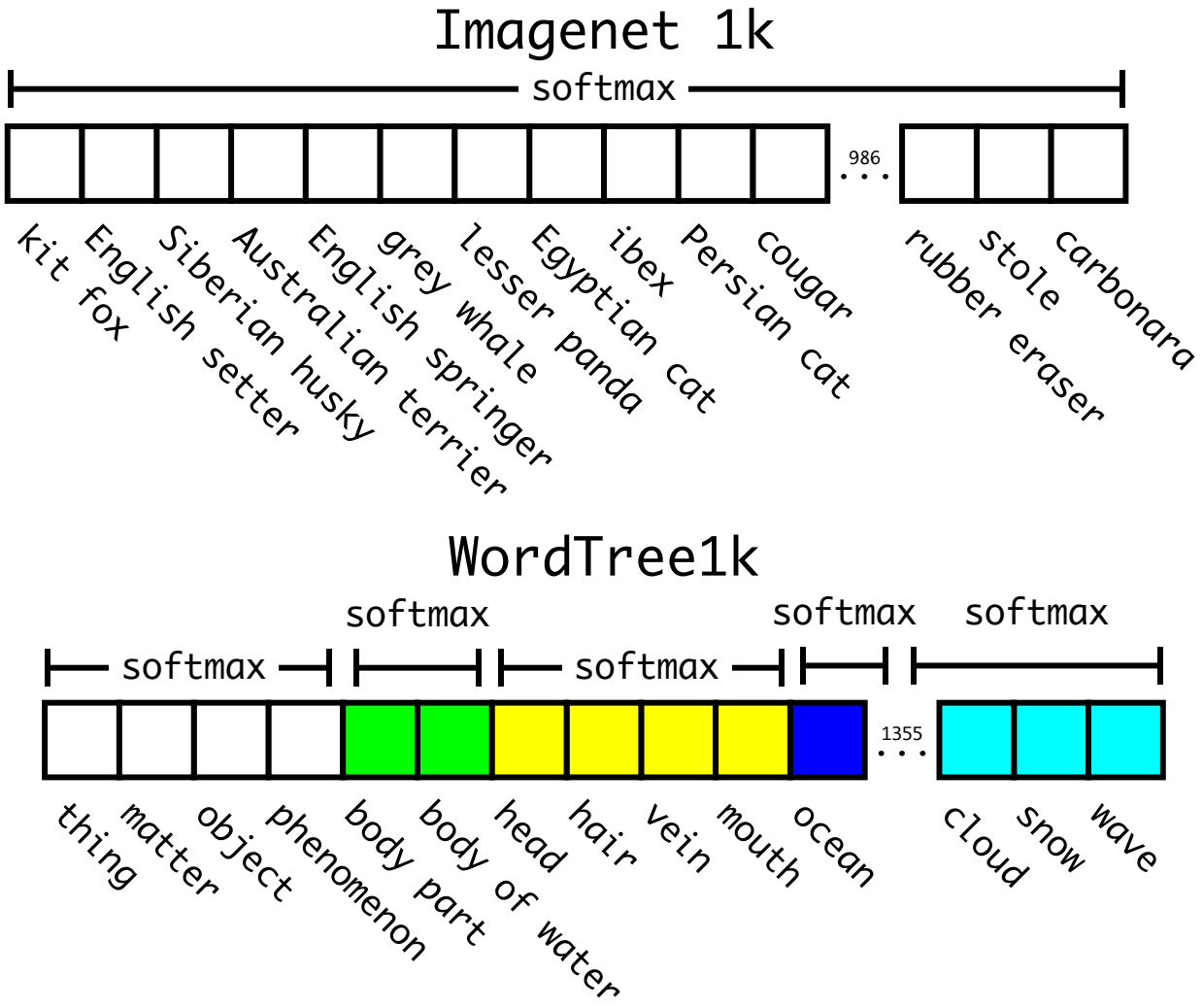


Figure 2.5: Prediction on ImageNet vs WordTree. Most ImageNet models use one large softmax to predict a probability distribution. Using WordTree we perform multiple softmax operations over co-hyponyms.

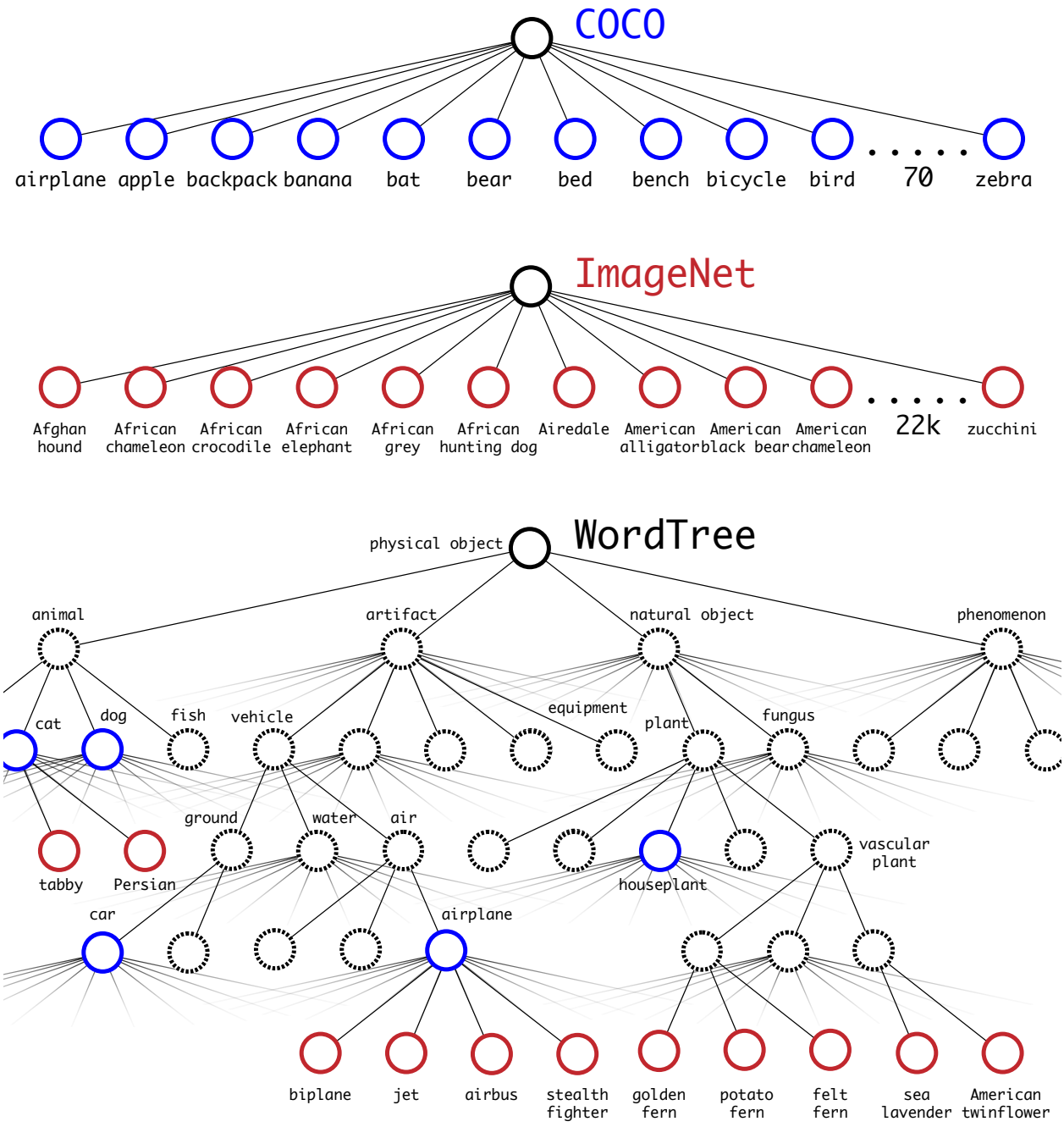


Figure 2.6: Combining datasets using WordTree hierarchy. Using the WordNet concept graph we build a hierarchical tree of visual concepts. Then we can merge datasets together by mapping the classes in the dataset to synsets in the tree. This is a simplified view of WordTree for illustration purposes.

diaper	0.0
horizontal bar	0.0
rubber eraser	0.0
sunglasses	0.0
swimming trunks	0.0
...	
red panda	50.7
fox	52.1
koala bear	54.3
tiger	61.0
armadillo	61.7

Table 2.7: YOLO9000 Best and Worst Classes on ImageNet. The classes with the highest and lowest AP from the 156 weakly supervised classes. YOLO9000 learns good models for a variety of animals but struggles with new classes like clothing or equipment.

Chapter 3

YOLOV3: AN INCREMENTAL IMPROVEMENT

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At 320×320 YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP_{50} in 51 ms on a Titan X, compared to 57.5 AP_{50} in 198 ms by RetinaNet, similar performance but $3.8\times$ faster. As always, all the code is online at https://pjreddie.com/yolo/.

3.1 Introduction

Sometimes you just kinda phone it in for a year, you know? I didn't do a whole lot of research this year. Spent a lot of time on Twitter. Played around with GANs a little. I had a little momentum left over from last year [88] [1]; I managed to make some improvements to YOLO. But, honestly, nothing like super interesting, just a bunch of small changes that make it better. I also helped out with other people's research a little.

Actually, that's what brings us here today. We have a camera-ready deadline [44] and we need to cite some of the random updates I made to YOLO but we don't have a source. So get ready for a TECH REPORT!

The great thing about tech reports is that they don't need intros, y'all know why we're here. So the end of this introduction will signpost for the rest of the paper. First we'll tell you what the deal is with YOLOv3. Then we'll tell you how we do. We'll also tell you about some things we tried that didn't work. Finally we'll contemplate what this all means.

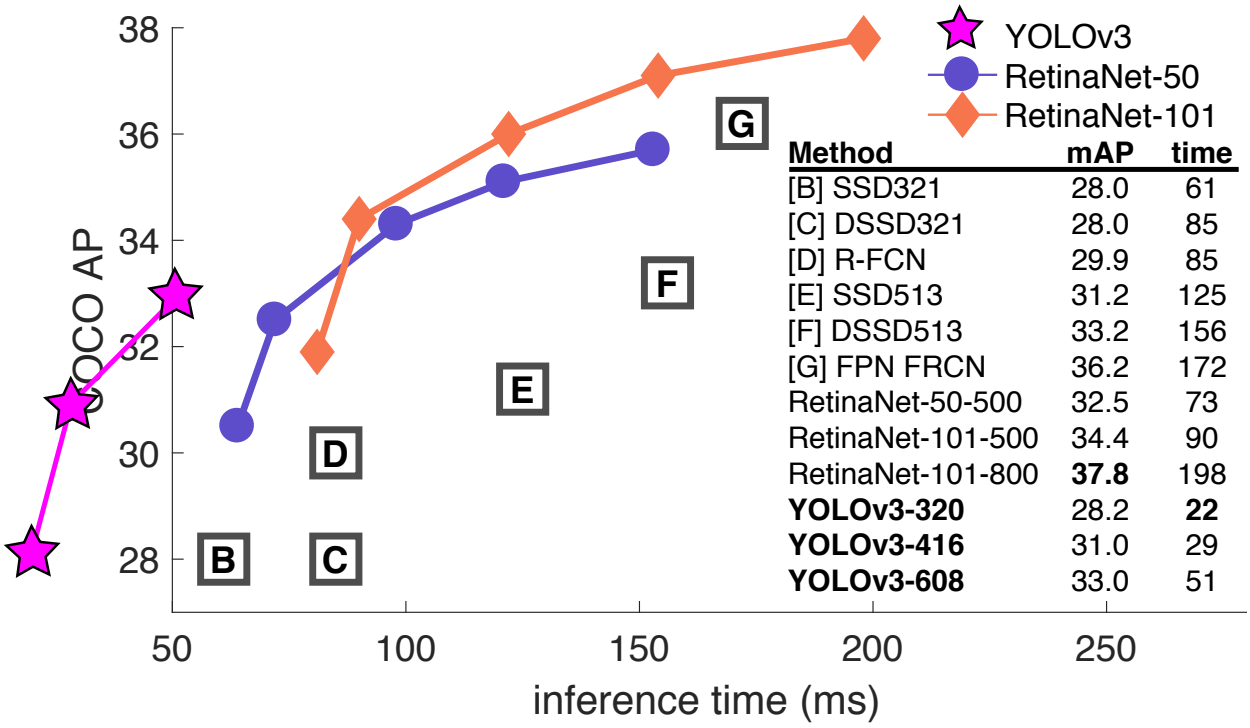


Figure 3.1: We adapt this figure from the Focal Loss paper [78]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

3.2 The Deal

So here's the deal with YOLOv3: We mostly took good ideas from other people. We also trained a new classifier network that's better than the other ones. We'll just take you through the whole system from scratch so you can understand it all.

3.2.1 Bounding Box Prediction

Following YOLO9000 our system predicts bounding boxes using dimension clusters as anchor boxes [100]. The network predicts 4 coordinates for each bounding box, t_x, t_y, t_w, t_h. If the cell is offset from the top left corner of the image by (c_x, c_y) and the bounding box prior has width and height p_w, p_h, then the predictions correspond to:

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

During training we use sum of squared error loss. If the ground truth for some coordinate prediction is $\hat{t}_*$ our gradient is the ground truth value (computed from the ground truth box) minus our prediction: $\hat{t}_* - t_*$. This ground truth value can be easily computed by inverting the equations above.

YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold we ignore the prediction, following [102]. We use the threshold of .5. Unlike [102] our system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness.

3.2.2 Class Prediction

Each box predicts the classes the bounding box may contain using multilabel classification. We do not use a softmax as we have found it is unnecessary for good performance, instead we simply use independent logistic classifiers. During training we use binary cross-entropy loss for the class predictions.

This formulation helps when we move to more complex domains like the Open Images Dataset [68]. In this dataset there are many overlapping labels (i.e. Woman and Person). Using a softmax imposes the assumption that each box has exactly one class which is often not the case. A multilabel approach better models the data.

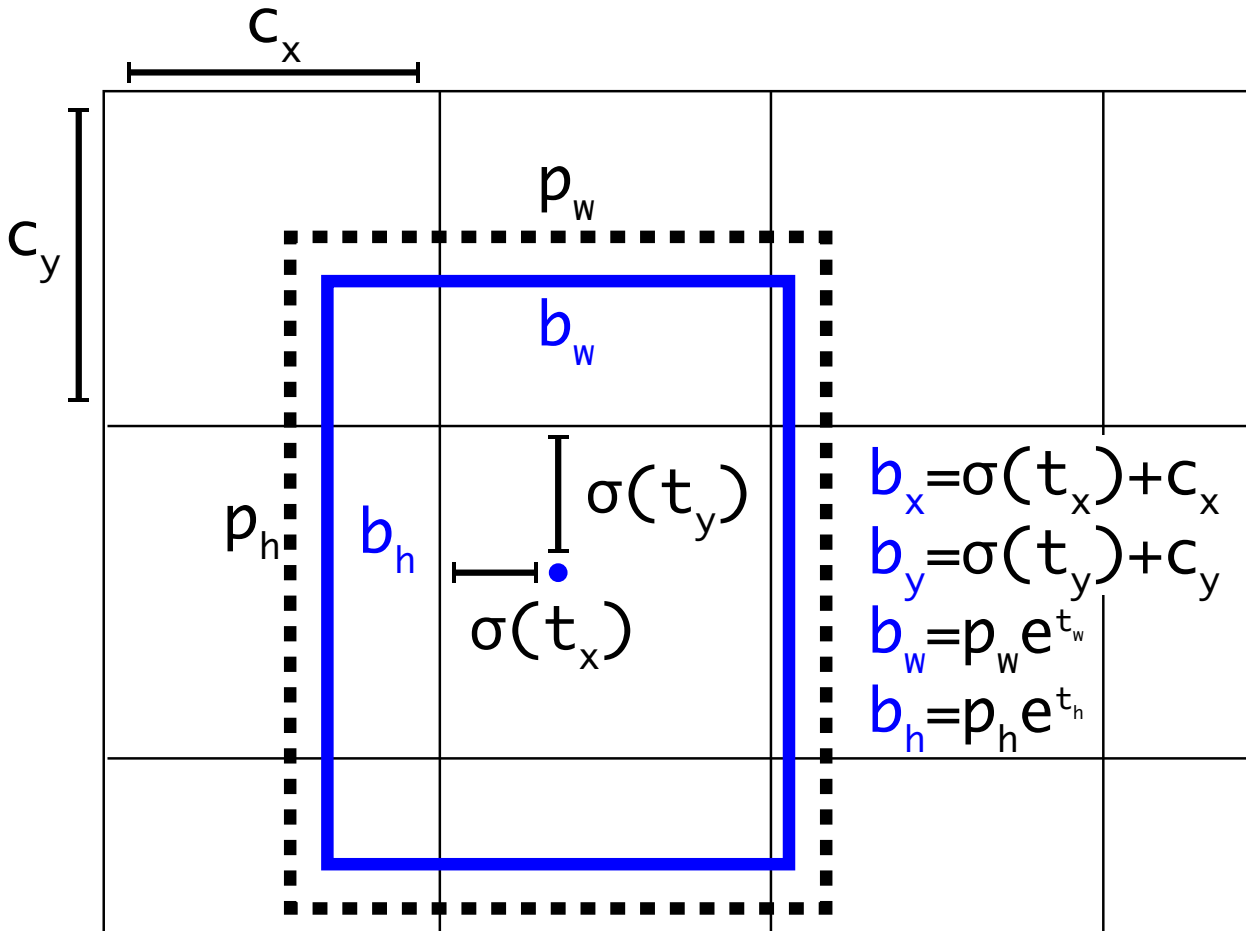


Figure 3.2: Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [100].

3.2.3 Predictions Across Scales

YOLOv3 predicts boxes at 3 different scales. Our system extracts features from those scales using a similar concept to feature pyramid networks [77]. From our base feature extractor we add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. In our experiments with COCO [80] we predict 3 boxes at each scale so the tensor is $N \times N \times [3 * (4 + 1 + 80)]$ for the 4 bounding box offsets, 1 objectness prediction, and 80 class predictions.

Next we take the feature map from 2 layers previous and upsample it by $2\times$. We also take a feature map from earlier in the network and merge it with our upsampled features using concatenation. This method allows us to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. We then add a few more convolutional layers to process this combined feature map, and eventually predict a similar tensor, although now twice the size.

We perform the same design one more time to predict boxes for the final scale. Thus our predictions for the 3rd scale benefit from all the prior computation as well as fine-grained features from early on in the network.

We still use k-means clustering to determine our bounding box priors. We just sort of chose 9 clusters and 3 scales arbitrarily and then divide up the clusters evenly across scales. On the COCO dataset the 9 clusters were: $(10 \times 13), (16 \times 30), (33 \times 23), (30 \times 61), (62 \times 45), (59 \times 119), (116 \times 90), (156 \times 198), (373 \times 326)$.

3.2.4 Feature Extractor

We use a new network for performing feature extraction. Our new network is a hybrid approach between the network used in YOLOv2, Darknet-19, and that newfangled residual network stuff. Our network uses successive 3×3 and 1×1 convolutional layers but now has some shortcut connections as well and is significantly larger. It has 53 convolutional layers so we call it.... wait for it..... Darknet-53!

This new network is much more powerful than Darknet-19 but still more efficient than ResNet-101 or ResNet-152. Here are some ImageNet results:

Each network is trained with identical settings and tested at 256×256, single crop accuracy. Run times are measured on a Titan X at 256×256. Thus Darknet-53 performs on par with state-of-the-art classifiers but with fewer floating point operations and more speed. Darknet-53 is better than ResNet-101 and $1.5\times$ faster. Darknet-53 has similar performance to ResNet-152 and is $2\times$ faster.

Darknet-53 also achieves the highest measured floating point operations per second. This means the network structure better utilizes the GPU, making it more efficient to evaluate and thus faster. That's mostly because ResNets have just way too many layers and aren't very efficient.

3.2.5 Training

We still train on full images with no hard negative mining or any of that stuff. We use multi-scale training, lots of data augmentation, batch normalization, all the standard stuff. We use the Darknet neural network framework for training and testing [96].

3.3 How We Do

YOLOv3 is pretty good! See table 3.3. In terms of COCOs weird average mean AP metric it is on par with the SSD variants but is $3\times$ faster. It is still quite a bit behind other models like RetinaNet in this metric though.

However, when we look at the "old" detection metric of mAP at IOU$= .5$ (or AP_{50} in the chart) YOLOv3 is very strong. It is almost on par with RetinaNet and far above the SSD variants. This indicates that YOLOv3 is a very strong detector that excels at producing decent boxes for objects. However, performance drops significantly as the IOU threshold increases indicating YOLOv3 struggles to get the boxes perfectly aligned with the object.

In the past YOLO struggled with small objects. However, now we see a reversal in that trend. With the new multi-scale predictions we see YOLOv3 has relatively high AP_S performance. How-

ever, it has comparatively worse performance on medium and larger size objects. More investigation is needed to get to the bottom of this.

When we plot accuracy vs speed on the AP_{50} metric (see figure 3.5) we see YOLOv3 has significant benefits over other detection systems. Namely, it's faster and better.

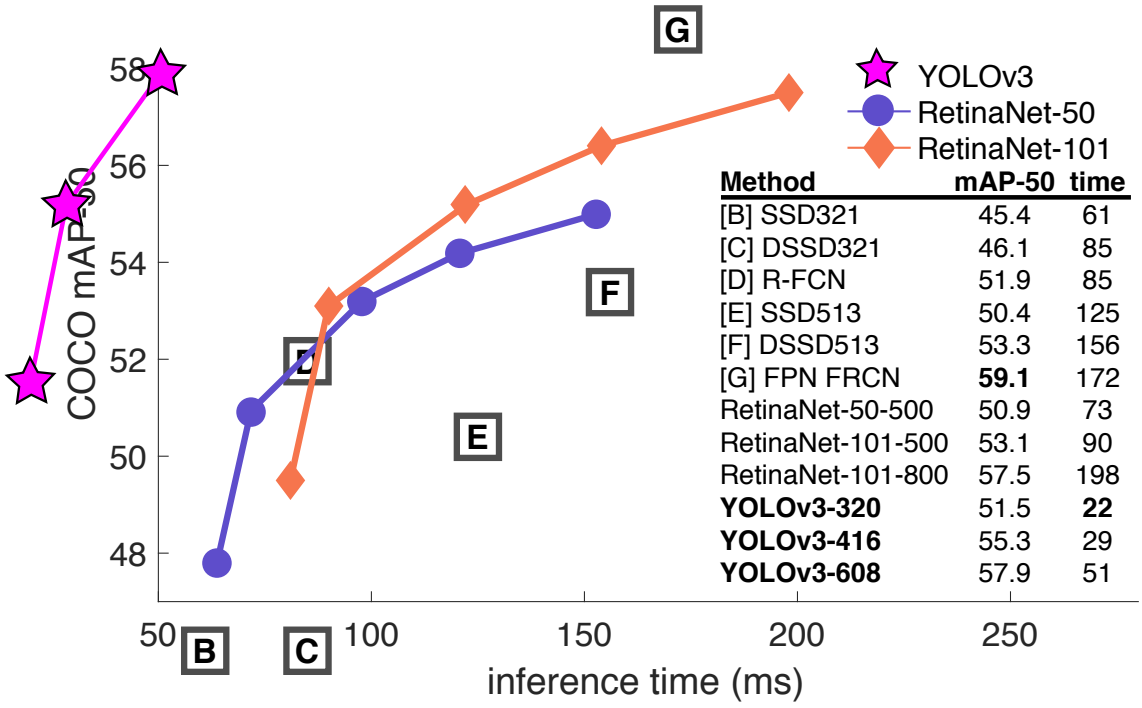


Figure 3.3: Again adapted from the [78], this time displaying speed/accuracy tradeoff on the mAP at .5 IOU metric. You can tell YOLOv3 is good because it's very high and far to the left. Can you cite your own paper? Guess who's going to try, this guy → [101]. Oh, I forgot, we also fix a data loading bug in YOLOv2, that helped by like 2 mAP. Just sneaking this in here to not throw off layout.

3.4 Things We Tried That Didn't Work

We tried lots of stuff while we were working on YOLOv3. A lot of it didn't work. Here's the stuff we can remember.

Anchor box x, y offset predictions. We tried using the normal anchor box prediction mechanism where you predict the x, y offset as a multiple of the box width or height using a linear activation. We found this formulation decreased model stability and didn't work very well.

Linear x, y predictions instead of logistic. We tried using a linear activation to directly predict the x, y offset instead of the logistic activation. This led to a couple point drop in mAP.

Focal loss. We tried using focal loss. It dropped our mAP about 2 points. YOLOv3 may already be robust to the problem focal loss is trying to solve because it has separate objectness predictions and conditional class predictions. Thus for most examples there is no loss from the class predictions? Or something? We aren't totally sure.

Dual IOU thresholds and truth assignment. Faster R-CNN uses two IOU thresholds during training. If a prediction overlaps the ground truth by .7 it is as a positive example, by $[.3 - .7]$ it is ignored, less than .3 for all ground truth objects it is a negative example. We tried a similar strategy but couldn't get good results.

We quite like our current formulation, it seems to be at a local optima at least. It is possible that some of these techniques could eventually produce good results, perhaps they just need some tuning to stabilize the training.

3.5 What This All Means

YOLOv3 is a good detector. It's fast, it's accurate. It's not as great on the COCO average AP between .5 and .95 IOU metric. But it's very good on the old detection metric of .5 IOU.

Why did we switch metrics anyway? The original COCO paper just has this cryptic sentence: "A full discussion of evaluation metrics will be added once the evaluation server is complete". Russakovsky et al report that that humans have a hard time distinguishing an IOU of .3 from .5! "Training humans to visually inspect a bounding box with IOU of 0.3 and distinguish it from one with IOU 0.5 is surprisingly difficult." [105] If humans have a hard time telling the difference, how much does it matter?

But maybe a better question is: "What are we going to do with these detectors now that we have them?" A lot of the people doing this research are at Google and Facebook. I guess at least

we know the technology is in good hands and definitely won't be used to harvest your personal information and sell it to.... wait, you're saying that's exactly what it will be used for?? Oh.

Well the other people heavily funding vision research are the military and they've never done anything horrible like killing lots of people with new technology oh wait.....[1]

I have a lot of hope that most of the people using computer vision are just doing happy, good stuff with it, like counting the number of zebras in a national park [93], or tracking their cat as it wanders around their house [107]. But computer vision is already being put to questionable use and as researchers we have a responsibility to at least consider the harm our work might be doing and think of ways to mitigate it. We owe the world that much.

In closing, do not @ me. (Because I finally quit Twitter).

[1]The author is funded by the Office of Naval Research and Google.

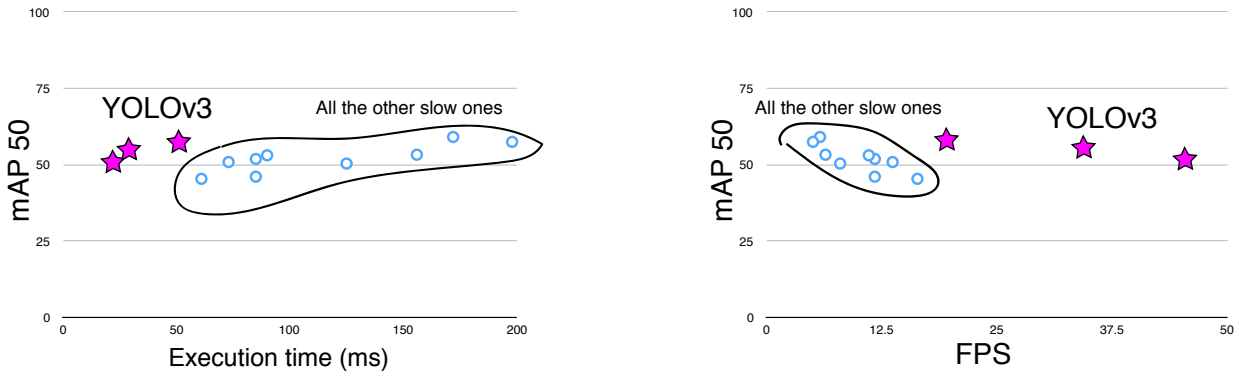


Figure 3.4: Zero-axis charts are probably more intellectually honest... and we can still screw with the variables to make ourselves look good!

Rebuttal

We would like to thank the Reddit commenters, labmates, emailers, and passing shouts in the hallway for their lovely, heartfelt words. If you, like me, are reviewing for ICCV then we know you probably have 37 other papers you could be reading that you'll invariably put off until the last week and then have some legend in the field email you about how you really should finish those reviews execept it won't entirely be clear what they're saying and maybe they're from the future? [34] Anyway, this paper won't have become what it will in time be without all the work your past selves will have done also in the past but only a little bit further forward, not like all the way until now forward. And if you tweeted about it I wouldn't know. Just sayin.

Reviewer #2 AKA Dan Grossman (lol blinding who does that) insists that I point out here that our graphs have not one but two non-zero origins. You're absolutely right Dan, that's because it looks way better than admitting to ourselves that we're all just here battling over 2-3% mAP. But here are the requested graphs. I threw in one with FPS too because we look just like super good when we plot on FPS.

Reviewer #4 AKA JudasAdventus on Reddit writes "Entertaining read but the arguments against the MSCOCO metrics seem a bit weak". Well, I always knew you would be the one to turn on me Judas. You know how when you work on a project and it only comes out alright so you have to figure out some way to justify how what you did actually was pretty cool? I was basically trying to do that and I lashed out at the COCO metrics a little bit. But now that I've staked out this hill I may as well die on it.

See here's the thing, mAP is already sort of broken so an update to it should maybe address some of the issues with it or at least justify why the updated version is better in some way. And that's the big thing I took issue with was the lack of justification. For PASCAL VOC, the IOU threshold was "set deliberately low to account for inaccuracies in bounding boxes in the ground truth data" [31]. Does COCO have better labelling than VOC? This is definitely possible since COCO has segmentation masks maybe the labels are more trustworthy and thus we aren't as worried about inaccuracy. But again, my problem was the lack of justification.

The COCO metric emphasizes better bounding boxes but that emphasis must mean it de-emphasizes something else, in this case classification accuracy. Is there a good reason to think that more precise bounding boxes are more important than better classification? A miss-classified example is much more obvious than a bounding box that is slightly shifted.

mAP is already screwed up because all that matters is per-class rank ordering. For example, if your test set only has these two images then according to mAP two detectors that produce these results are JUST AS GOOD:

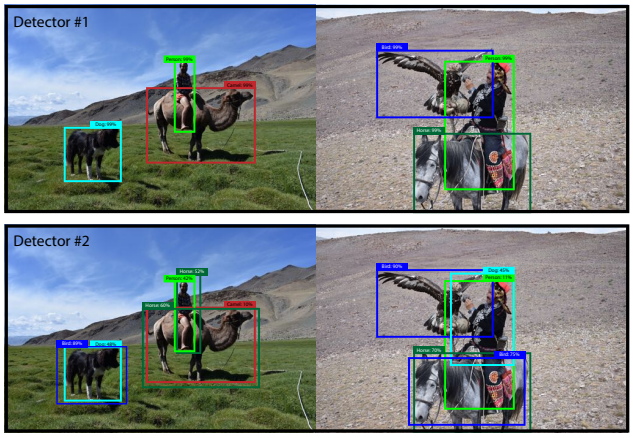


Figure 3.5: These two hypothetical detectors are perfect according to mAP over these two images. They are both perfect. Totally equal.

Now this is OBVIOUSLY an over-exaggeration of the problems with mAP but I guess my newly retconned point is that there are such obvious discrepancies between what people in the "real world" would care about and our current metrics that I think if we're going to come up with new metrics we should focus

on these discrepancies. Also, like, it's already mean average precision, what do we even call the COCO metric, average mean average precision?

Here's a proposal, what people actually care about is given an image and a detector, how well will the detector find and classify objects in the image. What about getting rid of the per-class AP and just doing a global average precision? Or doing an AP calculation per-image and averaging over that?

Boxes are stupid anyway though, I'm probably a true believer in masks except I can't get YOLO to learn them.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
	Convolutional	32	1 × 1	
1×	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
	Convolutional	64	1 × 1	
2×	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
	Convolutional	128	1 × 1	
8×	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
8×	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
4×	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 3.1: Darknet-53.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [100]	74.1	91.8	7.29	1246	**171**
ResNet-101[55]	77.1	93.7	19.7	1039	53
ResNet-152 [55]	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Table 3.2: Comparison of backbones. Accuracy, billions of operations, billion floating point operations per second, and FPS for various networks.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [55]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [77]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [59]	Inception-ResNet-v2 [118]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [111]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [100]	DarkNet-19 [100]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [82, 35]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [35]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [78]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [78]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608×608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Table 3.3: I'm seriously just stealing all these tables from [78] they take soooo long to make from scratch. Ok, YOLOv3 is doing alright. Keep in mind that RetinaNet has like $3.8\times$ longer to process an image. YOLOv3 is much better than SSD variants and comparable to state-of-the-art models on the AP_{50} metric.

Chapter 4

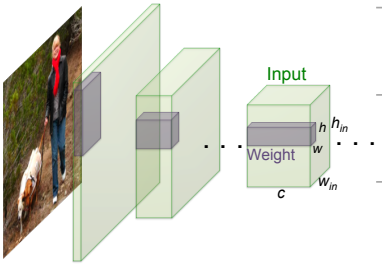
XNOR-NET: BINARY CONVOLUTIONAL NEURAL NETWORKS

We propose two efficient approximations to standard convolutional neural networks: Binary-Weight-Networks and XNOR-Networks. In Binary-Weight-Networks, the filters are approximated with binary values resulting in $32\times$ memory saving. In XNOR-Networks, both the filters and the input to convolutional layers are binary. XNOR-Networks approximate convolutions using primarily binary operations. This results in $58\times$ faster convolutional operations (in terms of number of the high precision operations) and $32\times$ memory savings. XNOR-Nets offer the possibility of running state-of-the-art networks on CPUs (rather than GPUs) in real-time. Our binary networks are simple, accurate, efficient, and work on challenging visual tasks. We evaluate our approach on the ImageNet classification task. The classification accuracy with a Binary-Weight-Network version of AlexNet is the same as the full-precision AlexNet. We compare our method with recent network binarization methods, BinaryConnect and BinaryNets, and outperform these methods by large margins on ImageNet, more than 16% in top-1 accuracy. Our code is available at: http://allenai.org/plato/xnornet.

4.1 Introduction

Deep neural networks (DNN) have shown significant improvements in several application domains including computer vision and speech recognition. In computer vision, a particular type of DNN, known as Convolutional Neural Networks (CNN), have demonstrated state-of-the-art results in object recognition [70, 114, 121, 54] and detection [40, 38, 102].

Convolutional neural networks show reliable results on object recognition and detection that are useful in real world applications. Concurrent to the recent progress in recognition, interesting advancements have been happening in virtual reality (VR by Oculus) [90], augmented reality (AR by HoloLens) [45], and smart wearable devices. Putting these two pieces together, we argue that it is the right time to equip smart portable devices with the power of state-of-the-art recognition systems. However, CNN-based recognition systems need large amounts of memory and computational power. While they perform well on expensive, GPU-



	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs; Real-Value Weights	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs; Binary Weights	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs; Binary Weights	XNOR , bitcount	~32x	~58x	%44.2

Figure 4.1: We propose two efficient variations of convolutional neural networks. **Binary-Weight-Networks**, when the weight filters contains binary values. **XNOR-Networks**, when both weigh and input have binary values. These networks are very efficient in terms of memory and computation, while being very accurate in natural image classification. This offers the possibility of using accurate vision techniques in portable devices with limited resources.

based machines, they are often unsuitable for smaller devices like cell phones and embedded electronics.

For example, AlexNet[70] has 61M parameters (249MB of memory) and performs 1.5B high precision operations to classify one image. These numbers are even higher for deeper CNNs *e.g.,*VGG [114] (see section 4.4.1). These models quickly overtax the limited storage, battery power, and compute capabilities of smaller devices like cell phones.

In this paper, we introduce simple, efficient, and accurate approximations to CNNs by binarizing the weights and even the intermediate representations in convolutional neural networks. Our binarization method aims at finding the best approximations of the convolutions using binary operations. We demonstrate that our way of binarizing neural networks results in ImageNet classification accuracy numbers that are comparable to standard full precision networks while requiring a significantly less memory and fewer floating point operations.

We study two approximations: Neural networks with binary weights and XNOR-Networks. In **Binary-Weight-Networks** all the weight values are approximated with binary values. A convolutional neural network with binary weights is significantly smaller ($\sim 32\times$) than an equivalent network with single-precision weight values. In addition, when weight values are binary, convolutions can be estimated by only addition

and subtraction (without multiplication), resulting in $\sim 2\times$ speed up. Binary-weight approximations of large CNNs can fit into the memory of even small, portable devices while maintaining the same level of accuracy (See Section 4.4.1 and 4.4.2).

To take this idea further, we introduce **XNOR-Networks** where both the weights and the inputs to the convolutional and fully connected layers are approximated with binary values[1]. Binary weights and binary inputs allow an efficient way of implementing convolutional operations. If all of the operands of the convolutions are binary, then the convolutions can be estimated by XNOR and bitcounting operations [14]. XNOR-Nets result in accurate approximation of CNNs while offering $\sim 58\times$ speed up in CPUs (in terms of number of the high precision operations). This means that XNOR-Nets can enable real-time inference in devices with small memory and no GPUs (Inference in XNOR-Nets can be done very efficiently on CPUs).

To the best of our knowledge this paper is the first attempt to present an evaluation of binary neural networks on large-scale datasets like ImageNet. Our experimental results show that our proposed method for binarizing convolutional neural networks outperforms the state-of-the-art network binarization method of [14] by a large margin (16.3%) on top-1 image classification in the ImageNet challenge ILSVRC2012. Our contribution is two-fold: First, we introduce a new way of binarizing the weight values in convolutional neural networks and show the advantage of our solution compared to state-of-the-art solutions. Second, we introduce XNOR-Nets, a deep neural network model with binary weights and binary inputs and show that XNOR-Nets can obtain similar classification accuracies compared to standard networks while being significantly more efficient. Our code is available at: http://allenai.org/plato/xnornet

4.2 Related Work

Deep neural networks often suffer from over-parametrization and large amounts of redundancy in their models. This typically results in inefficient computation and memory usage[23]. Several methods have been proposed to address efficient training and inference in deep neural networks.

Shallow networks: Estimating a deep neural network with a shallower model reduces the size of a network. Early theoretical work by Cybenko shows that a network with a large enough single hidden layer of sigmoid units can approximate any decision boundary [17]. In several areas (*e.g.,*vision and speech), however, shallow networks cannot compete with deep models [108]. [19] trains a shallow network on SIFT

[1]fully connected layers can be implemented by convolution, therefore, in the rest of the paper, we refer to them also as convolutional layers [84].

features to classify the ImageNet dataset. They show it is difficult to train shallow networks with large number of parameters. [4] provides empirical evidence on small datasets (*e.g.,*CIFAR-10) that shallow nets are capable of learning the same functions as deep nets. In order to get the similar accuracy, the number of parameters in the shallow network must be close to the number of parameters in the deep network. They do this by first training a state-of-the-art deep model, and then training a shallow model to mimic the deep model. These methods are different from our approach because we use the standard deep architectures not the shallow estimations.

Compressing pre-trained deep networks: Pruning redundant, non-informative weights in a previously trained network reduces the size of the network at inference time. Weight decay [49] was an early method for pruning a network. Optimal Brain Damage [72] and Optimal Brain Surgeon [51] use the Hessian of the loss function to prune a network by reducing the number of connections. Recently [48] reduced the number of parameters by an order of magnitude in several state-of-the-art neural networks by pruning. [124] proposed to reduce the number of activations for compression and acceleration. Deep compression [47] reduces the storage and energy required to run inference on large networks so they can be deployed on mobile devices. They remove the redundant connections and quantize weights so that multiple connections share the same weight, and then they use Huffman coding to compress the weights. HashedNets [11] uses a hash function to reduce model size by randomly grouping the weights, such that connections in a hash bucket use a single parameter value. Matrix factorization has been used by [24, 64]. We are different from these approaches because we do not use a pretrained network. We train binary networks from scratch.

Designing compact layers: Designing compact blocks at each layer of a deep network can help to save memory and computational costs. Replacing the fully connected layer with global average pooling was examined in the Network in Network architecture [76], GoogLenet[121] and Residual-Net[54], which achieved state-of-the-art results on several benchmarks. The bottleneck structure in Residual-Net [54] has been proposed to reduce the number of parameters and improve speed. Decomposing 3×3 convolutions with two 1×1 is used in [117] and resulted in state-of-the-art performance on object recognition. Replacing 3×3 convolutions with 1×1 convolutions is used in [61] to create a very compact neural network that can achieve $\sim 50\times$ reduction in the number of parameters while obtaining high accuracy. Our method is different from this line of work because we use the full network (not the compact version) but with binary parameters.

Quantizing parameters: High precision parameters are not very important in achieving high performance in deep networks. [43] proposed to quantize the weights of fully connected layers in a deep network by vector quantization techniques. They showed just thresholding the weight values at zero only decreases the top-1 accuracy on ILSVRC2012 by less than $\%10$. [3] proposed a provably polynomial time algorithm for training a sparse networks with +1/0/-1 weights. A fixed-point implementation of 8-bit integer was compared with 32-bit floating point activations in [125]. Another fixed-point network with ternary weights and 3-bits activations was presented by [60]. Quantizing a network with L_2 error minimization achieved better accuracy on MNIST and CIFAR-10 datasets in [2]. [81] proposed a back-propagation process by quantizing the representations at each layer of the network. To convert some of the remaining multiplications into binary shifts the neurons get restricted values of power-of-two integers. In [81] they carry the full precision weights during the test phase, and only quantize the neurons during the back-propagation process, and not during the forward-propagation. Our work is similar to these methods since we are quantizing the parameters in the network. But our quantization is the extreme scenario +1,-1.

Network binarization: These works are the most related to our approach. Several methods attempt to binarize the weights and the activations in neural networks.The performance of highly quantized networks (*e.g.,*binarized) were believed to be very poor due to the destructive property of binary quantization [15]. Expectation BackPropagation (EBP) in [115] showed high performance can be achieved by a network with binary weights and binary activations. This is done by a variational Bayesian approach, that infers networks with binary weights and neurons. A fully binary network at run time presented in [28] using a similar approach to EBP, showing significant improvement in energy efficiency. In EBP the binarized parameters were only used during inference. BinaryConnect [16] extended the probablistic idea behind EBP. Similar to our approach, BinaryConnect uses the real-valued version of the weights as a key reference for the binarization process. The real-valued weight updated using the back propagated error by simply ignoring the binarization in the update. BinaryConnect achieved state-of-the-art results on small datasets (*e.g.,*CIFAR-10, SVHN). Our experiments shows that this method is not very successful on large-scale datsets (*e.g.,*ImageNet). BinaryNet[14] propose an extention of BinaryConnect, where both weights and activations are binarized. Our method is different from them in the binarization method and the network structure. We also compare our method with BinaryNet on ImageNet, and our method outperforms BinaryNet by a large margin.[128] argued that the noise introduced by weight binarization provides a form of

regularization, which could help to improve test accuracy. This method binarizes weights while maintaining full precision activation. [5] proposed fully binary training and testing in an array of committee machines with randomized input. [66] retraine a previously trained neural network with binary weights and binary inputs.

4.3 Binary Convolutional Neural Network

We represent an L-layer CNN architecture with a triplet $\langle \mathcal{I}, \mathcal{W}, * \rangle$. $\mathcal{I}$ is a set of tensors, where each element $\mathbf{I} = \mathcal{I}_{l(l=1,...,L)}$ is the input tensor for the l^{th} layer of CNN (Green cubes in figure 4.1). $\mathcal{W}$ is a set of tensors, where each element in this set $\mathbf{W} = \mathcal{W}_{lk(k=1,...,K^l)}$ is the k^{th} weight filter in the l^{th} layer of the CNN. K^l is the number of weight filters in the l^{th} layer of the CNN. $*$ represents a convolutional operation with $\mathbf{I}$ and $\mathbf{W}$ as its operands[2]. $\mathbf{I} \in \mathbb{R}^{c \times w_{in} \times h_{in}}$, where (c, w_{in}, h_{in}) represents *channels, width* and *height* respectively.$\mathbf{W} \in \mathbb{R}^{c \times w \times h}$, where $w \leq w_{in}, \;\; h \leq h_{in}$. We propose two variations of binary CNN: **Binary-weights**, where the elements of $\mathcal{W}$ are binary tensors and **XNOR-Networks**, where elements of both $\mathcal{I}$ and $\mathcal{W}$ are binary tensors.

4.3.1 Binary-Weight-Networks

In order to constrain a convolutional neural network $\langle \mathcal{I}, \mathcal{W}, * \rangle$ to have binary weights, we estimate the real-value weight filter $\mathbf{W} \in \mathcal{W}$ using a binary filter $\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$ and a scaling factor $\alpha \in \mathbb{R}^+$ such that $\mathbf{W} \approx \alpha \mathbf{B}$. A convolutional operation can be appriximated by:

$$\mathbf{I} * \mathbf{W} \approx (\mathbf{I} \oplus \mathbf{B})\, \alpha \tag{4.1}$$

where, $\oplus$ indicates a convolution without any multiplication. Since the weight values are binary, we can implement the convolution with additions and subtractions. The binary weight filters reduce memory usage by a factor of $\sim 32\times$ compared to single-precision filters. We represent a CNN with binary weights by $\langle \mathcal{I}, \mathcal{B}, \mathcal{A}, \oplus \rangle$, where $\mathcal{B}$ is a set of binary tensors and $\mathcal{A}$ is a set of positive real scalars, such that $\mathbf{B} = \mathcal{B}_{lk}$ is a binary filter and $\alpha = \mathcal{A}_{lk}$ is an scaling factor and $\mathcal{W}_{lk} \approx \mathcal{A}_{lk}\mathcal{B}_{lk}$

[2]In this paper we assume convolutional filters do not have bias terms

Estimating binary weights:

Without loss of generality we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$. To find an optimal estimation for $\mathbf{W} \approx \alpha\mathbf{B}$, we solve the following optimization:

$$\begin{aligned} J(\mathbf{B}, \alpha) &= \|\mathbf{W} - \alpha\mathbf{B}\|^2 \\ \alpha^*, \mathbf{B}^* &= \underset{\alpha, \mathbf{B}}{\operatorname{argmin}} J(\mathbf{B}, \alpha) \end{aligned} \tag{4.2}$$

by expanding equation 4.2, we have

$$J(\mathbf{B}, \alpha) = \alpha^2\mathbf{B}^\mathsf{T}\mathbf{B} - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{W}^\mathsf{T}\mathbf{W} \tag{4.3}$$

since $\mathbf{B} \in \{+1, -1\}^n$, $\mathbf{B}^\mathsf{T}\mathbf{B} = n$ is a constant . $\mathbf{W}^\mathsf{T}\mathbf{W}$ is also a constant because $\mathbf{W}$ is a known variable. Lets define $\mathbf{c} = \mathbf{W}^\mathsf{T}\mathbf{W}$. Now, we can rewrite the equation 4.3 as follow: $J(\mathbf{B}, \alpha) = \alpha^2 n - 2\alpha\mathbf{W}^\mathsf{T}\mathbf{B} + \mathbf{c}$. The optimal solution for $\mathbf{B}$ can be achieved by maximizing the following constrained optimization: (note that α is a positive value in equation 4.2, therefore it can be ignored in the maximization)

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}}\{\mathbf{W}^\mathsf{T}\mathbf{B}\} \quad s.t. \;\; \mathbf{B} \in \{+1, -1\}^n \tag{4.4}$$

This optimization can be solved by assigning $\mathbf{B}_i = +1$ if $\mathbf{W}_i \geq 0$ and $\mathbf{B}_i = -1$ if $\mathbf{W}_i < 0$, therefore the optimal solution is $\mathbf{B}^* = \operatorname{sign}(\mathbf{W})$. In order to find the optimal value for the scaling factor α^*, we take the derivative of J with respect to α and set it to zero:

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T}\mathbf{B}^*}{n} \tag{4.5}$$

By replacing $\mathbf{B}^*$ with $\operatorname{sign}(\mathbf{W})$

$$\alpha^* = \frac{\mathbf{W}^\mathsf{T}\operatorname{sign}(\mathbf{W})}{n} = \frac{\sum|\mathbf{W}_i|}{n} = \frac{1}{n}\|\mathbf{W}\|_{\ell 1} \tag{4.6}$$

therefore, the optimal estimation of a binary weight filter can be simply achieved by taking the sign of weight values. The optimal scaling factor is the average of absolute weight values.

Training Binary-Weights-Networks:

Each iteration of training a CNN involves three steps; forward pass, backward pass and parameters update. To train a CNN with binary weights (in convolutional layers), we only binarize the weights during

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

Binarizing weight filters:
for $l = 1$ to L **do**
  for k^{th} filter in l^{th} layer **do**
    $\mathcal{A}_{lk} = \frac{1}{n}\|\mathcal{W}^t_{lk}\|_{\ell 1}$
    $\mathcal{B}_{lk} = \text{sign}(\mathcal{W}^t_{lk})$
    $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$
$\hat{\mathbf{Y}} =$ **BinaryForward**$(\mathbf{I}, \mathcal{B}, \mathcal{A})$ // standard forward propagation except that convolutions are computed using equation 4.1 or 4.11
$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} =$ **BinaryBackward**$(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$
$\mathcal{W}^{t+1} =$ **UpdateParameters**$(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)$ // Any update rules (*e.g.*,SGD or ADAM)
$\eta^{t+1} =$ **UpdateLearningrate**(η^t, t) // Any learning rate scheduling function

the forward pass and backward propagation. To compute the gradient for sign function $\text{sign}(r)$, we follow the same approach as [14], where $\frac{\partial \text{sign}}{\partial r} = r1_{|r|\leq 1}$. The gradient in backward after the scaled sign function is $\frac{\partial C}{\partial W_i} = \frac{\partial C}{\partial \widetilde{W}_i}(\frac{1}{n} + \frac{\partial \text{sign}}{\partial W_i}\alpha)$. For updating the parameters, we use the high precision (real-value) weights. Because, in gradient descend the parameter changes are tiny, binarization after updating the parameters ignores these changes and the training objective can not be improved. [14, 16] also employed this strategy to train a binary network.

Algorithm 1 demonstrates our procedure for training a CNN with binary weights. First, we binarize the weight filters at each layer by computing $\mathcal{B}$ and $\mathcal{A}$. Then we call forward propagation using binary weights and its corresponding scaling factors, where all the convolutional operations are carried out by equation 4.1. Then, we call backward propagation, where the gradients are computed with respect to the estimated weight filters $\widetilde{\mathcal{W}}$. Lastly, the parameters and the learning rate gets updated by an update rule *e.g.*,SGD update with momentum or ADAM [67].

Once the training finished, there is no need to keep the real-value weights. Because, at inference we only perform forward propagation with the binarized weights.

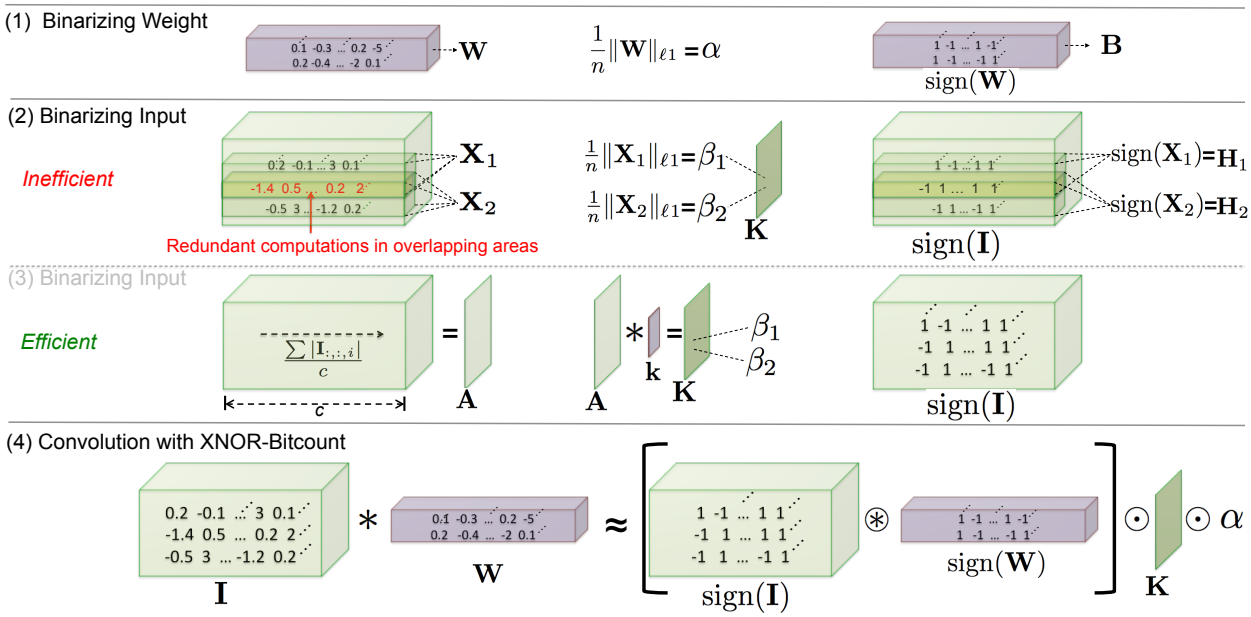


Figure 4.2: This figure illustrates the procedure explained in section 4.3.2 for approximating a convolution using binary operations.

4.3.2 XNOR-Networks

So far, we managed to find binary weights and a scaling factor to estimate the real-value weights. The inputs to the convolutional layers are still real-value tensors. Now, we explain how to binarize both weigths and inputs, so convolutions can be implemented efficiently using XNOR and bitcounting operations. This is the key element of our XNOR-Networks. In order to constrain a convolutional neural network $\langle \mathcal{I}, \mathcal{W}, * \rangle$ to have binary weights and binary inputs, we need to enforce binary operands at each step of the convolutional operation. A convolution consist of repeating a shift operation and a dot product. Shift operation moves the weight filter over the input and the dot product performs element-wise multiplications between the values of the weight filter and the corresponding part of the input. If we express dot product in terms of binary operations, convolution can be approximated using binary operations. Dot product between two binary vectors can be implemented by XNOR-Bitcounting operations [14]. In this section, we explain how to approximate the dot product between two vectors in $\mathbb{R}^n$ by a dot product between two vectors in $\{+1, -1\}^n$. Next, we demonstrate how to use this approximation for estimating a convolutional operation between two tensors.

Binary Dot Product:

To approximate the dot product between $\mathbf{X}, \mathbf{W} \in \mathbb{R}^n$ such that $\mathbf{X}^\mathsf{T}\mathbf{W} \approx \beta\mathbf{H}^\mathsf{T}\alpha\mathbf{B}$, where $\mathbf{H}, \mathbf{B} \in \{+1, -1\}^n$ and $\beta, \alpha \in \mathbb{R}^+$, we solve the following optimization:

$$\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha, \mathbf{B}, \beta, \mathbf{H}}{\operatorname{argmin}} \|\mathbf{X} \odot \mathbf{W} - \beta\alpha\mathbf{H} \odot \mathbf{B}\| \tag{4.7}$$

where $\odot$ indicates element-wise product. We define $\mathbf{Y} \in \mathbb{R}^n$ such that $\mathbf{Y}_i = \mathbf{X}_i\mathbf{W}_i$, $\mathbf{C} \in \{+1, -1\}^n$ such that $\mathbf{C}_i = \mathbf{H}_i\mathbf{B}_i$ and $\gamma \in \mathbb{R}^+$ such that $\gamma = \beta\alpha$. The equation 4.7 can be written as:

$$\gamma^*, \mathbf{C}^* = \underset{\gamma, \mathbf{C}}{\operatorname{argmin}} \|\mathbf{Y} - \gamma\mathbf{C}\| \tag{4.8}$$

the optimal solutions can be achieved from equation 4.2 as follow

$$\mathbf{C}^* = \operatorname{sign}(\mathbf{Y}) = \operatorname{sign}(\mathbf{X}) \odot \operatorname{sign}(\mathbf{W}) = \mathbf{H}^* \odot \mathbf{B}^* \tag{4.9}$$

Since $|\mathbf{X}_i|, |\mathbf{W}_i|$ are independent, knowing that $\mathbf{Y}_i = \mathbf{X}_i\mathbf{W}_i$ then,
$\mathbf{E}[|\mathbf{Y}_i|] = \mathbf{E}[|\mathbf{X}_i||\mathbf{W}_i|] = \mathbf{E}[|\mathbf{X}_i|]\,\mathbf{E}[|\mathbf{W}_i|]$ therefore,

$$\gamma^* = \frac{\sum|\mathbf{Y}_i|}{n} = \frac{\sum|\mathbf{X}_i||\mathbf{W}_i|}{n} \approx \left(\frac{1}{n}\|\mathbf{X}\|_{\ell 1}\right)\left(\frac{1}{n}\|\mathbf{W}\|_{\ell 1}\right) = \beta^*\alpha^* \tag{4.10}$$

Binary Convolution:

Convolving weight filter $\mathbf{W} \in \mathbb{R}^{c \times w \times h}$ (where $w_{in} \gg w, \;\; h_{in} \gg h$) with the input tensor $\mathbf{I} \in \mathbb{R}^{c \times w_{in} \times h_{in}}$ requires computing the scaling factor β for all possible sub-tensors in $\mathbf{I}$ with same size as $\mathbf{W}$. Two of these sub-tensors are illustrated in figure 4.2 (second row) by $\mathbf{X}_1$ and $\mathbf{X}_2$. Due to overlaps between subtensors, computing β for all possible sub-tensors leads to a large number of redundant computations. To overcome this redundancy, first, we compute a matrix $\mathbf{A} = \frac{\sum|\mathbf{I}_{:,:,i}|}{c}$, which is the average over absolute values of the elements in the input $\mathbf{I}$ across the channel. Then we convolve $\mathbf{A}$ with a 2D filter $\mathbf{k} \in \mathbb{R}^{w \times h}$, $\mathbf{K} = \mathbf{A} * \mathbf{k}$, where $\forall ij \;\; \mathbf{k}_{ij} = \frac{1}{w \times h}$. $\mathbf{K}$ contains scaling factors β for all sub-tensors in the input $\mathbf{I}$. $\mathbf{K}_{ij}$ corresponds to β for a sub-tensor centered at the location ij (across width and height). This procedure is shown in the third row of figure 4.2. Once we obtained the scaling factor α for the weight and β for all sub-tensors in $\mathbf{I}$ (denoted by $\mathbf{K}$), we can approximate the convolution between input $\mathbf{I}$ and weight filter $\mathbf{W}$ mainly using binary operations:

$$\mathbf{I} * \mathbf{W} \approx (\operatorname{sign}(\mathbf{I}) \circledast \operatorname{sign}(\mathbf{W})) \odot \mathbf{K}\alpha \tag{4.11}$$

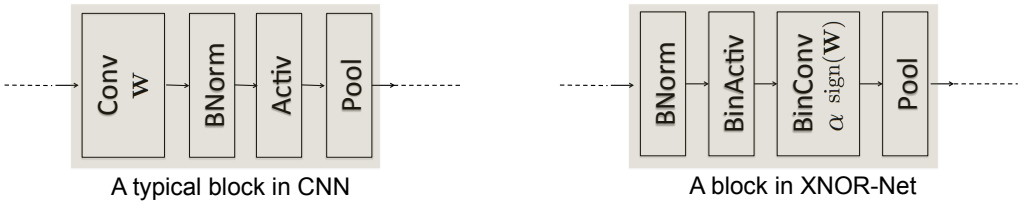


Figure 4.3: This figure contrasts the block structure in our XNOR-Network (right) with a typical CNN (left).

where $\circledast$ indicates a convolutional operation using XNOR and bitcount operations. This is illustrated in the last row in figure 4.2. Note that the number of non-binary operations is very small compared to binary operations.

Training XNOR-Networks:

A typical block in CNN contains several different layers. Figure 4.3 (left) illustrates a typical block in a CNN. This block has four layers in the following order: 1-Convolutional, 2-Batch Normalization, 3-Activation and 4-Pooling. Batch Normalization layer[63] normalizes the input batch by its mean and variance. The activation is an element-wise non-linear function (*e.g.,*Sigmoid, ReLU). The pooling layer applies any type of pooling (*e.g.,*max,min or average) on the input batch. Applying pooling on binary input results in significant loss of information. For example, max-pooling on binary input returns a tensor that most of its elements are equal to $+1$. Therefore, we put the pooling layer after the convolution. To further decrease the information loss due to binarization, we normalize the input before binarization. This ensures the data to hold zero mean, therefore, thresholding at zero leads to less quantization error. The order of layers in a block of binary CNN is shown in Figure 4.3(right).

The binary activation layer(BinActiv) computes $\mathbf{K}$ and $\mathrm{sign}(\mathbf{I})$ as explained in section 4.3.2. In the next layer (BinConv), given $\mathbf{K}$ and $\mathrm{sign}(\mathbf{I})$, we compute binary convolution by equation 4.11. Then at the last layer (Pool), we apply the pooling operations. We can insert a non-binary activation(*e.g.,*ReLU) after binary convolution. This helps when we use state-of-the-art networks (*e.g.,*AlexNet or VGG).

Once we have the binary CNN structure, the training algorithm would be the same as algorithm 1.

Binary Gradient: The computational bottleneck in the backward pass at each layer is computing a convolution between weight filters(w) and the gradients with respect of the inputs (g^{in}). Similar to binariza-

tion in the forward pass, we can binarize g^{in} in the backward pass. This leads to a very efficient training procedure using binary operations. Note that if we use equation 4.6 to compute the scaling factor for g^{in}, the direction of maximum change for SGD would be diminished. To preserve the maximum change in all dimensions, we use $\max_i(|g_i^{in}|)$ as the scaling factor.

k-bit Quantization: So far, we showed 1-bit quantization of weights and inputs using $\text{sign}(x)$ function. One can easily extend the quantization level to k-bits by using $q_k(x) = 2(\frac{[(2^k-1)(\frac{x+1}{2})]}{2^k-1} - \frac{1}{2})$ instead of the sign function. Where $[.]$ indicates rounding operation and $x \in [-1, 1]$.

4.4 Experiments

We evaluate our method by analyzing its efficiency and accuracy. We measure the efficiency by computing the computational speedup (in terms of number of high precision operation) achieved by our binary convolution vs. standard convolution. To measure accuracy, we perform image classification on the large-scale ImageNet dataset. This paper is the first work that evaluates binary neural networks on the ImageNet dataset. Our binarization technique is general, we can use any CNN architecture. We evaluate AlexNet [70] and two deeper architectures in our experiments. We compare our method with two recent works on binarizing neural networks; BinaryConnect [16] and BinaryNet [14]. The classification accuracy of our binary-weight-network version of AlexNet is as accurate as the full precision version of AlexNet. This classification accuracy outperforms competitors on binary neural networks by a large margin. We also present an ablation study, where we evaluate the key elements of our proposed method; computing scaling factors and our block structure for binary CNN. We shows that our method of computing the scaling factors is important to reach high accuracy.

4.4.1 Efficiency Analysis

In an standard convolution, the total number of operations is $cN_{\mathbf{W}}N_{\mathbf{I}}$, where c is the number of channels, $N_{\mathbf{W}} = wh$ and $N_{\mathbf{I}} = w_{in}h_{in}$. Note that some modern CPUs can fuse the multiplication and addition as a single cycle operation. On those CPUs, Binary-Weight-Networks does not deliver speed up. Our binary approximation of convolution (equation 4.11) has $cN_{\mathbf{W}}N_{\mathbf{I}}$ binary operations and $N_{\mathbf{I}}$ non-binary operations. With the current generation of CPUs, we can perform 64 binary operations in one clock of CPU, therefore the speedup can be computed by $S = \frac{cN_{\mathbf{W}}N_{\mathbf{I}}}{\frac{1}{64}cN_{\mathbf{W}}N_{\mathbf{I}}+N_{\mathbf{I}}} = \frac{64cN_{\mathbf{W}}}{cN_{\mathbf{W}}+64}$.

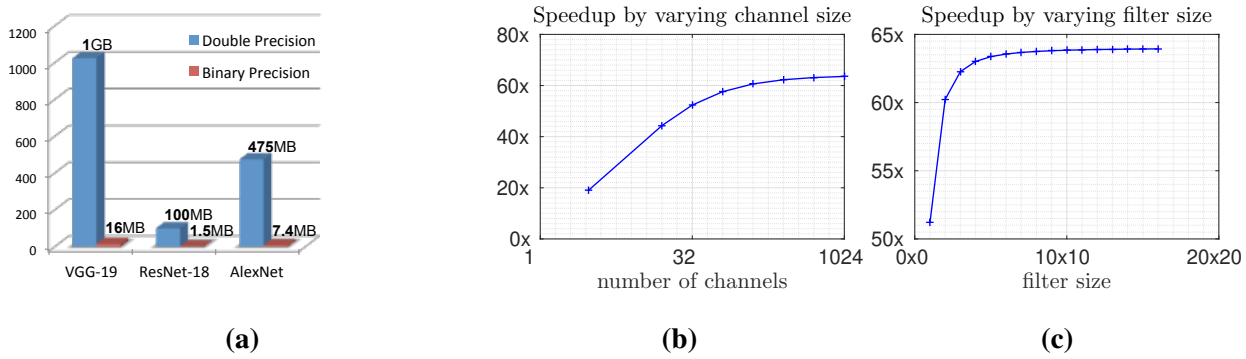


Figure 4.4: This figure shows the efficiency of binary convolutions in terms of memory(a) and computation(b-c). (a) is contrasting the required memory for binary and double precision weights in three different architectures(AlexNet, ResNet-18 and VGG-19). (b,c) Show speedup gained by binary convolution under (b)-different number of channels and (c)-different filter size

The speedup depends on the channel size and filter size but not the input size. In figure 4.4-(b-c) we illustrate the speedup achieved by changing the number of channels and filter size. While changing one parameter, we fix other parameters as follows: $c = 256$, $n_{\mathbf{I}} = 14^2$ and $n_{\mathbf{W}} = 3^2$ (majority of convolutions in ResNet[54] architecture have this structure). Using our approximation of convolution we gain 62.27$\times$ theoretical speed up, but in our CPU implementation with all of the overheads, we achieve 58x speed up in one convolution (Excluding the process for memory allocation and memory access). With the small channel size ($c = 3$) and filter size ($N_{\mathbf{W}} = 1 \times 1$) the speedup is not considerably high. This motivates us to avoid binarization at the first and last layer of a CNN. In the first layer the chanel size is 3 and in the last layer the filter size is 1×1. A similar strategy was used in [14]. Figure 4.4-a shows the required memory for three different CNN architectures(AlexNet, VGG-19, ResNet-18) with binary and double precision weights. Binary-weight-networks are so small that can be easily fitted into portable devices. BinaryNet [14] is in the same order of memory and computation efficiency as our method. In Figure 4.4, we show an analysis of computation and memory cost for a binary convolution. The same analysis is valid for BinaryNet and BinaryConnect. The key difference of our method is using a scaling-factor, which does not change the order of efficiency while providing a significant improvement in accuracy.

4.4.2 Image Classification

We evaluate the performance of our proposed approach on the task of natural image classification. So far, in the literature, binary neural network methods have presented their evaluations on either limited domain or simplified datasets *e.g.,*CIFAR-10, MNIST, SVHN. To compare with state-of-the-art vision, we evaluate our method on ImageNet (ILSVRC2012). ImageNet has $\sim$1.2M train images from 1K categories and 50K validation images. The images in this dataset are natural images with reasonably high resolution compared to the CIFAR and MNIST dataset, which have relatively small images. We report our classification performance using Top-1 and Top-5 accuracies. We adopt three different CNN architectures as our base architectures for binarization: AlexNet [70], Residual Networks (known as ResNet) [54], and a variant of GoogLenet [121].We compare our Binary-weight-network (**BWN**) with BinaryConnect(**BC**) [16] and our XNOR-Networks(**XNOR-Net**) with BinaryNeuralNet(**BNN**) [14]. BinaryConnect(BC) is a method for training a deep neural network with binary weights during forward and backward propagations. Similar to our approach, they keep the real-value weights during the updating parameters step. Our binarization is different from BC. The binarization in BC can be either deterministic or stochastic. We use the deterministic binarization for BC in our comparisons because the stochastic binarization is not efficient. The same evaluation settings have been used and discussed in [14]. BinaryNeuralNet(BNN) [14] is a neural network with binary weights and activations during inference and gradient computation in training. In concept, this is a similar approach to our XNOR-Network but the binarization method and the network structure in BNN is different from ours. Their training algorithm is similar to BC and they used deterministic binarization in their evaluations.

CIFAR-10 : BC and BNN showed near state-of-the-art performance on CIFAR-10, MNIST, and SVHN dataset. BWN and XNOR-Net on CIFAR-10 using the same network architecture as BC and BNN achieve the error rate of 9.88% and 10.17% respectively. In this paper we explore the possibility of obtaining near state-of-the-art results on a much larger and more challenging dataset (ImageNet).

AlexNet: [70] is a CNN architecture with 5 convolutional layers and two fully-connected layers. This architecture was the first CNN architecture that showed to be successful on ImageNet classification task. This network has 61M parameters. We use AlexNet coupled with batch normalization layers [63].

Train: In each iteration of training, images are resized to have 256 pixel at their smaller dimension and then a random crop of 224×224 is selected for training. We run the training algorithm for 16 epochs with

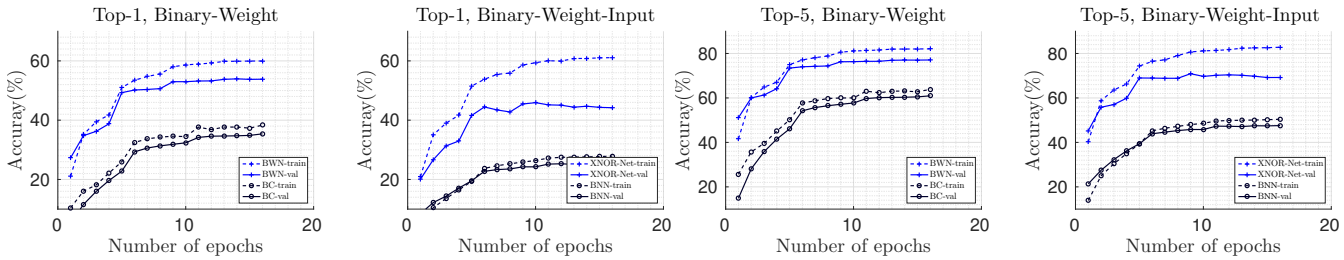


Figure 4.5: This figure compares the imagenet classification accuracy on Top-1 and Top-5 across training epochs. Our approaches BWN and XNOR-Net outperform BinaryConnect(BC) and BinaryNet(BNN) in all the epochs by large margin($\sim$17%).

Classification Accuracy(%)									
Binary-Weight				Binary-Input-Binary-Weight				Full-Precision	
BWN		BC[14]		XNOR-Net		BNN[14]		AlexNet[70]	
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2

Table 4.1: This table compares the final accuracies (Top1 - Top5) of the full precision network with our binary precision networks; Binary-Weight-Networks(BWN) and XNOR-Networks(XNOR-Net) and the competitor methods; BinaryConnect(BC) and BinaryNet(BNN).

batche size equal to 512. We use negative-log-likelihood over the soft-max of the outputs as our classification loss function. In our implementation of AlexNet we do not use the Local-Response-Normalization(LRN) layer[3]. We use SGD with momentum=0.9 for updating parameters in BWN and BC. For XNOR-Net and BNN we used ADAM [67]. ADAM converges faster and usually achieves better accuracy for binary inputs [14]. The learning rate starts at 0.1 and we apply a learning-rate-decay=0.01 every 4 epochs.

Test: At inference time, we use the 224×224 center crop for forward propagation.

Figure 4.5 demonstrates the classification accuracy for training and inference along the training epochs for top-1 and top-5 scores. The dashed lines represent training accuracy and solid lines shows the validation

[3]Our implementation is followed by https://gist.github.com/szagoruyko/dd032c529048492630fc

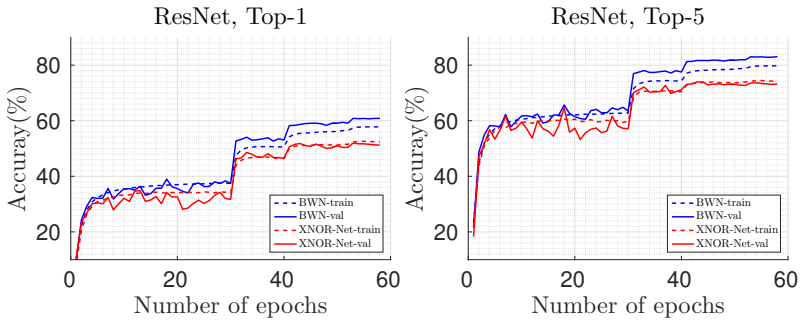


Figure 4.6: This figure shows the classification accuracy; (a)Top-1 and (b)Top-5 measures across the training epochs on ImageNet dataset by Binary-Weight-Network and XNOR-Network using ResNet-18.

	ResNet-18		GoogLenet	
Network Variations	top-1	top-5	top-1	top-5
Binary-Weight-Network	60.8	83.0	65.5	86.1
XNOR-Network	51.2	73.2	N/A	N/A
Full-Precision-Network	69.3	89.2	71.3	90.0

Table 4.2: This table compares the final classification accuracy achieved by our binary precision networks with the full precision network in ResNet-18 and GoogLenet architectures.

accuracy. In all of the epochs our method outperforms BC and BNN by large margin ($\sim$17%). Table 4.1 compares our final accuracy with BC and BNN. We found that the scaling factors for the weights (α) is much more effective than the scaling factors for the inputs (β). Removing β reduces the accuracy by a small margin (less than 1% top-1 alexnet).

Binary Gradient: Using XNOR-Net with binary gradient the accuracy of top-1 will drop only by 1.4%.

Residual Net : We use the ResNet-18 proposed in [54] with short-cut type B.[4]

Train: In each training iteration, images are resized randomly between 256 and 480 pixel on the smaller dimension and then a random crop of 224×224 is selected for training. We run the training algorithm for 58 epochs with batch size equal to 256 images. The learning rate starts at 0.1 and we use the learning-rate-decay equal to 0.01 at epochs number 30 and 40.

Test: At inference time, we use the 224×224 center crop for forward propagation.

Figure 4.6 demonstrates the classification accuracy (Top-1 and Top-5) along the epochs for training and inference. The dashed lines represent training and the solid lines represent inference. Table 4.2 shows our final accuracy by BWN and XNOR-Net.

GoogLenet Variant : We experiment with a variant of GoogLenet [121] that uses a similar number of parameters and connections but only straightforward convolutions, no branching[5]. It has 21 convolutional layers with filter sizes alternating between 1×1 and 3×3.

Train: Images are resized randomly between 256 and 320 pixel on the smaller dimension and then a random crop of 224×224 is selected for training. We run the training algorithm for 80 epochs with batch size of 128. The learning rate starts at 0.1 and we use polynomial rate decay, $\beta = 4$.

Test: At inference time, we use a center crop of 224×224.

4.4.3 Ablation Studies

There are two key differences between our method and the previous network binariazation methods; the binararization technique and the block structure in our binary CNN. For binarization, we find the optimal scaling factors at each iteration of training. For the block structure, we order the layers in a block in a way that decreases the quantization loss for training XNOR-Net. Here, we evaluate the effect of each of these elements in the performance of the binary networks. Instead of computing the scaling factor α using equation 4.6, one can consider α as a network parameter. In other words, a layer after binary convolution multiplies the output of convolution by an scalar parameter for each filter. This is similar to computing the affine parameters in batch normalization. Table 4.3-a compares the performance of a binary network with two ways of computing the scaling factors. As we mentioned in section 4.3.2 the typical block structure in

[4]We used the Torch implementation in https://github.com/facebook/fb.resnet.torch

[5]We used the Darknet [96] implementation: http://pjreddie.com/darknet/imagenet/#extraction

Binary-Weight-Network		
Strategy for computing α	top-1	top-5
Using equation 4.6	56.8	79.4
Using a separate layer	46.2	69.5

(a)

XNOR-Network		
Block Structure	top-1	top-5
C-B-A-P	30.3	57.5
B-A-C-P	44.2	69.2

(b)

Table 4.3: In this table, we evaluate two key elements of our approach; computing the optimal scaling factors and specifying the right order for layers in a block of CNN with binary input. (a) demonstrates the importance of the scaling factor in training binary-weight-networks and (b) shows that our way of ordering the layers in a block of CNN is crucial for training XNOR-Networks. C,B,A,P stands for Convolutional, BatchNormalization, Acive function (here binary activation), and Pooling respectively.

CNN is not suitable for binarization. Table 4.3-b compares the standard block structure C-B-A-P (Convolution, Batch Normalization, Activation, Pooling) with our structure B-A-C-P. (A, is binary activation).

4.5 Conclusion

We introduce simple, efficient, and accurate binary approximations for neural networks. We train a neural network that learns to find binary values for weights, which reduces the size of network by $\sim 32\times$ and provide the possibility of loading very deep neural networks into portable devices with limited memory. We also propose an architecture, XNOR-Net, that uses mostly bitwise operations to approximate convolutions. This provides $\sim 58\times$ speed up and enables the possibility of running the inference of state of the art deep neural network on CPU (rather than GPU) in real-time.

Chapter 5

CONCLUSION

This was a fascinating time to be in graduate school for computer vision. I got to witness in real-time the paradigm shift from classical computer vision techniques to almost exclusively convolutional neural network-based approaches. This massive technological leap seems likely to be a once-in-a-career phenomenon.

5.1 Object Detection

Although a core problem in computer vision, it was initially unclear how detection algorithms could benefit from convolutional neural networks. Detection is inherently predicting a variable-sized output (there may be no objects or many in a scene) while neural networks require a fixed output prediction size for most configurations. Early on researchers simply tried to adapt the new features generated from CNNs for SVM-based approaches like DPM, to no avail (I tried this and Ross said he did too). It took a brand new approach to take advantage of the power of CNNs for detection.

5.1.1 R-CNN

R-CNN initially demonstrated that CNNs could be used for detection and it's successors set the bar for accuracy in object detection [39] [42] [102]. Most current research in detection builds off of the R-CNN framework [52] [78] [77]. The region proposal network of Faster R-CNN is similar in structure and functionality to YOLO (they were developed concurrently). The additional step of ROI pooling and classification/bounding box refinement gives Faster R-CNN an edge in accuracy, especially for smaller objects. I suspect some of this is due to the fact that R-CNN uses a shared classifier for all regions, meaning it sees more data during training. The classifier also may see a more consistent view of the object/background making the classification task easier.

Mask R-CNN in itself is an incredible advance, combining detection and segmentation [52]. The added

information from training on segmentation gives Mask R-CNN an accuracy improvement over vanilla Faster R-CNN. I tried training a similar mask regression layer in YOLO but never managed to get good results. YOLO doesn't see a consistent view of the object during it's bounding box regression/potential mask prediction phase since it doesn't have an ROI pooling step. This makes mask prediction less feasible without some careful thought as to the parameterization.

5.1.2 SSD

SSD (developed concurrently) and it's offshoots are most similar to YOLO [83] [35]. I initially tried to make YOLO fully convolutional but a bug in Darknet reshuffled the channels when I was trying to calculate the detections so I only managed to get the version with fully connected layers working in version 1. SSD uses fully convolutional networks from the start avoiding the size and overfitting issues with fully connected layers.

The overlap between SSD and YOLO is interesting as a case of convergent development. For example YOLOv2 and DSSD, which came out within a month of each other on arXiv, both use K-means clustering on the training boxes to find a suitable set of anchor boxes. However, some differences still remain between the two frameworks. For example SSD uses hard negative in training while YOLO relies on the conditional probability formulation to make progress in training despite the class imbalance of positive and negative boxes.

5.1.3 Alternative Formulations

I'm excited to see that new, innovative detection frameworks are still emerging. New parameterizations like CornerNet show that predictions don't need to happen from the center out, the network is just as capable of finding the boundaries of objects and predicting them [71]. This perhaps suggests that the network learns some latent representation of objects that could equally be used to predict the center/size of objects or their boundaries directly or perhaps other parameterizations.

TensorMask and other bottom-up approaches to segmentation remind us that instance segmentation is a generalization of object detection [12]. Solving instance segmentation as a precursor to detection feels similar to parts-based approaches to detection and also seems closer to how humans approach the problem. Perhaps when the datasets are available detection will be modeled as a 3D parts model / segmentation task

with bounding boxes placed around the predicted structure.

5.2 Efficient Networks

There are a few easy ways to make neural networks more accurate at vision tasks. One is adding more data; the more samples you have from a distribution the less likely you are to overfit and the better you are at generalizing to unseen data. The second, often easier, option is to make the model larger (and find some way to deal with the potential overfitting problems that may arise. This has led to models with thousands of layers, millions of parameters, and billions of FLOPs. Naturally, as people are more interested in deploying these networks, researchers are focusing more on making networks that are more efficient across a variety of metrics.

5.2.1 Network Design

Generally the three metrics people report are number of parameters, number of floating point operations, and execution speed on a given device. While these things are loosely correlated, they have a complex relationship. Fully connected layers have a very large number of parameters but are very fast to execute. However, depthwise-separable convolutional layers with a small group size have very few parameters but can take a long time to execute on GPU relative to their number of FLOPs. On most machines CPU execution speed is directly tied to number of FLOPs but on GPU execution speed is more about leveraging the parallelism of a model. Thin, deep models are much less efficient in terms of speed than wide, shallow models on GPU.

Early attempts to optimize model efficiency rely on grad student descent of different model designs. SqueezeNet and MobileNet showed that designs like AlexNet and VGG were far from optimal and that it could be practical to run networks on resource-constrained hardware to solve real problems [61] [58]. Soon big companies like Google started throwing resources at the design space trying thousands of combinations of building blocks to see what works best [131].

Since I focus on real-time applications of detection I hand-design networks specifically for low latency. I want my networks to execute as fast as possible on a single image on GPU. While techniques like depthwise seperable convolutions or thin, deep ResNets can improve accuracy relative to number of floating point operations, they decrease the parallelism of the network too much for low-latency applications. Darknet53 is the same number of floating point operations as ResNet-101, more accurate, and almost 50% faster.

5.2.2 Model Quantization and Approximation

Distinct from model design, quantization and approximation techniques take existing designs and make them more efficient. XNOR-Nets represent one extreme of quantization techniques. Quantization from 32-bit floats to 16-bit has basically no effect on training and is standard practice for training on the latest GPUs. Even quantizing to 8-bit or 4-bit can have minimal effect on accuracy at test time [13] [6]. Unlike these approaches, XNOR-Nets take a significant hit in terms of accuracy but enable a range of binary optimizations that don't require specialized hardware support. They can also have a very small footprint which is nice for model distribution and fast loading on mobile devices.

Combining YOLO with XNOR-Nets yields a detection model capable of running in real-time on mobile devices without relying on GPU acceleration or cloud processing. This enables a range of possibilities for assistive devices, robotics, mobile photography and more. And it made my advisor a millionaire so that's pretty cool :-D

5.3 The Future

Computer vision approaches a crossroads. Our algorithms approach saturation on standard benchmarks for the core vision problems (classification, segmentation, detection). From outward appearances these problems might seem solved. However, users of vision systems still fight an uphill battle applying these techniques to their domain, facing weeks or months of data collection, labelling, retraining, and tuning to get good performance.

There is a keen need for single or few-shot learning in object detection. Many YOLO users want to quickly train their detectors on new object classes and don't have the time or resources to train from scratch. I'm researching methods for robust learning with few examples that involve predicting a vector space embedding of an object's category instead of the category itself. This would enable new categories to be added on the fly without retraining. It would also allow for tracking and reidentification of objects between frames or images.

Efficiency also remains an issue; the best way to improve neural network accuracy is still to add more layers. Attention models show some promise because they allow the network to concentrate on certain, important areas of an image. I'm interested in exploring methods to use attention models to reduce the computation a network performs by ignoring unimportant regions and only performing partial convolutions

instead of simply reweighting the fully computed features.

Looking further into the future, vision excels at pattern matching tasks (classification, object detection, etc.) but fails at more complex tasks that necessitate higher-level reasoning. I'm excited about new approaches that see vision as a part of a broader solution to tasks in AI. There is strong potential for vision and language to mutually benefit from building shared models of the world. Similarly, robots that combine vision and interaction to explore the world—embodied vision—offer opportunities to more closely model how biological agents learn. Humans do not learn to see in a vacuum, we learn in holistic, interconnected process that involves sensing, interaction, and prediction. The next frontier in vision could be tying together many processes into a single agent.

BIBLIOGRAPHY

[1] Analogy. *Wikipedia*, Mar 2018. 53

[2] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Fixed point optimization of deep convolutional neural networks for object recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1131–1135. IEEE, 2015. 71

[3] Sanjeev Arora, Aditya Bhaskara, Rong Ge, and Tengyu Ma. Provable bounds for learning some deep representations. *arXiv preprint arXiv:1310.6343*, 2013. 71

[4] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014. 70

[5] Carlo Baldassi, Alessandro Ingrosso, Carlo Lucibello, Luca Saglietti, and Riccardo Zecchina. Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses. *Physical review letters*, 115(12):128101, 2015. 72

[6] Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, pages 7948–7956, 2019. 88

[7] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. *arXiv preprint arXiv:1512.04143*, 2015. 41

[8] Matthew B Blaschko and Christoph H Lampert. Learning to localize objects with structured output regression. In *Computer Vision–ECCV 2008*, pages 2–15. Springer, 2008. 18

[9] Lubomir Bourdev and Jitendra Malik. Poselets: Body part detectors trained using 3d human pose annotations. In *International Conference on Computer Vision (ICCV)*, 2009. 25

[10] Hongping Cai, Qi Wu, Tadeo Corradi, and Peter Hall. The cross-depiction problem: Computer vision algorithms for recognising objects in artwork and in photographs. *arXiv preprint arXiv:1505.00110*, 2015. 24

[11] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *arXiv preprint arXiv:1504.04788*, 2015. 70

[12] Xinlei Chen, Ross Girshick, Kaiming He, and Piotr Dollár. Tensormask: A foundation for dense object segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2061–2069, 2019. 86

[13] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3009–3018. IEEE, 2019. 88

[14] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, 2016. 69, 71, 74, 75, 78, 79, 80, 81

[15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014. 71

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3105–3113, 2015. 71, 74, 78, 80

[17] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. 69

[18] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 18, 25

[19] Yann N Dauphin and Yoshua Bengio. Big neural networks waste capacity. *arXiv preprint arXiv:1301.3583*, 2013. 69

[20] Thomas Dean, Mark Ruzon, Michael Segal, Jonathon Shlens, Sudheendra Vijayanarasimhan, Jay Yagnik, et al. Fast, accurate detection of 100,000 object classes on a single machine. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 1814–1821. IEEE, 2013. 19, 20

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 2, 6

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 31

[23] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013. 69

[24] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014. 70

[25] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013. 18

[26] Jian Dong, Qiang Chen, Shuicheng Yan, and Alan Yuille. Towards unified object detection and semantic segmentation. In *Computer Vision–ECCV 2014*, pages 299–314. Springer, 2014. 28

[27] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2155–2162. IEEE, 2014. 19, 21

[28] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015. 71

[29] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015. 12

[30] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 2, 6, 31

[31] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 63

[32] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010. 10, 18

[33] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010. 3, 45

[34] David Forsyth. personal communication. 62

[35] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017. 66, 86

[36] Spyros Gidaris and Nikos Komodakis. Object detection via a multi-region & semantic segmentation-aware CNN model. *CoRR*, abs/1505.01749, 2015. 28

[37] Shiry Ginosar, Daniel Haas, Timothy Brown, and Jitendra Malik. Detecting people in cubist art. In *Computer Vision-ECCV 2014 Workshops*, pages 101–116. Springer, 2014. 24

[38] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448, 2015. 67

[39] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra 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. 3, 85

[40] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 580–587. IEEE, 2014. 10, 18, 28, 67

[41] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 11, 18, 19, 20, 21, 28

[42] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 39, 40, 41, 85

[43] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014. 71

[44] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox, and Ali Farhadi. Iqa: Visual question answering in interactive environments. *arXiv preprint arXiv:1712.03316*, 2017. 53

[45] ML Gottmer. Merging reality and virtuality with microsoft hololens. 2015. 67

[46] Stephen Gould, Tianshi Gao, and Daphne Koller. Region-based segmentation and object detection. In *Advances in neural information processing systems*, pages 655–663, 2009. 18

[47] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 70

[48] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015. 70

[49] Stephen José Hanson and Lorien Y Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pages 177–185, 1989. 70

[50] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *Computer Vision–ECCV 2014*, pages 297–312. Springer, 2014. 28

[51] Babak Hassibi and David G Stork. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann, 1993. 70

[52] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 85

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *arXiv preprint arXiv:1406.4729*, 2014. 20

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, 2015. 67, 70, 79, 80, 83

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 31, 39, 40, 66

[56] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 16

[57] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. In *Computer Vision–ECCV 2012*, pages 340–353. Springer, 2012. 22

[58] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 87

[59] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. 66

[60] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pages 1–6. IEEE, 2014. 71

[61] Forrest N Iandola, Matthew W Moskewicz, Khalid Ashraf, Song Han, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 70, 87

[62] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 32, 39

[63] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 77, 80

[64] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 70

[65] Juhwan Kim and Son-Cheol Yu. Convolutional neural network-based real-time rov detection using forward-looking sonar image. In *Autonomous Underwater Vehicles (AUV), 2016 IEEE/OES*, pages 396–400. IEEE, 2016. 2

[66] Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016. 72

[67] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 74, 81

[68] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*, 2017. 55

[69] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 32

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 67, 68, 78, 80, 81

[71] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018. 86

[72] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPs*, volume 89, 1989. 70

[73] Karel Lenc and Andrea Vedaldi. R-cnn minus r. *arXiv preprint arXiv:1506.06981*, 2015. 21

[74] Rainer Lienhart and Jochen Maydt. An extended set of haar-like features for rapid object detection. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pages I–900. IEEE, 2002. 18

[75] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013. 13

[76] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 39, 70

[77] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017. 57, 66, 85

[78] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017. iv, 54, 59, 66, 85

[79] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2, 6, 31, 41

[80] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 57

[81] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015. 71

[82] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 66

[83] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, and Scott E. Reed. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015. 39, 40, 41, 86

[84] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. 69

[85] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999. 18

[86] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 3(4):235–244, 1990. 42

[87] Dmytro Mishkin. Models accuracy on imagenet 2012 val. https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val. Accessed: 2015-10-2. 14

[88] Isaac Newton. *Philosophiae naturalis principia mathematica*. William Dawson & Sons Ltd., London, 1687. 53

[89] Liang Niu, Cheng Qian, John-Ross Rizzo, Todd Hudson, Zichen Li, Shane Enright, Eliot Sperling, Kyle Conti, Edward Wong, and Yi Fang. A wearable assistive technology for the visually impaired with door knob detection and real-time feedback for hand-to-handle manipulation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1500–1508, 2017. 2

[90] VR Oculus. Oculus rift-virtual reality headset for 3d gaming. *URL: http://www. oculusvr. com*, 2012. 67

[91] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *Computer vision, 1998. sixth international conference on*, pages 555–562. IEEE, 1998. 18

[92] Jason Parham, Tanya Berger-Wolf, and Daniel Rubenstein. Animal population censusing at scale with citizen science and photographic identification. 2017. 2

[93] Jason Parham, Jonathan Crall, Charles Stewart, Tanya Berger-Wolf, and Daniel Rubenstein. Animal population censusing at scale with citizen science and photographic identification. 2017. 61

[94] Konstantin Pogorelov, Michael Riegler, Sigrun Losada Eskeland, Thomas de Lange, Dag Johansen, Carsten Griwodz, Peter Thelin Schmidt, and Pål Halvorsen. Efficient disease detection in gastrointestinal videos–global features versus neural networks. *Multimedia Tools and Applications*, 76(21):22493–22525, 2017. 2

[95] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016. 5

[96] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016. 2, 14, 40, 58, 83

[97] Joseph Redmon and Anelia Angelova. Real-time grasp detection using convolutional neural networks. *CoRR*, abs/1412.3128, 2014. 19

[98] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016. 2, 3, 39, 40

[99] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016. iii, 1, 3

[100] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 6517–6525. IEEE, 2017. iv, 54, 56, 66

[101] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. iv, 3, 5, 59

[102] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015. 19, 20, 21, 28, 32, 35, 39, 40, 41, 55, 67, 85

[103] Shaoqing Ren, Kaiming He, Ross B. Girshick, Xiangyu Zhang, and Jian Sun. Object detection networks on convolutional feature maps. *CoRR*, abs/1504.06066, 2015. 14, 28

[104] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015. 14, 32

[105] Olga Russakovsky, Li-Jia Li, and Li Fei-Fei. Best of both worlds: human-machine collaboration for object annotation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2121–2131, 2015. 60

[106] Mohammad Amin Sadeghi and David Forsyth. 30hz object detection with dpm v5. In *Computer Vision–ECCV 2014*, pages 65–79. Springer, 2014. 19, 20, 21

[107] Micah Scott. Smart camera gimbal bot – scanlime:027, Dec 2017. 61

[108] Frank Seide, Gang Li, and Dong Yu. Conversational speech transcription using context-dependent deep neural networks. In *Interspeech*, pages 437–440, 2011. 69

[109] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *CoRR*, abs/1312.6229, 2013. 18, 19

[110] Zhiqiang Shen and Xiangyang Xue. Do more dropouts in pool5 feature maps for better object detection. *arXiv preprint arXiv:1409.6911*, 2014. 28

[111] Abhinav Shrivastava, Rahul Sukthankar, Jitendra Malik, and Abhinav Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016. 66

[112] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. 1

[113] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 31, 37, 39

[114] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 67, 68

[115] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems*, pages 963–971, 2014. 71

[116] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016. 31

[117] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, 2016. 70

[118] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. 2017. 66

[119] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 12

[120] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. 37

[121] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. 67, 70, 80, 83

[122] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016. 6, 31

[123] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013. 18

[124] Hien Van Nguyen, Kevin Zhou, and Raviteja Vemulapalli. Cross-domain synthesis of medical images using efficient location-sensitive deep network. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015*, pages 677–684. Springer, 2015. 70

[125] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, volume 1, 2011. 71

[126] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 4:34–47, 2001. 18

[127] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004. 19

[128] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013. 71

[129] Junjie Yan, Zhen Lei, Longyin Wen, and Stan Z Li. The fastest deformable part model for object detection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 2497–2504. IEEE, 2014. 19, 20, 21

[130] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *Computer Vision–ECCV 2014*, pages 391–405. Springer, 2014. 18

[131] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 87