Spring 5-22-2019

Robust Lightweight Object Detection

Siddharth Kumar
San Jose State University

Robust Lightweight Object Detection

A project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

By

Siddharth Kumar

May 2019

The Designated Project Committee Approves the Project Titled

Robust Lightweight Object Detection

by

Siddharth Kumar

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2019

Abstract

Robust Lightweight Object Detection

By Siddharth Kumar

Object detection is a very challenging problem in computer vision and has been a prominent subject of research for nearly three decades. There has been a promising increase in the accuracy and performance of object detectors ever since deep convolutional networks (CNN) were introduced. CNNs can be trained on large datasets made of high resolution images without flattening them, thereby using the spatial information. Their superior learning ability also makes them ideal for image classification and object detection tasks. Unfortunately, this power comes at the big cost of compute and memory. For instance, the Faster R-CNN detector required 180 billion FLOPs for training, and has over 100 million parameters.

In this project, we explore the popular state-of-the-art object detectors and present their contributions and shortcomings. Then we explore the recent lightweight detectors which try to address the issue of high resource requirements by building leaner models. Building upon the contributions of the state-of-the-art object detectors, and recent developments in CNN training, we propose our own lightweight detector. We proposed a novel CNN block, to improve the inter-channel dependency in feature maps, called the inter-channel dependency block (ICDB). Through experiments on benchmark datasets we demonstrated our model attains better accuracy compared to the previous methods. Three benchmarking datasets PASCAL VOC 2007, KITTI and COCO have been used to demonstrate that our model scales well to different scenarios.

***Keywords*: Convolutional neural networks, deep learning, Faster R-CNN, neural networks, object detection, SSD, YOLO**

Acknowledgements

I would like to thank Dr. Robert Chun for giving me the opportunity to work on this project and for his continued guidance and support necessary to complete this project. I would also like to thank my committee members for their invaluable feedback that helped me improve this project. Finally, I would like to thank my family and friends and for their continuous support and guidance throughout the duration of this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In simple words, object detection is classification and localization of common everyday objects in natural scene images or videos. Ever since deep convolutional neural networks (CNN) were applied to object detection, there has been a tremendous improvement in the accuracy of object detectors. These detectors can be trained on big datasets with high-resolution images, with multiple object classes.

Object detection is a popular topic of research in compute science because of its wide scope of applications. With minor changes, generic object detection techniques can be used in domain-specific applications like pose estimation, pedestrian detection [27], [6] , face detection [36], [39], autonomous driving , human-behaviour analysis, to name a few. It is the most fundamental technique for getting useful insights into images and videos.

Given the diverse applications, it is imperative that an object detection system is *'robust'* i.e., can handle a comprehensive set of inputs with good output accuracy, have high detection speed and make efficient use of resources, to be cost effective. All of this makes object detection a challenging task. Moreover, the real-life images suffer from lighting-problems, occlusion, pose, viewpoint and size variations, which make the task of object detection harder.

Over the recent years, thanks to steadfast developments in machine learning and deep learning, researchers have been able to address most of these problems, yielding promising results. Based on these recent developments, the process of object detection (using deep learning) can split into three simpler steps, as follows:

Region Selection: Natural images may contain multiple objects with varying sizes and positions, this is why the entire image need to be searched for objects. A naive approach would be to use a sliding windows of changing sizes that goes through the entire image. This generates all possible bounding-boxes, most of which would not contain any object. These windows make region selection slow. Therefore it is important to have a efficient method for generating region proposals such that it eliminates most of the useless proposals and retains most of the relevant proposals. In chapter three, some of more recent detectors have been presented that used more sophisticated techniques for generating region proposals. It can observed that having a good region selection method makes the detector faster and more accurate. In the second part of chapter three, there are method that eliminate the region selection step and transform object detection into a regression problem.

Feature extraction: The most important and the compute intensive step is feature extraction. It involves generating different types of features from the image that may be used for recognizing the objects. Before deep learning, feature like Histogram of sparse codecs (HSC) [32], HOG [3] and Haar-like [22] were used for object detection. Unfortunately, these feature had to be extracted manually from the image and then fed in the corresponding detector. Moreover, the scope of these feature were limited and they could not account the multiple objects of varying sizes in the image. With the introduction of convolutional neural networks (CNN), feature extraction became much more powerful, thanks to high learning and expressive capacity of CNNs. Since feature extraction is the most compute intensive step, it is important that it is done efficiently. Through the different object detectors presented in the subsequent chapters it can be seen that an efficient method for feature extraction and substantially speed-up the detector and forms the basis of **lightweight object detection**.

Classification: The last step in detection is classification. It involves using the features extracted in the previous step to generate the class scores and bounding-box offsets. Most deep learning based object detector use two parallel Softmax layers [12] to generate the class score and the bounding-box offsets respectively.

CNNs were used for the first time for object detection in 2012. On the PASCAL VOC [8] benchmark dataset, they improved the detection accuracy by 30% over the previous methods based on the HSC based object detectors. This was due to the following advantages. First, these methods used sliding window technique to generate object proposal which gives a large number of redundant proposals. Second, the manually computed feature tend to have a big semantic gap with the input image. Third, deep CNNs can learn more complex feature from an image. Although the better learning ability was due much larger number of parameter which make the process of object detection compute expensive and slow.

R-CNN, region based CNN is the first method to used CNN to obtain state-of-the-art accuracy. Rather that the naive sliding window approach it uses selective search [38] to generate region proposals over the input image. Each of these region proposals are then fed into a CNN for classification and the bounding-box offsets are generated through regression. Due its novel approach and superior accuracy R-CNN led to a new class of object detectors like Fast R-CNN, Faster R-CNN, which incrementally improved on both detection accuracy and performance.

Another class of detectors are **proposal-free**. YOLO [29] and SSD [26] are examples of proposal-free detectors. YOLO generates a fixed grid on the input image and performs regression on it. This regression is done using a CNN which generates an objectness score for each of the boxes on the grid. Objectness score is how likely a box contains a class of object. Single-shot mulit-box detector or SSD, uses anchor boxes and multi-scale feature maps for object detection. It is faster that most of the previous state-of-the-art detectors. Therefore, most of the lightweight detectors borrow design principles from SSD.

State-of-the-art detectors such as Faster R-CNN, YOLO, SSD have large number of parameters making them compute intensive and slow. Lightweight detectors are an attempt to address this problem by building detectors with leaner model which yield similar or slightly lower accuracy at a considerably lower compute cost. This makes lightweight detectors suitable for devices with limited resources, such as laptops and smartphones.

In this project, we have explored the different object detectors that use deep CNN for feature extraction and classification. The advantages and shortcomings of each of these methods have been presented. Then, the recent developments in lightweight detectors has been examined. Recent development in CNNs design and training has been investigated and used to build a model a more robust and accurate lightweight detector while keep the parameter count in check. Finally, the proposed inter-channel dependency block (ICDB) has been explained, and how the ICDB block improves the information flow and ultimately the detector accuracy. Experiments on the benchmark datasets show that the proposed method performs better compared to the previous lightweight methods with respect to accuracy. Although the model also has marginally more parameters compared to it's counterparts.

1.1 Report Organization

Chapter 2, explains some of the fundamentals concepts of deep learning and CNNs that we frequently come across during object detection. Chapter 3, presents the all the state-of-the-art object detectors. In chapter 4, the lightweight detectors have been presented. In chapter 5, the proposed method and the corresponding implementation has been discussed in detail. In chapter 6, the different experiments and their results on the selected benchmark datasets has been presented. These experiments show that the proposed method scales well do different datasets and also give better detection accuracy compared to its lightweight counterparts.

Chapter 2

Background

In this chapter, the basics of deep learning and CNNs has been discussed. Beginning with a short introduction to deep learning concepts.Then, VGG16 a commonly used CNN in object detectors for feature extraction has been used to explain different layers in a CNN. Section 2.3 talks about CNN training using Stochastic Gradient Descent (SGD). Section 2.4, enumerates some of the many benefits of CNNs in computer vision tasks. The chapter ends with the definitions of some important terms, used for measuring the accuracy of object detectors.

2.1 Deep Learning

Pitts *et. al* [28] presented a paper on neural networks in 1947. They used mathematical equations to represent the learning process of a neuron. The following years saw slow research and development related to neural networks. Three decades later, in 1980, after the development of the back-propagation technique, neural network were again a hot topic of research. This new training technique made it much faster to train neural network, making them feasible for practical applications. More recently, Hinton *et. al* [15] demonstrated that deep neural network are capable at building better speech recognition models than Hidden Markow Models (HMMs). Since then, there has been a rapid growth in the field of deep learning. Some of the major breakthroughs are as follows:

- Free and easy access to high-quality datasets, ImageNet [4], Common Objects in Context (COCO) [24] and PASCAL VOC [7]. These dataset made it easier to perform deep learning research.
- Powerful CPUs and GPUs made it possible to build bigger and powerful models.
- Distributed training techniques that allowed models to be trained in parallel across multiple systems.
- Development of Auto-encoders [5], improved neural network training by improving the initialization of weights.
- Batch Normalization (BN) was introduced in 2015, allowed the training of deeper neural network, by reducing extreme variations in the network weights. These days, BN is used in almost all neural networks as well as convolutional neural networks.
- Overfitting happens when a network is very closely trained on the train set and performs poorly on new inputs. It can occur due to small number of training examples or when the model trains for too many iterations. Overfitting can be remedied by regularization [1]. Regularization modifies the loss function by adding a penalty term. This term reduces large variations in the weights by reducing the value of the gradient term. Dropout is another technique to avoid overfitting. It involves fixing the weights of certain weights during training.
- Development of more powerful CNNs like like MobileNet [16], ResNet [14], GoogLeNet [37], AlexNet [20], gave rise to better models for detection and classification.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNN) have been used to successfully build many end-to-end models for image classification and object detection. These models automatically extract features and perform classification without the need for any manual intervention. Using the example of VGG16 the different layers in a CNN has been explained here.

The architecture of VGG16 has been shown in the figure 2.1. It uses 13 convolutional layers, three fully-connected layers, 3 max pool layers and a softmax output layer.

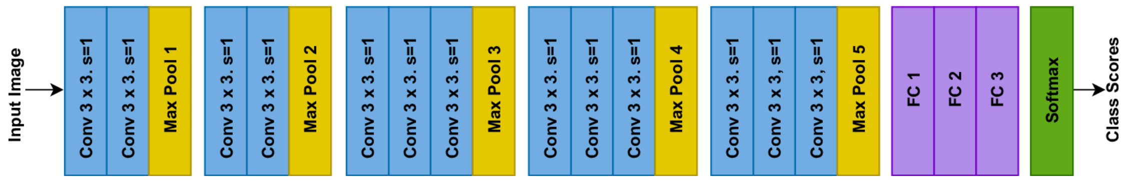


FIGURE 2.1: VGG16 architecture.

Convolutional layer : Consist of N number of filters of $H \times W$ dimensions that perform convolutions over the input image or feature map. The size of the filter is called its **Receptive Field**. Each of the filters contain C channels, same as the input channels. The filters perform matrix multiplication with the corresponding values in the receptive field and store the result in a new matrix, which is output feature map.

Using the sliding window technique, the filter goes through the entire input map (image or feature map). The output is a new map that can be used as input to other convolution layers. It can also be subjected to prediction or classification. For example, consider a 3×3 filter and a 32×32 input map. The input has three channels. For convolution, the filter will slide over the entire image, performing matrix multiplications. The number of blocks the filter moves in each step is called **stride**. The size of the output map depends on the filter size and strides.

Consider an input RBG image of size, 32×32, and 12 filters of size, 3×3 with three channels each. With stride $= 1$, each filter will produce an output map of size, 32×32 with 3 channels each. The outputs of all the filters combined will give the final feature map, which will have $12 * 3 = 36$ channels.

Pooling Layer : It is used to downsample the feature maps or transform them into 1D feature vectors. There are three types of pooling:

- Global Pooling: A single value from each channel of the feature map is selected.
- Average Pooling: The average of values in a fixed size window is selected.
- Max Pooling: The maximum value in a fixed size window is selected.

Sometimes global pooling and average pooling may be combined to form **global average pooling**.

Full Connected Layer : Contain 1D feature vector that use all the values of the previous layer. FC layers transform feature maps to 1D feature vectors used to predict class scores.

Softmax Layer: A special type of FC layer, without weights (parameters), used to output class scores by converting the values in the feature vectors to probabilities. Using the softmax equation:

$$\sigma(i) = \frac{e^{Z_i}}{\sum_{j=1..K} e^{Z_j}} \tag{2.1}$$

Here the number of classes $= K$ and Z_i is the value i^{th} value in the feature vector input into into the softmax layer.

2.2.1 Training

The Back-propagation algorithm [1] is used to train neural networks and CNNs. It consists of two step, forward pass and backward pass, which is done for multiple iterations, until the loss value is small or lower than a selected threshold. An input image going through all the layer of a CNN continues a forward pass. It ends with the calculation of loss, using the **loss function**. The loss value is the propagated backwards and the gradient of the loss with weights are calculated using Stochastic gradient descent (SGD). This gradient is then used to compute the new weights.

The inital weights in the CNN filter can be set randomly or by using an Auto-encoder. They can also be initialized by pre-training a CNN classifier on a image dataset. The *Xavier* method for initialization involves using random values in the range of the **activation / gating function**. Activation functions are used to introduce non-linearity on the function learned by a model. Sigmoid and ReLU are examples of activation functions. Acivation function are used to add non-linearity to the function learned by the model.

2.3 Some Terms and Definitions

Object detectors mainly use four metrics for evaluation, IoU, Precision, Recall and mAP. This the following section, the definitions of these metrics has been discussed.

2.3.1 Intersection Over Union (IOU)

IoU is used to calculate the accuracy of a predicted bounding box. It measures the overlap between two bounding boxes. If the ground-truth box is given by (B_g) and the predicted box is (B_{pred}), then IoU for for these two bounding boxes is given by:

2.3.2 Precision and Recall

In object detection (or machine learning in general), a correct prediction is called **True Positive or TP**. When the detector predicts an object that does not exists, we get **False Positive or FP**. When a object present in the ground-truth is not present in the prediction, it is called a . **False Negative or FN**.

Precision is defined as the fraction (or percentage) of all true positive prediction out of all predictions.

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

Recall is all the true positive predictions divided by all ground truth predictions. Number of GT predictions = TP + FN.

$$Recall = \frac{TP}{groundtruth} = \frac{TP}{TP + FN} \tag{2.3}$$

2.3.3 Average Precision

In simple words, the area under the Precision vs Recall (PR) curve is called Average Precision. Unfortunately, it is difficult to calculate the area under the PR curve, as it consists of multiple variations and lines crossing over each other. To simplify this, the precision value $(p_i(r_i))$ for a given recall (r_i) is replaced with the max precision for a recall $\geq r_i$. This converts the zig-zag lines in the PR curve to orthogonal lines, making it a combination of rectangles of different dimensions. For recall $= r_i$, let the interpolated precision be, $p_{interp}(r_i)$, AP is given by equations 2.4, 2.5.

$$AP = \sum_{i=1..N} p_{intrp}(r_i).(r_i - r_{i-1}) \tag{2.4}$$

$$p_{intrp}(r_i) = max_{\widehat{r} \geq r_i}(p(\widehat{r})) \tag{2.5}$$

Mean Average Precision (mAP):

For a detector the AP for all the distinct classes in evaluated. The average of all these APs is mAP. It is a very commonly used metric for comparing detectors. All benchmarking dataset provide their own evaluation scripts to calculate mAP using the predicted bounding-box coordinates and class labels. For all the experiments in this project, mAP has been computed to determine the accuracy of the model. Fo K classes, mAP is given by:

$$mAP = \frac{1}{K} \sum_{i=1..K} AP_i \tag{2.6}$$

Chapter 3

Object Detection Methods

Object detection comprises of classification and localization of objects in images and videos. For practical purposes, the output images are labelled with bounding boxes and the object classes. This chapter presents the existing state-of-the-art detectors. These detectors have been classified between two classes based on the detection approach employed by the method.

Starting with R-CNN [12], most of the state-of-the-art detectors have been discussed in the order in which they were published. For each detector, the key changes and contributions have been discussed. Along with a list of drawbacks.

3.1 Deep Learning Based Object Detectors

Most object detectors that came out in the past 7-8 year have used CNNs for feature extraction. These feature are then subjected to a set of prediction layers that generate the bounding-box offsets and the class scores. These modern object detectors can be classified into two types:

- Region proposal based.
- Regression/Classification based.

Region Proposal Based:

These detectors have a two stage detection process. The first step involves the generation

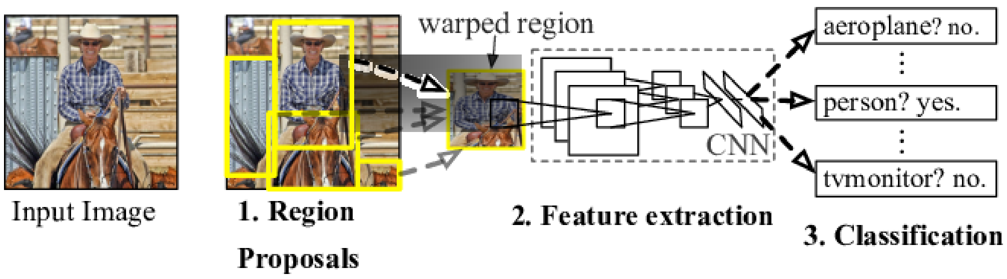


FIGURE 3.1: R-CNN architecture [12].

of all region proposals. This steps involves generating multiple bounding boxes for all possible objects in the image. These bounding box coordinates along with the input images fed into a feature extractor CNN. The feature maps generated by the CNN are then transformed into 1D feature vectors to get class scores and bounding box coordinates. R-CNN [12], Fast R-CNN [11], Faster R-CNN [31], SPP-net [13], R-FCN [2], FPN [23] used region proposals.

Regression/Classification based:
Detectors that do not use any separated step for region proposals, instead use a combined step to generate bounding boxes and class scores. YOLO [30] and SSD[25] are the popular regression/classification based detector.

3.1.1 Region Proposal Based Framework

3.1.1.1 R-CNN

R-CNN is one of the first detectors to use deep CNNs for object detection. It attained an overall accuracy of 53 % mAP on the PASCAL VOC 07 dataset. This was 30% more the previous best, a Histogram Sparse Codecs (HSC) detector[32]. Due it promising results, R-CNN started a new era of object of deep CNN based object detectors.

The design of R-CNN is shown in the figure 3.1. It has three major parts:

- **Region proposal generation:** Using selective search [38], multiple bounding boxes are generated for all the objects in the input image. For R-CNN, 2K proposals with the higher objectness score are considered. Selective search uses hierarchical grouping and several saliency measure to score the boxes. It is much faster than a naive brute-force approach of generating all possible boxes.

- **Feature Extraction:** The input image is cropped and wrapped to the region proposals and passed through the Krizhevsky's CNN [20]. Krizhevsky's CNN has 5 convolution layers and 2 fully-connected (FC) layers. The output of the CNN is a 4096-D feature vector.
- **Classification:** For this part, first an SVM classifier is trained on the dataset. Then the output of the feature extraction step is passes into the classifier to generate the class scores. The bounding box regressor is used to predict the class scores. The bounding box regressor is a neural network with one FC layer and softmax output.

The major shorcomings of R-CNN are as follows:

- It has a very cumbersome training process. There are four different modules in the model, and each required separate training. This problem is addressed in the upcoming detectors.
- The input size of the feature extractor CNN is fixes due to the presence of FC layers. Therefore, each region proposal has to pass through the CNN one at a time. This step is very time consuming as 2k region proposal have to be processed.
- Despite having a high recall rate, SS generated redundant proposals and is time consuming. It the more recent detector we see that the region proposal are handle by a CNN which shares computations with the feature extractor.

3.1.1.2 SPP-net

The input size of R-CNN is limited because of FC layers. The feature extractor required the input to be cropped and wrapped on the region proposal to a fixed size. This may cause distortions to the objects inside the region proposal, leading to information loss. Subsequently causing the detection accuracy to drop. More prominently in images that contain multiple objects of different sizes. Using spatial pyramid pooling (SPP), SPP-net [13] attempts to alleviate these problems.

Three level SPP is shown in figure 3.2. SPP uses multiple pooling layers to transform the feature maps into feature vectors for the FC layers. Consider a feature map with

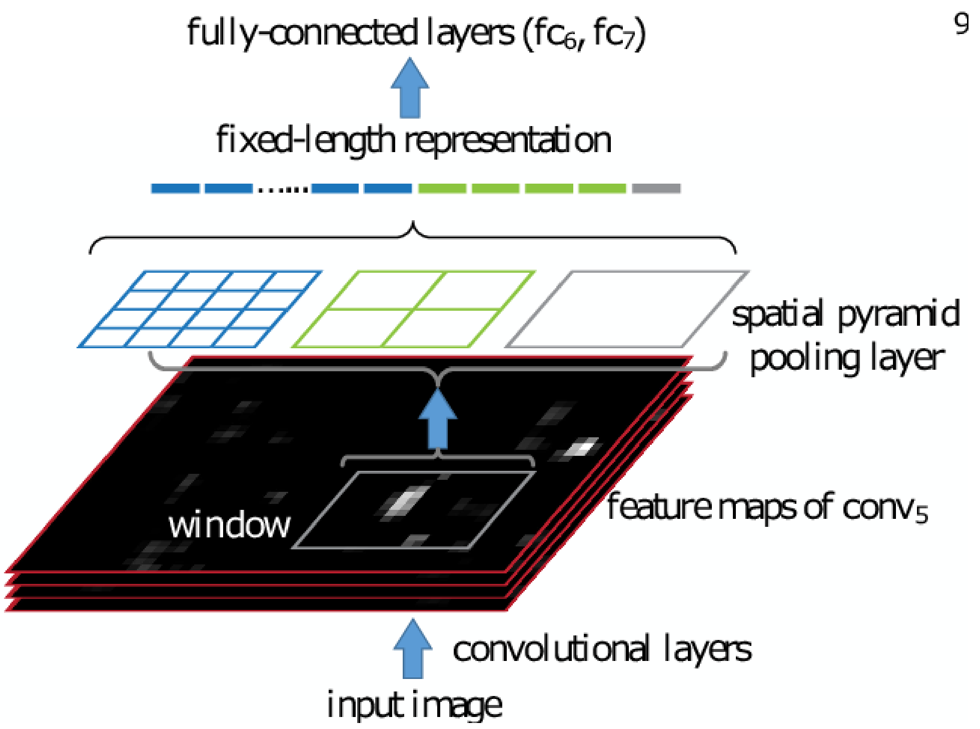

9
fully-connected layers (fc_6, fc_7)
fixed-length representation
spatial pyramid pooling layer
window
feature maps of $conv_5$
convolutional layers
input image

FIGURE 3.2: 3 level spatial pyramid pooling used in the SPP-net detector. [13].

256 channels. Using a 3-level SPP, the maps are pooled in a feature vector of length 256 taking one value per channel. Using 4 values per channel, they are pooled into a 4×256 length feature vector. Finally, taking 16 values from each channel, they are pooled into a 16×256 feature vector. The three feature vector are then concatenated. Rest of the layer are same as R-CNN.

SPP-net uses the Krizhevsky's CNN [20] to obtain feature maps. Feature maps of different sizes are transformed into fixed length feature vectors using the SPP layer. SPP-net processes all the region proposal in one pass, making it much faster R-CNN. By avoiding information loss and object distortions cause by cropping and wrapping the input image, it increases detection accuracy.

3.1.1.3 Fast R-CNN

SPP-net suffered from some major drawbacks. Like R-CNN it had a cumbersome, multi-stage pipeline. This required additional parameter storage and lead to slow training and detection speed. The during training, the layers before the the spatial pyramid pooling layer could not be fine-tuned, causing a drop in accuracy. Faster R-CNN [11] solves

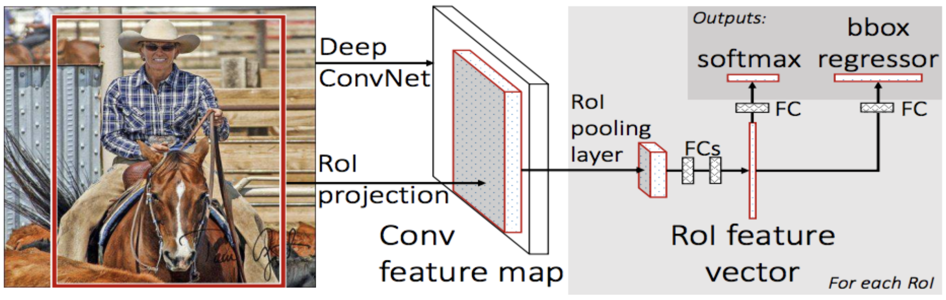


FIGURE 3.3: Fast R-CNN architecture.[31].

these problems by creating a new Region of Interest (RoI) pooling layer and multi-task loss.

RoI Pooling:
This new layer was added to fixed dimension feature maps from region proposals of arbitrary sizes. It was done using max pooling on the region of the feature maps inside a region proposal (also RoI).

Multi-task Loss:
The bounding box offsets and the class score were predicted using two parallel output layers. Let $p = (p_0, p_1, ..p_{K-1})$ be the probabilities for the K classes. The bounding box offsets are, $l^i = (l^i_a, l^i_b, l^i_c, l^i_d)$, where $i = 0..K$. Fast R-CNN uses a novel loss function that combined both these outputs, called the multi-task loss.

Multi-task loss and RoI pooling allowed for end-to-end training eliminating the problem of CNN fine-tuning. Since there was no need for individual modules, the model size also reduced. These modification allowed Fast R-CNN to be more accurate and faster than SPP-net.

Some of the major shortcomings of Fast R-CNN are as follows:

- The RoIs are forward passed one at a time, which made it slow.
- Region proposal generation using selective search was a bottleneck.

3.1.1.4 Faster R-CNN

Faster R-CNN introduced a Region proposal network (RPN) to address the problems with RoI pooling and SS based region proposal generation. It used VGGNet as the feature extractor CNN. The RPN was another CNN that would now generate the region proposals. Feature maps are shared between the feature extractor and RPN. It can predict the class scores and bounding box offsets for all the RoIs simultaneously.

RPN:
It is used to rank the pre-defined anchor boxes [11]. The output of RPN is probability of how likely an anchor box contains an object. If this score is higher that a threshold, the anchor box is considered as a region proposal. Moreover, since there are no FC layer, it can process all the anchor boxes on the feature map at the same time.

Feature Extractor:
Similar to Fast R-CNN, VGG16 has been used as the feature extractor. The RoI pooling layer transforms the output feature maps to feature maps with region proposals. Which goes into two parallel outputs.

Training:
It uses a novel, 4-step alternating training, for fast end-to-end training. These 4 steps are:

- Fine-tuning of the RPN using ImageNet.
- Fine-tuning of the detection network using ImageNet.
- The RPN and the detection network are then combined and the RPN is selectively trained by fixing the parameters in the detection network.
- Finally, the other layer of the detection network are fine-tuned keeping the CNN layers fixes.

On PASCAL VOC 07 and 12, Faster R-CNN obtains an overall 73.2 % mAP and a speed of 5FPS on the Nvidia Titan X GPU.

3.1.1.5 R-FCN

With respect to the RoI pooling layer, Fast R-CNN, SPP-net and Faster R-CNN can be divided into two components. First, is the full-convolution layer with shared feature maps and second is the detection network. This design was inspired by the pioneering

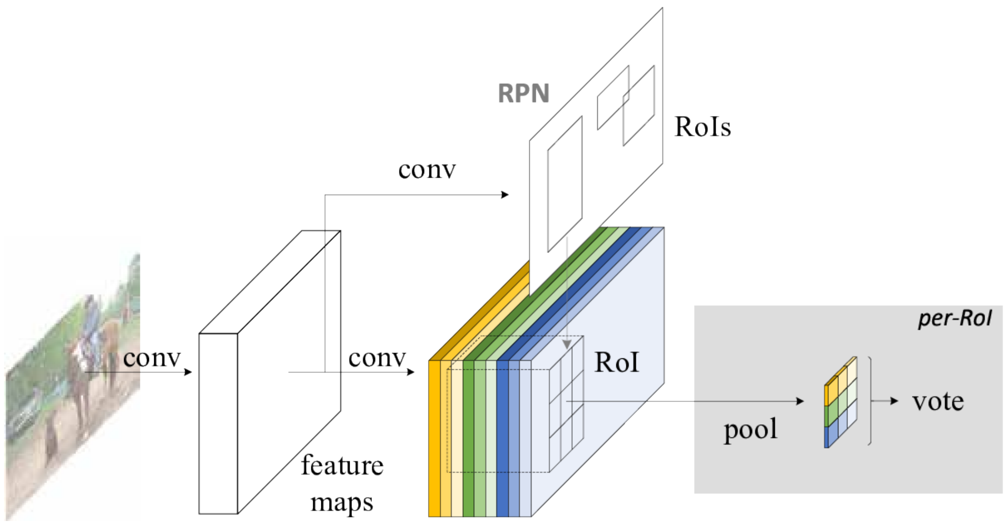


FIGURE 3.4: R-FCN architecture. [2].

CNNs, AlexNet [20] and VGGNet [13], which also contain conv layers followed by FC layers.

In FCN [34], R-FCN uses a fully-convolutionary shared network. Figure 3.4 shows the R-FCN architecture. It consists of a fully-convolutionary network with shared feature map with a regional proposal network (RPN). It uses positive-sensitive score maps to solve the translation invariance problem. The entire network is trained to end-to-end.

For functionality extraction, R-FCN uses ResNet-101 [14]. ResNet-101 is a fully-fledged network of 101 layers that also include skips for improving the training of deep conv networks. When trained on the PASCAL VOC 07 + 12 train set, it achieves an accuracy of 77.7 mAP. More importantly, it take about 0.17s per picture.

3.1.1.6 FPN: Feature Pyramid Network

The ideas behind the pyramid network feature (FPN) for object detection are shown in figure 3.5. In the past [9], [13] used many featured image-pyramids (figure 3.5(a)). The main objective is for the detection of various objects of different sizes to improve the invariance of the scale. This process unfortunately requires time and memory for extensive training. In order to avoid this, certain techniques are used to create a single feature map (figure 3.5(b)).

A feature pyramid is essentially made of feature maps of different sizes. In conventional CNNs, feature maps of different scales are generated at different layers, and these

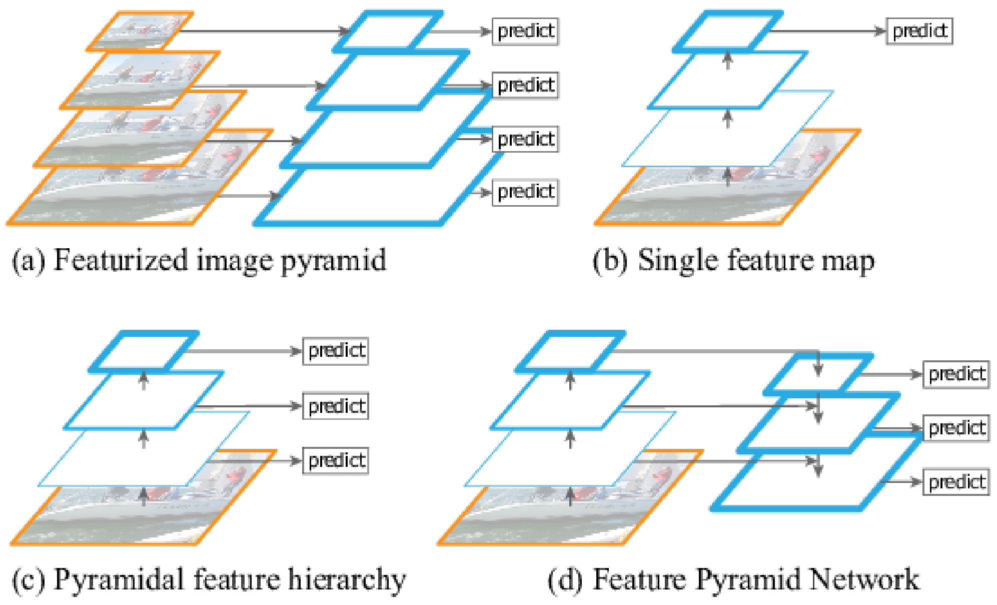


FIGURE 3.5: Structure of FPN [23]. (a) Feature pyramid using an image pyramid. (b) Feature pyramid using just a single feature map. (c) Variant of (b) using the entire feature pyramid for prediction. (D) FPN combines (b) and (c).

layers have semantic gaps between them. FPN combines these multi-scale feature maps to reduce this semantic gap.

The **bottom-up path** is the forward pass of the backbone convnet, which produces feature hierarchy through a 2 step down sample of the corresponding maps. The layers with the same size of output maps are grouped into the same network stage and the output of the last layer of each stage is chosen as the reference set of feature maps to create the following top-down path.

In the top-down paths, the maps from the later stages are upsampled and combined with the similar sized maps from the bottom-up path using lateral connections. The channel size of the upsampled map is reduced using a 1×1 convolution layer. The maps are merged using simple element-wise addition. The feature pyramid allows rich semantics to be extracted from all levels and is trained with all the scales, thus ensuring a state-of-the- art detection without sacrificing speed and memory. The FPN can be used with different feature extractors and can be used at different stages of the detection process.

3.1.2 Regression/Classification Based Frameworks

Two stage detectors like Faster R-CNN [31] use multi-step alternating training to generate feature maps. This multi-step process causes a bottleneck and slows the process. Single-stage regression based detectors perform prediction directly in a set of anchor boxes and the input image, without any need to explicitly generate region proposals. This makes them faster and suitable for real-time applications. The most popular single stage detectors, You only look once (YOLO) [29] and and Single Shot MultiBox Detector (SSD) [26] have been discussed here.

3.1.2.1 YOLO

The first detector to use the single-stage strategy for detection is YOLO. The input image is divided into a $N \times N$ grid. The prediction is done for each grid cell to generate bounding boxes and scores. The feature map generated by the CNN is used to perform the predictions.

YOLO can perform at 45FPS, a smaller version of YOLO, called Fast YOLO can do detection at 150FPS.

3.1.2.2 SSD

YOLO suffered from low accuracy due the absence of batch normalization, deep CNNs, anchor boxes and multi-scale feature maps. Single Shot Multibox Detector (SSD) addressed these issues and also introduced custom conv layers on top of VGG16.

Using these latest developments in CNNs, it more accurate than Faster R-CNN while being upto $3\times$ faster. It is faster and more accurate that YOLO as well. Unfortunately, SSD struggles when working images containing very small objects.

Chapter 4

Lightweight Object Detection Methods

Faster R-CNN, YOLO and SSD are full-size state-of-the-art detection methods. All these methods use powerful convolutional neural networks to automatically extract features from the input image. Although this high accuracy is due to the large model size and more number of parameters. This means that they require large compute and memory resources. Moreover they require long training time, also making them expensive in terms of power usage. These requirements limit the application of such 'full-size' detectors in low-end battery powered devices.

Lightweight detectors address these concerns by using much smaller models to perform detection. This makes them much faster to train and test compared to the full-size detectors. Taking advantage of recent developments in CNN design and training these methods can have as low as 15 million parameters, 200× the full-size detector Faster R-CNN. Tiny-YOLO, a smaller version of YOLO has only 15 million parameters and perform as fast as 200fps on the PASCAL VOC 07 [8] dataset.

In this chapter, the most recent state-of-the-art lightweight detectors have been presented. Instead of getting into the details of each of the methods the unique contributions of these methods have been presented. Finally, the proposed method has been presented that used the novel inter-channel dependency block (ICDB) block.

4.1 Tiny-YOLO

Tiny-YOLO [30] is the concise version of YOLO. It uses the DarkNet Reference Model[1], instead on the 24 layer deep CNN used on YOLO. This makes Tiny-YOLO much faster compared to YOLO but with a considerable loss in accuracy.

The DarkNet Reference Model, is a full-convolutional network with 15 Layers. A full-convolutional network means no full-connected (FC) layers. Having no FC layer improves the learning ability of a model. DarkNet has $10\times$ lower parameters compared to Krizchevsky's CNN [20], used in R-CNN [12].

Tiny-YOLO obtains 57.1 mAP accuracy on the PASCAL VOC 07 dataset, with a test speed of 7ms per image. The model has approximately, 6.97 billion parameters.

4.2 MobileNet-SSD

MobileNet [16] is the most widely used deep CNN for computer vision applications in mobile devices, hence the name. MobileNet introduced a new highly-efficient alternative to the traditional convolution operation, called Depthwise Separable (DS) convolutions. It also introduced two hyper-parameters to control model size. These contributions have been discussed in the following sections, along the architecture and training process. It is seen that using MobileNet as the feature extractor in SSD considerably reduces the model size and boosts performance with a small drop in accuracy.

4.2.1 Depthwise Separable Convolutions

Depthwise separable convolutions factor the traditional convolution operation into two efficient steps: (i) Depthwise convolutions, (ii) Pointwise convolutions.

Depthwise Convolutions:
They are shallow layers where the filter contains channels equal to the number of input channels. Each channel in the filter is responsible for convolution with the corresponding input channel. This means that the output feature map contains channels equal to the input image. In the traditional convolution operation, there are N filters with C channels

[1]https://pjreddie.com/darknet/imagenet/reference

each, where C = input channels. These operations have been explained in more detail in chapter 5, "Implementation".

Pointwise Convolutions:
Pointwise convolutions are used to add depth to the feature maps produced by the depthwise convolutions. If number of input channels = C, then pointwise convolution required N filters of dimension 1×1 with C channels each. The output generated by each of these filters are then stacked together to produce the final feature map. The dimensions of this feature map is same the one that would be produced by traditional convolution operation using N D filters with C channels each.

4.2.2 Architecture and Training

MobileNet consists of 13 depthwise convolution layers, one FC layer, one average pooling and softmax output. Feature maps are downsampled by pooling with stride = 2. Every depthwise convolution layer is followed by a batch normalization (BN) [18] layer and ReLU activation (gating function). BN normalizes the scale variations in the outputs generated by the convolutional layers. This allows for a higher value of the learning rate, faster training and reduces the need for regularization steps like dropout.

Since, convolution operation is basically a large number of matrix multiplications. Making these operations more efficient, can speed-up training. So, using pointwise convolutions with 1×1 filters reduces the number of matrix multiplications.

For training, MobileNet used an adaptive learning rate computed using the RMSprop [33] technique. RMSprop allows for faster convergence. The weight-decay for MobileNet is small because of a smaller model. The official implementation of MobileNet has been used for our experiments.

4.2.3 Width Multiplier and Resolution Multiplier

Width-multiplier, α was introduced to control the model size by reducing the width of each layer when required, $\alpha \in (0, 1]$. Consider an input channel with C channels and width multiplier = α, the number of input channels reduce to αM.

Resolution Multiplier, ρ was used to reduce model size by reducing the resolution of the input image and each of the feature maps by the convolution layers. Consider an input image of dimensions, $D \times D$, using the ρ the input, the input resolution reduces to $\rho D \times D$, as $\rho \in (0, 1]$.

MobileNet-SSD is a detector that uses MobileNet as the feature extractor, instead of the original VGG16. It has an accuracy of 68 mAP on the PASCAL VOC 07 dataset. This is almost 7% lower than the accuracy of SSD with VGG16. On the other hand, SSD has over 5× more parameter compared to MobileNet-SSD.

4.3 DSOD

Full-size detectors like Faster R-CNN, YOLO, SSD require pre-training, done using the ImageNet dataset [20]. Training these models from scratch on the detection dataset leads to lower accuracy. Moreover, the pre-training step induces a learning bias, caused by differences in the class distribution, loss function and the outputs of the classifier and the detector. Fine-tuning the detectors reduces this bias, but only to a certain extent. The process of per-training adds an extra overhead to the training process. Deeply Supervised Object Detector (DSOD) uses dense connections [35] along a novel training technique to create object detectors from scratch without the need for pre-training and fine-tuning. It is one of the first detectors to obtain state-of-the-art accuracy while performing training from scratch.

DSOD contributions are as follows:

- First state-of-the-art detector to train from scratch.
- Introduced a set of design principles that gives detector that perform better than the previous state-of-the-art detectors.

4.3.1 Architecture of DSOD

DSOD is a single-stage detector and does not require region proposals. Similar to SSD, it concatenates multi-scale feature maps and performs the predictions on it. The structure of DSOD can be divided into two parts:

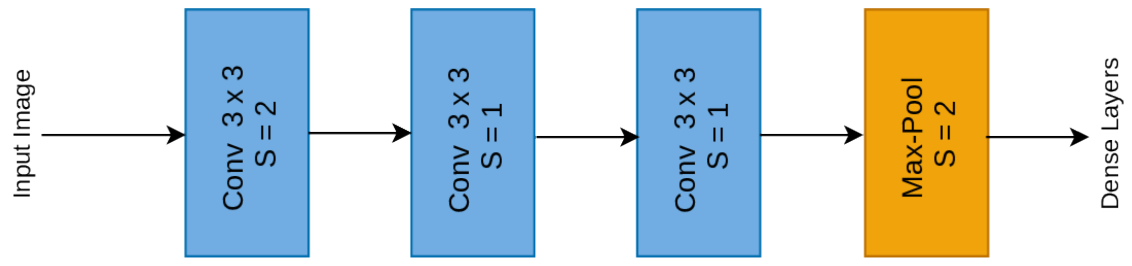


FIGURE 4.1: Stem Block in DSOD. 'S' = stride.

- **Backbone Network:** It consists of a stem block, 4 dense blocks, with 4 transition layers between them. Two of these transition layers with pooling and the other without pooling layers. The stem block consists of 3 convolutional layers and a max pooling layer. The dense blocks contains 5 densely connected conv layers. The transition layer are placed after each of the dense blocks. The backbone network is also called the back-end sub-network.

- **Prediction Network:** It consists of multi-scale feature map, like SSD [26]. These feature maps are concatenated then passed through two softmax layers to predict the bounding box offsets and the class score respectively. It is also referred as the front-end sub-network.

4.3.2 Training DSOD

In the DSOD the authors present three design facets that can be combined to create deeply supervised networks that can be trained from scratch. These are as follows:

- **Proposal-free:** Through experiments the authors show that only detectors like SSD and YOLO, that do not require object proposals, can converge without pre-training. This is cause by the RoI pooling layer. It generates feature maps for the region proposals but reduces the accuracy of the gradients propagated to the convolution layer.

- **Stem Block:** A new stem block containing three conv layer and one max-pool layer has been used inn DSOD. The DSOD stem block is shown in the figure 4.1. By demonstrating better detection accuracy, the authors show that the use of stem block improve the information flow from the input to the dense blocks. It improve the accuracy of DSOD by 1.8 % mAP compared to DSOD without the stem block.

- **Deep Supervision:** The idea here is to have a loss function that helps the output as well as the hidden layers. Dense blocks [17] are used to achieve this. Where every layer is connected all its previous layers. These extra connection improve supervision while using only one loss function. These block imporve both accuracy and performance of the model. Since, each layer in these blocks only contains typically 12 layers, the number of parameters are also reduced.

The official DSOD implementation github-DSOD has been used for all the experiments. The training steps remain the same as for SSD with a few changes. L2 regularization [35] has been applied to all the feature maps used in the output.

4.4 Tiny-DSOD

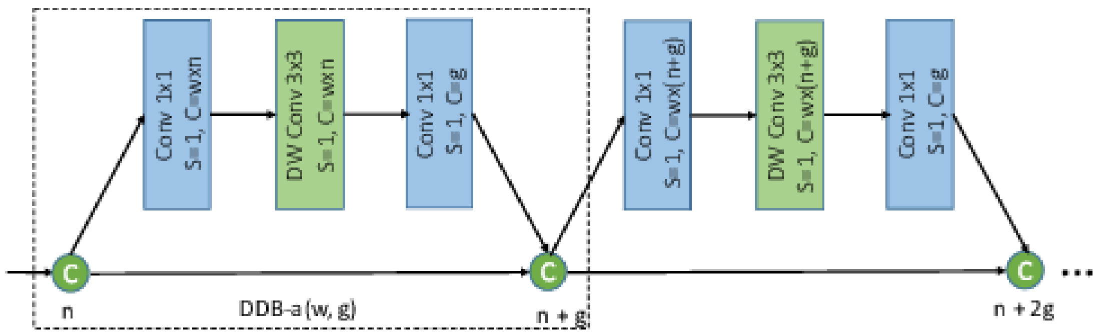
Tiny-DSOD is the most recent lightweight detector with state-of-the-art accuracy. It is the lighter version of DSOD. Similar to DSOD, it consists of the backend network and the front-end network. The backend network is use the compute efficient depthwise convolutions inside dense blocks, forming the depthwise dense block (DDB). The front-end network uses a lightweight version of FPN called L-FPN. It also uses depthwise convolutions instead of regular convolutions.

4.4.1 Architecture of Tiny-DSOD

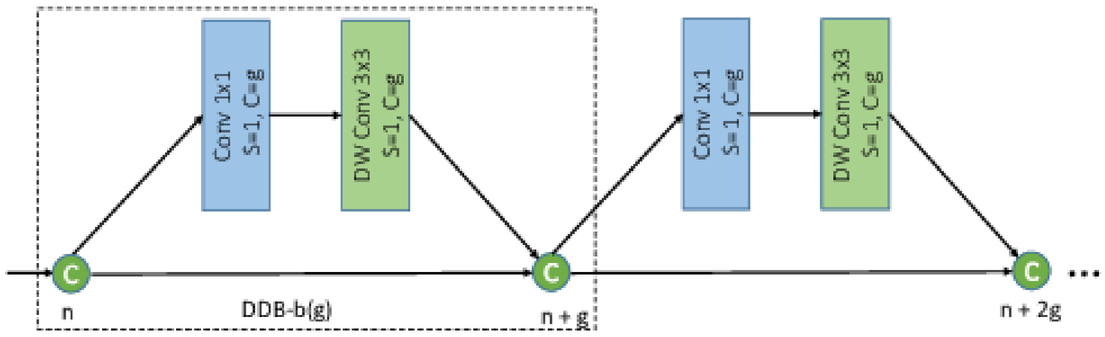
4.4.1.1 Backbone Sub-network

The backbone in Tiny-DSOD consist of dense blocks with compute efficient depthwise convolutions. This new kind of dense block is called depth-wise dense block (DDB). The authors proposed dense blocks of two types: (i) DDB-a and (ii) DDB-b.

DDB-a consists of a hyper-parameter w, called model capacity and another one called growth rate, g. An number of channels in the input is increased w times in the first convolution layer. The output of this layer is fed into depthwise convolution layer followed by a pointwise layer. The input and the output feature maps are then concatenated to form the final output of the block. g is the number of channels in the pointwise convolution layer.



(a) DDB-a



(b) DDB-b

FIGURE 4.2: The two types of dense blocks used in Tiny-DSOD. Here 'S' stands for stride and 'C' for number of channels. [21].

The total space complexity of the N DDB-a blocks is $O(N^3g^2)$ where g is the growth rate. Therefore model complexity increases quickly with increase in N. Also, the concatenation of the feature maps generated by two adjacent DDB-a units introduce some redundant parameters.

DDB-b addresses some of these issues by making some design modifications in DDB-a. First, the model capacity hyper-parameters is omitted from the block. This makes the number of channels in each of feature maps equals to g. Second, the pointwise convolution layer after the depthwise layer is removed to get rid of redundant parameters. This also reduces the overall space complexity of the block to $O(N^2g^2)$.

Hence, DDB-b block has a lower space complexity and is used as the basic building block in the backend network in Tiny-DSOD.

4.4.2 Front-end Network

A lightweight version of FPN [23] has been used to create the front-end network in Tiny-DSOD. This structure is called L-FPN. Similar to FPN is contains two pathways, a downsampling pathway and an upsampling pathway, these pathways are inter-connected using lateral connection to share feature maps. The main advantage of using an FPN is that it combines small, feature rich maps with larger, shallower maps to obtain semantically rich feature representations.

The upsampling pathways are made of upsampling block consisting of bi-linear interpolation followed by depthwise convolution. The downsampling block consists of two parallel pathways. First one contains a pointwise convolution layer followed by a max pooling layer. The second contains a pointwise layer followed by a depthwise convolution layer.

Tiny-DSOD achieves better accuracy compared to previously proposed lightweight detectors. More importantly, it does so while keeping the parameter count low, by using novel blocks like DDB-b and L-FPN. The official Tiny-DSOD implementation has been used as the baseline for our method. The proposed method adds new layers to Tiny-DSOD to improve the overall detection accuracy. In the next chapter, the proposed method has been explained in detail.

Chapter 5

Implementation Details

In the previous chapters we saw most of the state-of-the-art full-size and lightweight detectors. Lightweight detectors such as MobileNet-SSD and Tiny-DSOD introduced some promising techniques such as depthwise convolutions, dense blocks and lightweight FPN to obtain higher accuracy, similar to full-size detectors. Drawing from the success of these techniques, a new lightweight detector has been proposed that uses a novel inter channel dependency block (ICDB). The subsequent section present the details about our proposed method and the implementation.

5.1 Caffe

The official Caffe [19] implementation of Tiny-DSOD[1] has been used as the baseline for implementing the proposed method. Caffe is an open-source deep learning framework developed by the Berkeley Artificial Intelligence Research (BAIR) group. It has a large community of contributors and free access, a lot of academic research in deep learning is done using Caffe. Popular detectors like SSD, DSOD, Tiny-DSOD were developed using Caffe. Some of the major advantages of Caffe are as follows:

- **Nvidia CUDA** : Caffe has support for most of the recent Nvidia GPUs that use the Nvidia CUDA library to perform highly efficient arithmetic operations. Since these operations form the basis of most deep learning models, having GPU support is beneficial for optimal performance of object detectors.

[1]https://github.com/lyxok1/Tiny-DSOD

- **ATLAS :** Automatically Tuned Linear Algebra Software, is an open-source library which provides memory and compute efficient implementation of common linear algebra operation such as vector and matrix multiplication. The implementation more efficient hardware utilization, without the need to write common sub-routines in low-level code.

Apart from these major advantage, Caffe uses C++ boost library and OpenCV to allow easy access to several image processing methods.

5.2 Proposed Model

The proposed detector consists of two parts:

- **Back-end Network:** It is made of dense blocks like in Tiny-DSOD and the proposed **Inter-channel dependency block (ICDB)**.
- **Front-end Network:** This part of the network uses a lightweight feature pyramid network to generate multi-scale feature maps for accurate prediction.

5.3 Back-end Network

The backend network consists of 3 convolution layers, 1 depthwise convolution layer, 5 depthwise dense block (DDB) layers with ICDB and 6 max pooling layers for downsampling the feature maps. The DDB consists of dense blocks using depthwise convolutions instead of the regular convolutions. In upcoming sections, the structure and working of the dense block, depthwise convolutions and ICDB block has been discussed.

5.3.1 Dense Blocks

Dense blocks were first introduced in DenseNets [17]. DSOD [35] used it for build a detector that can be trained from scratch with state-of-the-art accuracy and better performance. Later, Tiny-DSOD replaced the convolution layers in these dense block

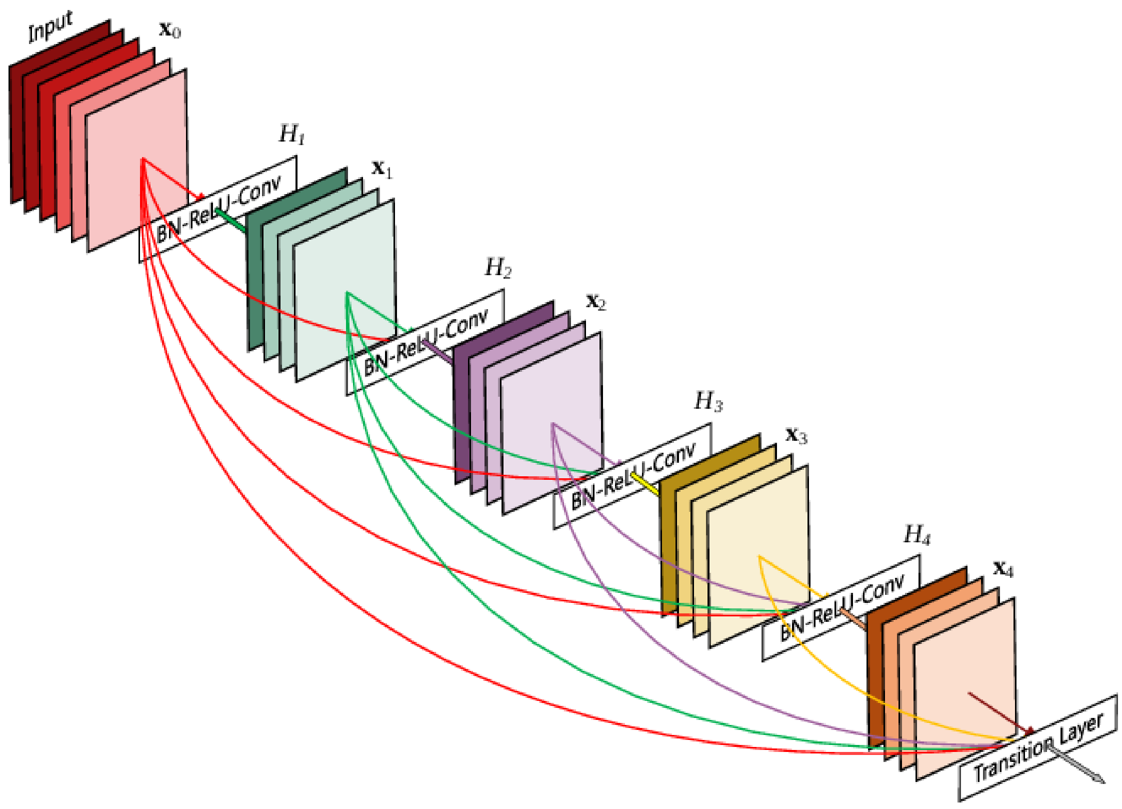


FIGURE 5.1: The contents of a dense block with growth rate of $k = 4$.[17]

with depthwise convolution to get the depthwise dense blocks, dense blocks with skip-connections.

Skip-connection were first introduced by ResNets. It was used to train very deep CNNs (> 30 hidden layers) as they suffered from the vanishing gradient problem. The skip-connection improve the gradient flow between the layers and also improve information flow. Further, DenseNet introduced dense connections between convolutional layers. In denseNets, every convolution layer is connected to all the previous layers in one dense block.

Therefore, inside a dense block, the n^{th}, L_n convolution layer is connected to all the previous layers, $L_0, L_1..., L_{n-1}$. The N^{th} layer gets the feature maps from all the $N-1$ layers. In the implementation, all the feature maps are combined in one 3-D numpy array.

To facilitate the concatenation of the feature maps, downsampling or upsampling is done inside a dense block. The downsampling of the feature maps are done using

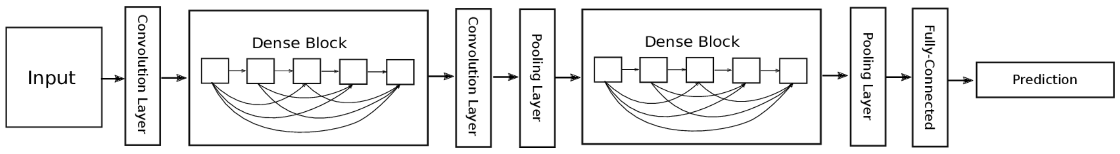


FIGURE 5.2: Dense Blocks with the transition layers between them.

transition layers between the dense blocks. These transition layers are made of a batch normalization (BN) layer, pointwise convolution layer and a max pooling layer.

Every convolution layer in a dense block has a fixed of channels give by, growth-rate (g). The value is typically, $g = 12$. This is much lower compared to traditional convolution layer, which can have over 500 channels. Experiments show dense blocks based networks perform better than ResNets in both accuracy and speed, while containing $1/50^{th}$ parameters. This is one of the major reasons for using dense blocks in our detector.

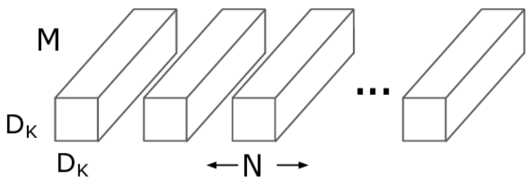
5.3.2 Depthwise Separable Convolutions

In this section, the arithmetic behind depthwise separable (DS) convolutions [16] has been presented. DS convolutions divide standard convolutions into pointwise and depthwise convolutions. This process reduces the number of operations in the convolution operation by a factor of over $\frac{1}{N}$, where N is the number of filters in the convolution layer.

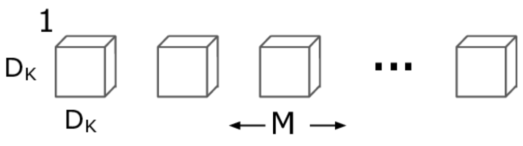
Consider N convolutional filters of dimensions, $D_L \times D_L$, with M channels each. Such a filter can be represented by a 4-D array as, $D_L \times D_L \times M \times N$. This filter is broken into a depthwise filter of dimensions, $D_L \times D_L \times 1 \times M$, and pointwise filter with dimensions, $1 \times 1 \times M \times N$. Figure 5.3 shows this factorization. Considering an input image with with dimensions $D_I \times D_I \times M$. The cost of the convolution operation will be:

$$D_I \cdot D_I \cdot D_L \cdot D_L \cdot M \cdot N \tag{5.1}$$

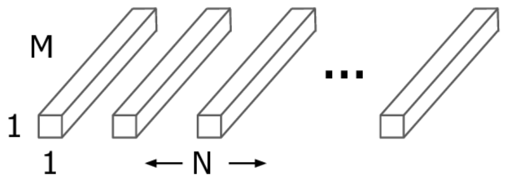
Now, considering DS convolutions, the first step is depthwise and then pointwise convolutions. Depthwise convolutions will have one filter / channel of the input image, the computation cost of this step would be:



(a)



(b)



(c)

FIGURE 5.3: Regular filters vs depthwise separable convolutional filters. [16]

$$D_I \cdot D_I \cdot D_L \cdot D_L \cdot M \tag{5.2}$$

The result of this operation is filtering the features from the input feature map. Then, pointwise convolutions are applied to create new features. Filters of size $1 \times 1 \times N$ are used for pointwise convolutions. The cost of this step is:

$$D_I \cdot D_I \cdot N \cdot M \tag{5.3}$$

Finally, the total cost of DS convolutions is:

$$D_I \cdot D_I \cdot D_L \cdot D_L \cdot M + D_I \cdot D_I \cdot N \cdot M \tag{5.4}$$

Comparing the cost of regular convolutions and DS convolutions, we get:

$$\frac{D_I \cdot D_I \cdot D_L \cdot D_L \cdot M + D_I \cdot D_I \cdot N \cdot M}{D_I \cdot D_I \cdot D_K \cdot D_L \cdot M \cdot N} = \frac{1}{N} + \frac{1}{D_L^2} \tag{5.5}$$

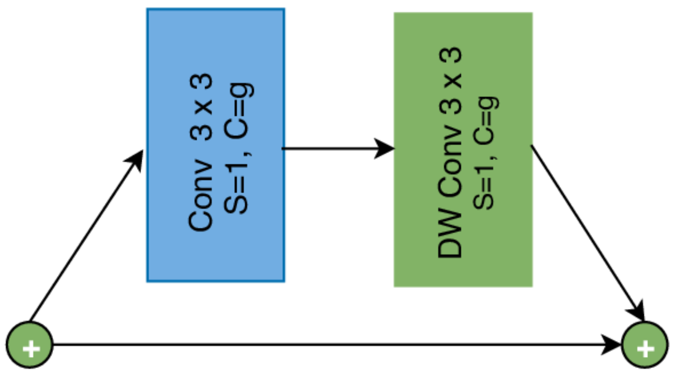


FIGURE 5.4: The new DDB block in the proposed model.

It can be seen that DS convolutions reduce the computation cost by a ratio of $\frac{1}{N} + \frac{1}{D_L^2}$. Making them much more efficient that regular convolution operations.

5.3.3 DDB

Using the DS convolutions inside dense blocks, Tiny-DSOD created the depthwise dense blocks (DDB). Combining these blocks gives the advantage of lower number of channels and lower computational cost. Our proposed model consists of DDB blocks, each made of a 1×1 conv layer and channels $= g$, followed by a depthwise conv layer with same number of channels. Figure 5.4 shows the variation of DDB used in the proposed model.

5.3.4 ICDB

Feature maps produced by the convolution operation is a good representation of the features in the input image. Typically, these feature maps contain hundreds of channels, and more channels means more parameters but it also makes the feature map semantically rich. Convolutional filter are effective in processing images because they can model the spatial relations of the objects in an image. Unfortunately, there is no information on the relationship between all the channels in a feature map. Using simple two layer neural network, we add the ICDB block in the back-end network, so that the model can learn the relationships between the different channels in a feature map. This allows the model to learn which channels contain more relevant information for the current task (object detection).

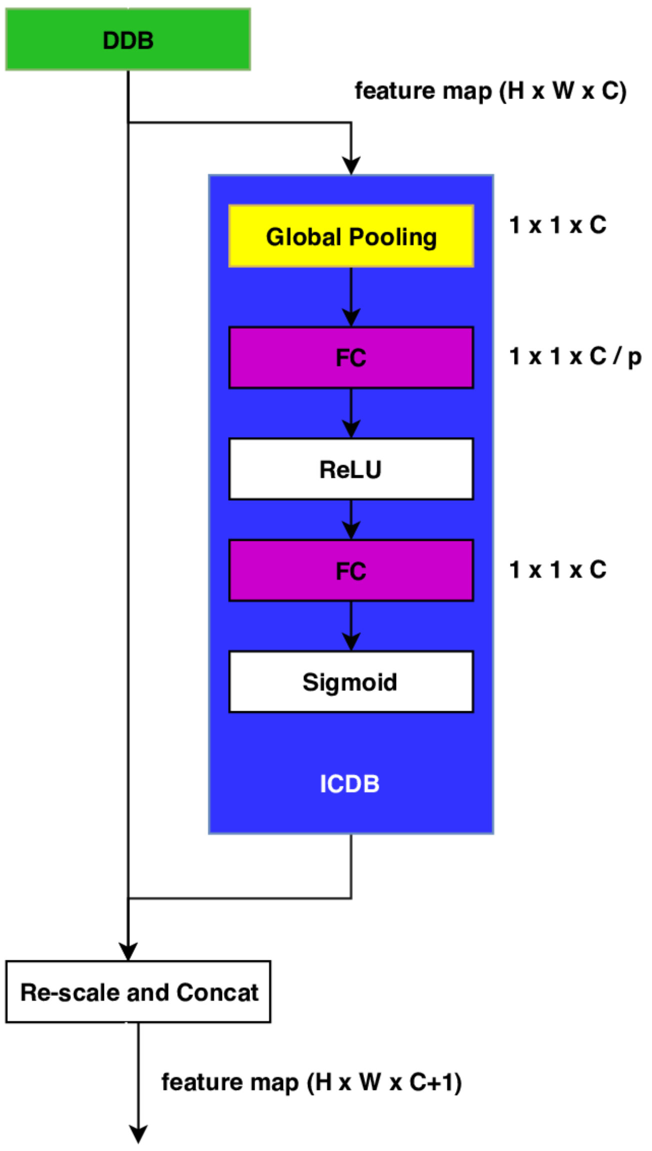


FIGURE 5.5: Inter-channel dependency block (ICDB) as used in the proposed method.

The proposed ICDB block is shown in the figure 6.4. The feature map generated by the DDB blocks becomes the input of ICDB. The information each of the channel is aggregated in a feature of vector of size C using global average pooling. This feature vector is then subjected to ReLU to add non-linearity and passed into a FC layer which reduced its length by a factor of p to reduce parameter count, as a feature map can have hundreds of channels. A small value of p means more parameters in ICDB. Experiments have shown that the value of $p = 8$ gives a good trade-off between parameter count and accuracy.

Finally, the output of the second FC layer is passed through the sigmoid gating function and re scaled into a 2D feature map with one channel and stacked on top of the feature maps generated by the DDB.

5.4 Front-end Network

The front-end network is made of L-FPN. A variation of FPN that uses DS convolutions, similar to the one used in Tiny-DSOD. The structure of the network is shown in figure 5.8. It consists of a downsampling followed by an upsampling pathways with lateral connection to share feature maps.

First the feature maps produced by the backend network fed into the downsampling pathway. This feature maps goes through five downsampling blocks. All the down sampled feature maps are then subjected to the upsampling block. The output of each of the upsampled blocks and the smallest downsampling block is stacked together to form the prediction layer. The prediction layer is fed into two parallel softmax layers. One outputs the bounding box offsets and other gives the class scores.

5.4.1 Downsampling Block

Figure 5.6 shows the downsampling block used in our model. It uses max pooling to reduce the dimensions of the input feature maps.

5.4.2 Upsampling Block

Figure 5.7 show the upsampling block. Using bi-linear interpolation the feature map size is doubled. This map is then fed into a depthwise convolution block.

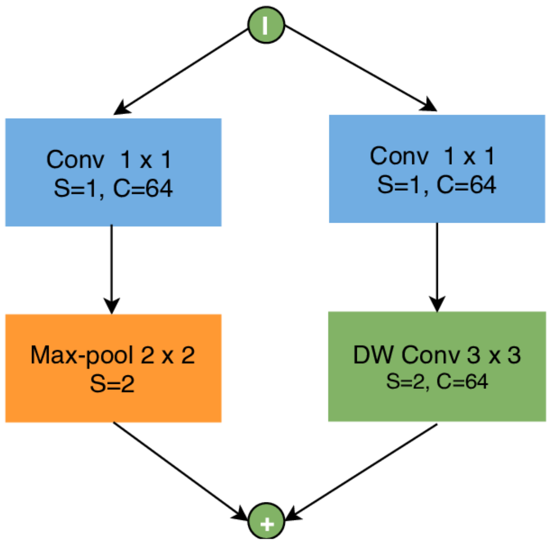


FIGURE 5.6: Downsampling block in the font-end network.

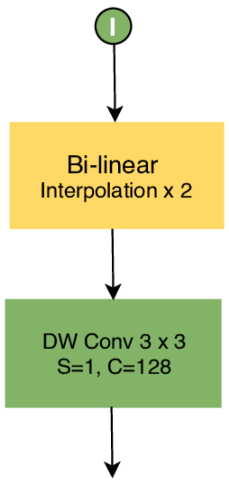


FIGURE 5.7: Upsampling block in the font-end network.

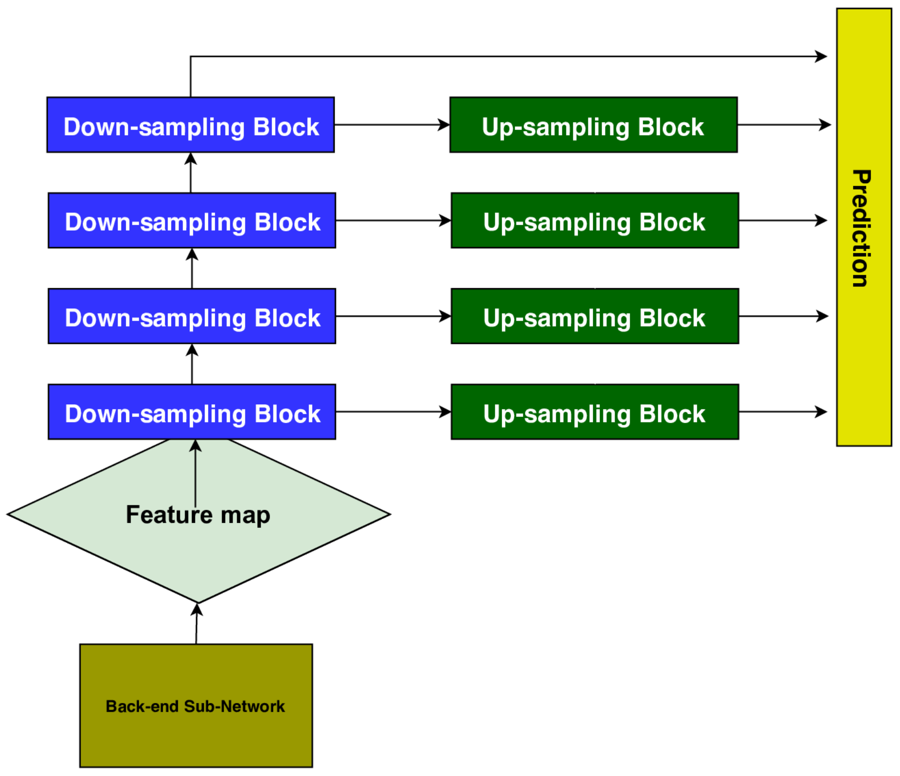


FIGURE 5.8: The front-end network.

Chapter 6

Experiments and Results

State-of-the-art detectors like Faster R-CNN, SSD, YOLO, DSOD, Tiny-YOLO, MobileNet-SSD and the proposed method have been trained and tested on three benchmark datasets, PASCAL VOC 07 and 12 [8], KITTI [10] and COCO [24]. In the subsequent the experiments of each of the dataset is presented along the training parameters. For each model and dataset we have computed the number of parameters, computations (FLOPs) and overall accuracy (mAP) has been recorded and compared.

TABLE 6.1: System Specifications

Operating System	64-bit Ubuntu 18.04 LTS
CPU	Core i9
Memory	32GB
GPU	Nvidia RTX 2080 Ti
CUDA	Ver. 10.1

6.1 Datasets

The ImageNet [4] dataset has been used for pre-training Faster R-CNN, SSD and YOLO. PASCAL VOC [7], KITTI [10] and COCO [24] have been used to fine tune and test the detectors. The following table outline these datasets.

Table 6.2: Datasets Used for Experiments.

Dataset	Classes	Images	Size
ImageNet	200	4,56,567	55G
PASCAL VOC 07	20	9,963	2.7G
KITTI	80	7,481	6.2G
COCO	80	123,287	24G

6.1.1 ImageNet

One of the biggest dataset containing images from 200 classes. It was first introduced in 2010 as a part of the Large Scale Visual Recognition Challenge (ILSCRC). The challenge consists of classification, object localization and detection task. The ImageNet dataset is divided into subsets for each of these challenges. Along with the manually labelled and quality controlled ground truths.

The images in ImageNet are ordered based on the WordNet hierarchy[1]. WordNet consists of set of words and phrases that carry the same meaning. Hence, they are also called ”synomym sets” or ”sysnets”.

ImageNet has been used for pre-training Faster R-CNN, YOLO, SSD and MobileNet-SSD. This is done by training the feature extractor CNN in each of the detectors as a classifier, whose weights are then saved in a file. These weights are then used to initialize the new detector.

6.1.2 PASCAL VOC

This dataset was created for the PASCAL Visual Object Challenge[2]. For the experiments, we have considered the Pascal VOC 2007 and 2012 datasets as they have been extensively used for benchmarking detectors by researchers. Unlike ImageNet, this dataset consists of higher resolution images containing multiple objects, photographed under natural conditions. Both VOC 07 + 12 consist of 20 classes and a total of 20K images.

[1]https://wordnet.princeton.edu/

[2]http://host.robots.ox.ac.uk/pascal/VOC/voc2007/

TABLE 6.3: PASCAL VOC 2007 Detection Results.

Method	**Param. Count**	**Operations (FLOPs)**	**Accuracy (mAP)**
Faster-RCNN	134.80M	181.12B	73.2
SSD	26.20M	31.75B	77.2
Tiny-YOLO	15.22M	6.97B	57.1
MobileNet-SSD	5.60M	1.14B	68.0
DSOD-smallest	5.86M	5.29B	73.6
Tiny-DSOD	0.94M	1.06B	72.1
Proposed Method	1.31M	1.2B	73.4

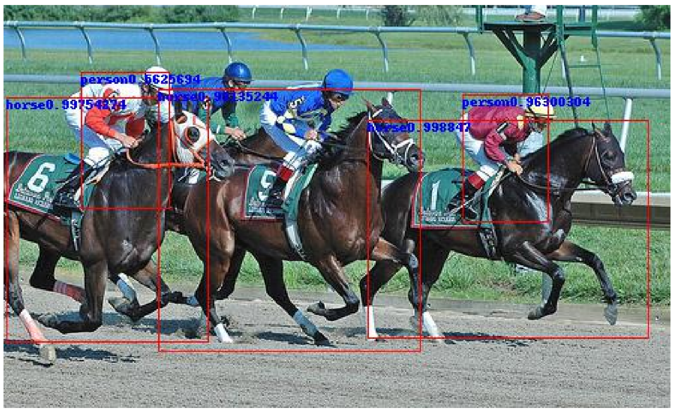
6.1.2.1 Results on PASCAL VOC

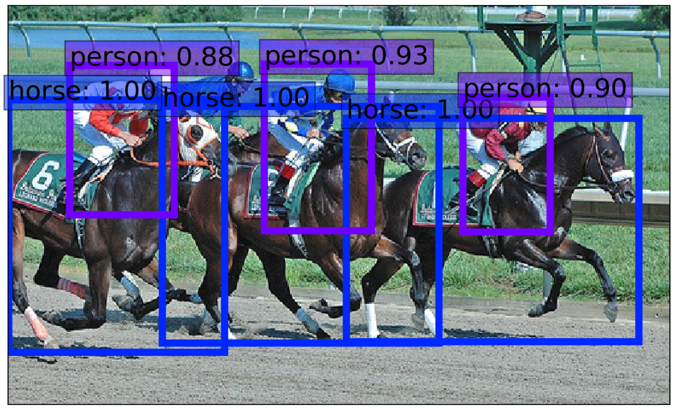
All the model compared have been trained on combined VOC 07 and 12 train set. For the proposed model, the batch size is 128. The initial learning rate is 0.1 which is reduced by a factor of 10 every 20K iterations. Total number of iterations is 160K. SGD has been used for gradient calculations, with momentum = 0.1. Weight decay of 0.0005 has been considered in the SGD to reduce the effect of large gradients and avoid overfitting. The weights are randomly initialized, with values $\in (0, 1)$.

Table 6.3 shows the results on VOC dataset. Faster R-CNN and SSD are the full-size detectors and Tiny-YOLO, MobileNet-SSD, DSOD-smallest, Tiny-DSOD and proposed method are the lightweight detectors. These detectors have been considered for comparison as they were the state-of-the-art at time of their introduction.

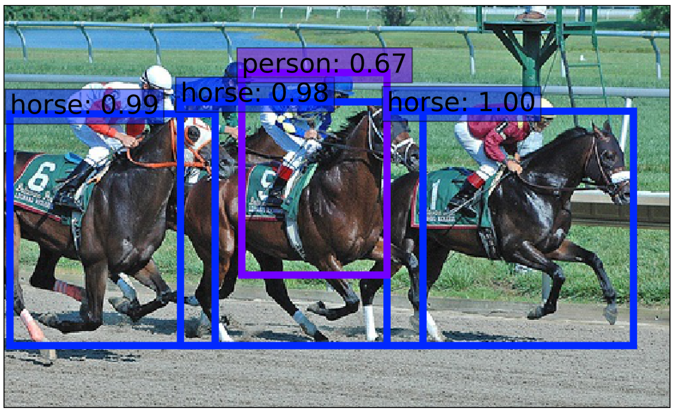
As seen in the table, the proposed model obtains an overall accuracy of 73.4 mAP, better than all the lightweight detectors except DSOD-smallest. Although DSOD-smallest has has $4\times$ the parameters. The computational cost of the proposed method is much lower.

Among all, the highest accuracy attained by SSD, 77.2 mAP, 4.1% higher that the proposed method. Unfortunately, the SSD detector is over $20\times$ bigger than the proposed method. Therefore, it can be said that the proposed model achieves a better trade-off between accuracy and complexity. The detection results on a sample image from the VOC 07 test set has been shown in the figure 6.1. Note that the bounding boxes show the location of the object, labelled with the class and the confidence score.

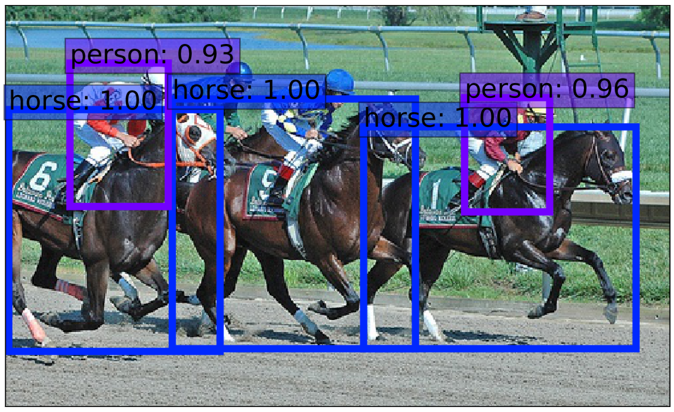

(a) SSD on PASCAL VOC 2007



(b) DSOD on PASCAL VOC 2007



(c) Tiny-DSOD on PASCAL VOC 2007



(d) Our method on PASCAL VOC 2007

FIGURE 6.1: Detection results of some of the methods on sample image taken from PASCAL VOC 2007 dataset.

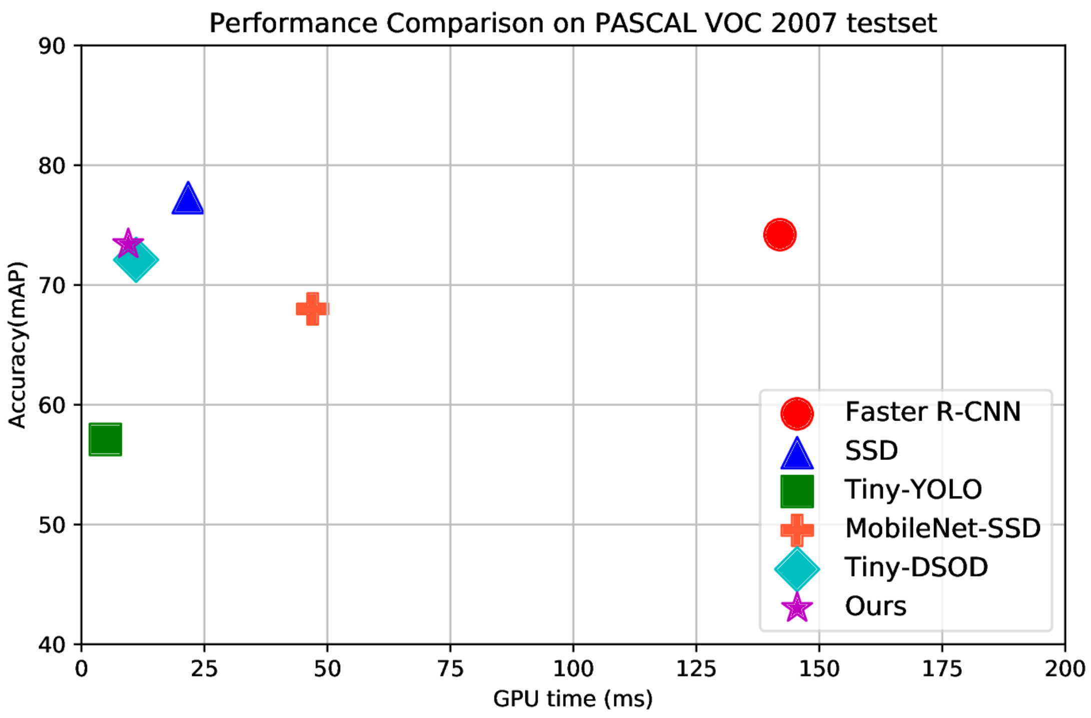


FIGURE 6.2: PASCAL VOC 07 and 12, detection speed, running on the our system.

6.1.3 KITTI

KITTI is a specialized dataset aimed at creating models for autonomous driving. The images have been taken by human driven cars equipped with multiple image sensors. The object detection benchmark in KITTI has been used for this project.

This dataset contains 7000 images with 80 classes. Only the relevant classes such as cars, pedestrians, traffic-lights etc, have been labelled in the ground-truth images. The dataset has been divided into a train and test set containing 70% and 30% of the images.

6.1.3.1 Results on KITTI

The KITTI images are much larger compared to the VOC dataset. Therefore, the resolution of all the layers had to be increased for training. The higher resolution improved accuracy but also increased the parameter count. All the models were trained on 5k images and evaluated on 3k images. Considering the larger images and lesser number of training examples, the batch size was reduced to 64. The total number of iterations is 60k. Initial learning rate is 0.01 which is divided by a factor of 2 every 10k iterations.

Table 6.4: KITTI Detection Results.

Method	**Params**	**FLOPs**	**Accuracy(mAP)**	**Time (ms)**
Tiny-YOLO	22.55M	35.6B	69.8	9.1
MobileNet-SSD	1.98M	9.7B	76.7	28.09
Tiny-DSOD	0.85M	4.1B	77.0	15.15
Proposed Method	1.23M	6.3B	**77.6**	15.9

SGD with momentum = 0.1 has been used to adjust the model weights. Weigh-decay = 0.0005, to reduce over-fitting. All the layers are initialized with random weights. Table 6.4 shows that out method achieves the higher accuracy among all its lightweight counterparts. This increase is accompanied with a marginal increase in the model parameters. The detection result on a sample image from the KITTI dataset is shown in the figure 6.3.

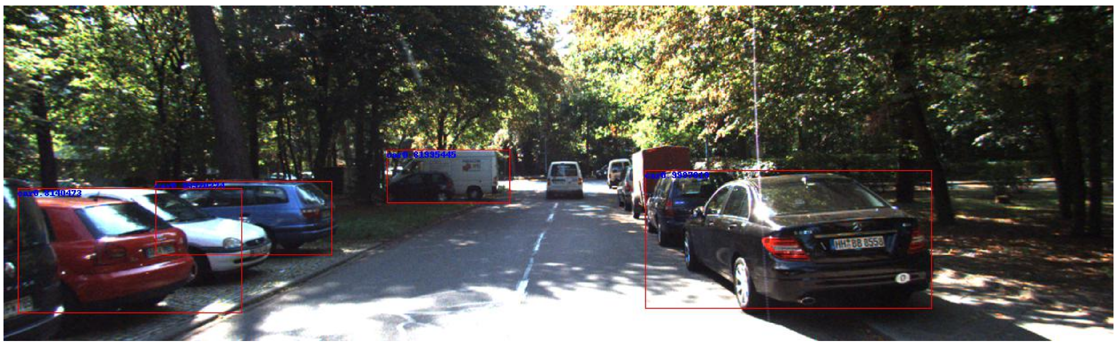
6.1.4 COCO

One of most challenging object detection dataset created using the images of common objects. It consists of 200,000 images with object from 80 classes. All the objects in the ground-truths are annotated with bounding boxes and class labels.

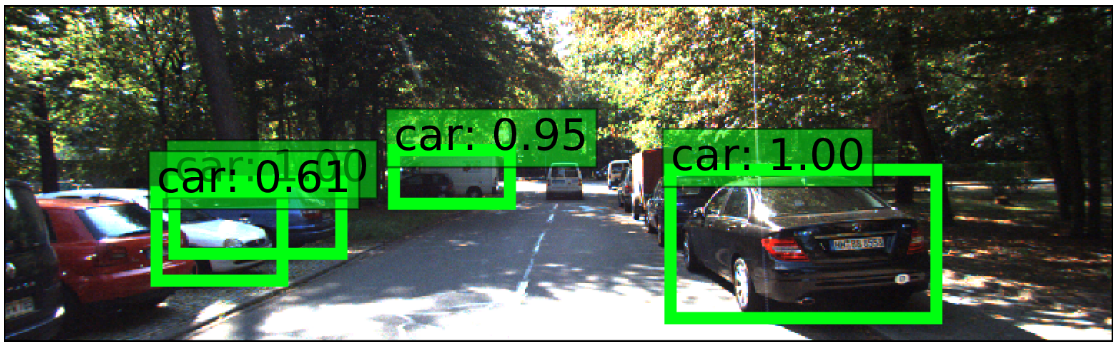
6.1.4.1 Results on COCO

All the models have been trained on the COCO trainval containing 35k images. The results have been evaluated on the 2015 test-dev dataset. The input size is fixed at 300×300 for all models. For the proposed model, the batch size = 128. Learning rate = 0.1 for 80k iterations, then divided by 10 every 60k iterations. Total training iterations is 320K as 80 classes require larger output layers and more training. The results are shown in table 6.5.

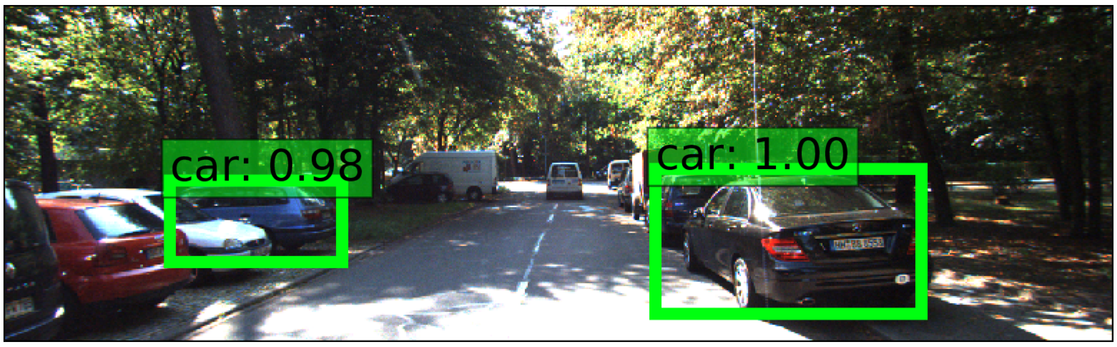
The proposed model attained better accuracy compared to other lightweight detectors. The model size is also much smaller compared to others, except Tiny-DSOD. These results show that the proposed model is accurate and efficient for different applications, hence can be called a ”robust lightweight detector”. The result on a sample COCO image is shown in figure 6.5.



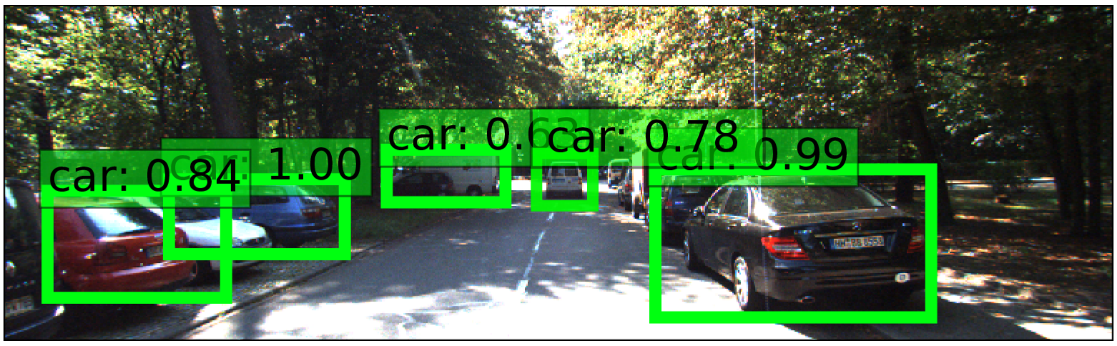
(a) SSD on KITTI



(b) DSOD on KITTI



(c) Tiny-DSOD on KITTI



(d) Our method on KITTI

FIGURE 6.3: KITTI Results.

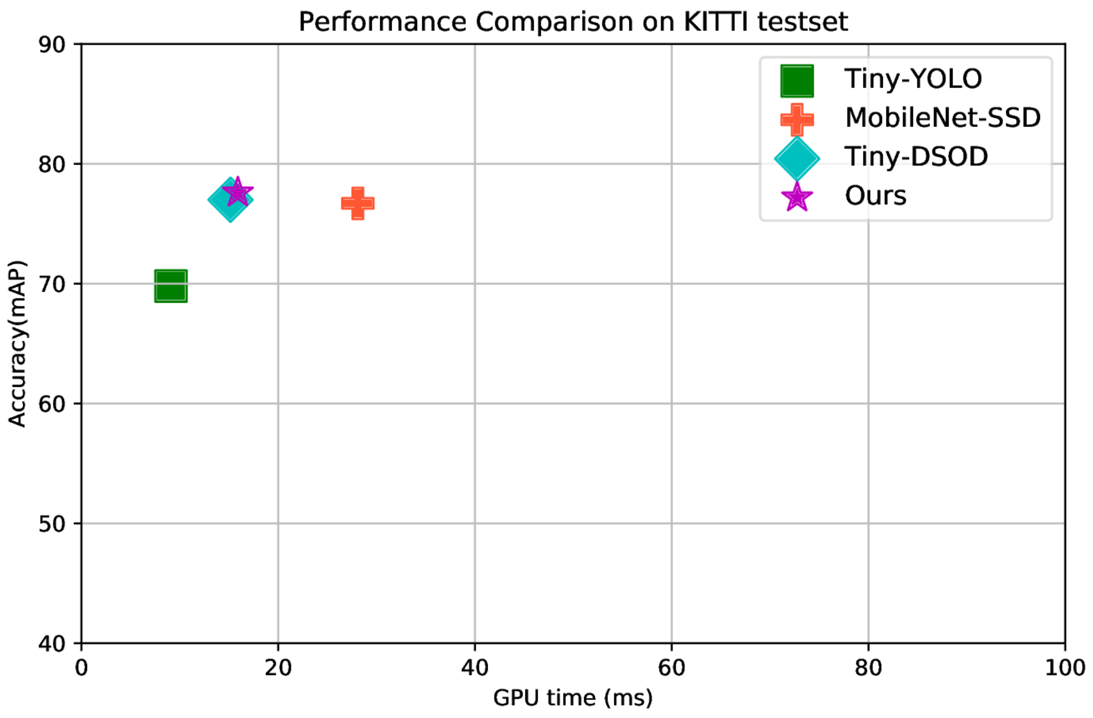
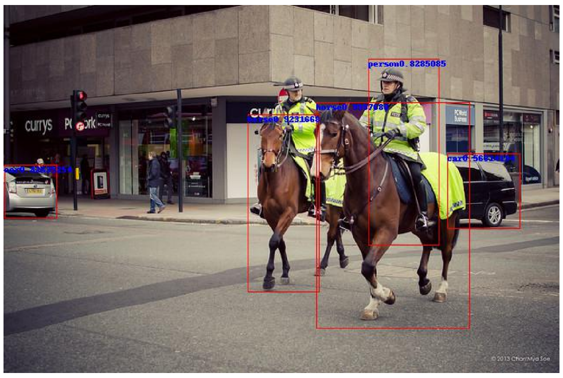


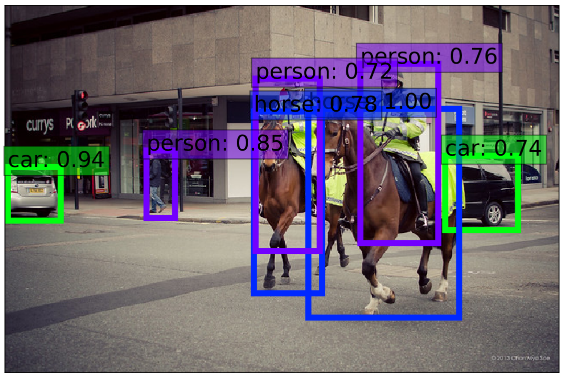
FIGURE 6.4: Time taken per image for KITTI. Tested on system configuration given in table 6.1

TABLE 6.5: Detection Results on COCO test-dev dataset.

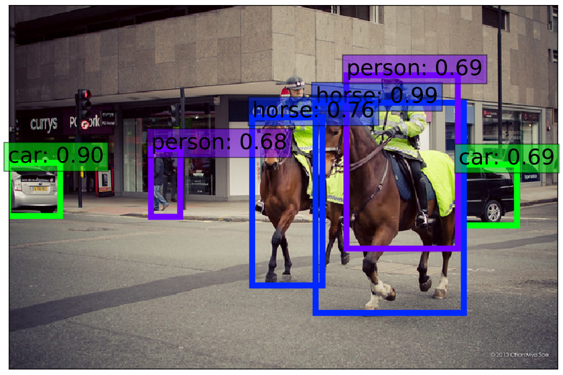
Method	**Param. Count**	**Ops. (FLOPs)**	**mAP**	**Time/Image(ms)**
MobileNet-SSD	4.30M	0.80B	22.1	46
Tiny-Yolo	18.43M	34.36B	21.6	5.1
Tiny-DSOD	1.15M	1.12B	23.2	18.14
Proposed Method	1.32M	1.55B	24.6	26.3
SSD	34.30M	34.34B	25.4	145



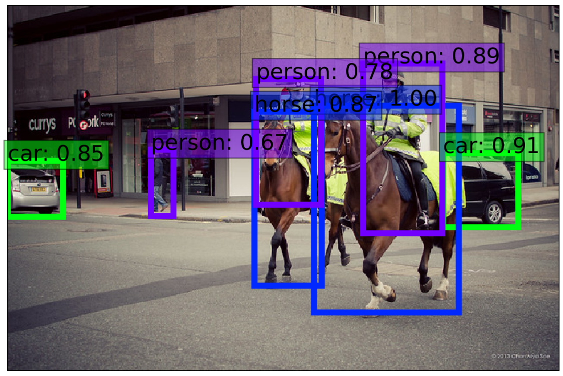
(a) SSD on COCO



(b) DSOD on COCO



(c) Tiny-DSOD on COCO



(d) Our method on COCO

FIGURE 6.5: Results on a sample from COCO.

Chapter 7

Conclusion and Future Work

CNNs have shown to have high capacity to learn which is very beneficial when dealing with high-resolution images containing objects from multiple classes. The major challenges to object detection such as occlusion, viewpoint changes, scale variations, lighting problems has been addressed by CNNs much better that older methods. Therefore, CNN based object detector have been extensively development in the recent year. Unfortunately, most of these detector have large number of parameters and a very high computational cost. Full-size detectors such as Faster R-CNN achieves state-of-the-art accuracy on the VOC dataset, but take 181 billion FLOPs to train. Such detectors require high-end system to make training and testing feasible.

In an effort to reduce the model complexity of the modern object detectors, a new category of detectors recently evolved, called lightweight object detectors. These detectors are designed to work in resource-restricted scenarios, such as mobile devices and laptops. Example of such detectors are Tiny-YOLO, MobileNet-SSD and Tiny-DSOD. Most of these method borrow the basic design principles from their full-size counterparts and at the same time use some novel techniques to reduce the model size and computation cost.

After exploring the contributions of the state-of-the-art detectors, we proposed a new lightweight detector with modified dense block and a new inter-channel dependency block (ICDB). The proposed model attains the accuracy of 73.4 mAP on the PASCAL VOC 07 testset. 77.6 mAP on KITTI and 24.6 mAP on the COCO dataset. Comparison shows that the proposed model achieves superior accuracy with a marginal increase in the model size.

In the future, the accuracy of the proposed model may be improved by using dense blocks in the backbone network and by setting a higher value for the growth rate of the dense blocks. Instead of one, multiple ICDB blocks can be added in parallel to improve the model's accuracy. Moreover, the proposed model can also be trained for 3D object detection, by small modifications to the input and output. It can also be used for real-time object tracking by training on appropriate datasets.

Bibliography

[1] Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828, 2014.

[2] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc., 2016.

[3] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE Computer Society, 2005.

[4] 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.

[5] Li Deng, Michael L. Seltzer, Dong Yu, Alex Acero, Abdel rahman Mohamed, and Geoffrey E. Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In Takao Kobayashi, Keikichi Hirose, and Satoshi Nakamura, editors, *INTERSPEECH*, pages 1692–1695. ISCA, 2010.

[6] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(4):743–761, April 2012.

[7] 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.

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010.

[9] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, September 2010.

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[11] Ross Girshick. Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 1440–1448, Washington, DC, USA, 2015. IEEE Computer Society.

[12] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, Nov 2012.

[16] 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. *CoRR*, abs/1704.04861, 2017.

[17] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[18] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. pages 448–456, 2015.

[19] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[21] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-dsod: Lightweight object detection for resource-restricted usages. *CoRR*, abs/1807.11013, 2018.

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

[23] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 936–944, 2017.

[24] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. 2016. To appear.

[26] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[27] Eshed Ohn-Bar and Mohan M. Trivedi. To boost or not to boost? on the limits of boosted trees for object detection. *CoRR*, abs/1701.01692, 2017.

[28] Walter Pitts and Warren S McCulloch. How we know universals the perception of auditory and visual forms. *The Bulletin of mathematical biophysics*, 9(3):127–147, 1947.

[29] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[30] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 779–788, 2016.

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[32] Xiaofeng Ren and Deva Ramanan. Histograms of sparse codes for object detection. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 3246–3253, Washington, DC, USA, 2013. IEEE Computer Society.

[33] Sebastian Ruder. An overview of gradient descent optimization algorithms., 2016. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[34] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, April 2017.

[35] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. DSOD: learning deeply supervised object detectors from scratch. *CoRR*, abs/1708.01241, 2017.

[36] K. . Sung and T. Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):39–51, Jan 1998.

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

[38] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[39] Paul Viola and Michael J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, May 2004.