SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving

Bichen Wu[1], Forrest Iandola[1,2], Peter H. Jin[1], Kurt Keutzer[1,2]
[1]UC Berkeley, [2]DeepScale
bichen@berkeley.edu, forrest@deepscale.ai, phj@berkeley.edu, keutzer@berkeley.edu

Abstract

Object detection is a crucial task for autonomous driving. In addition to requiring high accuracy to ensure safety, object detection for autonomous driving also requires real-time inference speed to guarantee prompt vehicle control, as well as small model size and energy efficiency to enable embedded system deployment.

In this work, we propose SqueezeDet, a fully convolutional neural network for object detection that aims to simultaneously satisfy all of the above constraints. In our network we use convolutional layers not only to extract feature maps, but also as the output layer to compute bounding boxes and class probabilities. The detection pipeline of our model only contains a single forward pass of a neural network, thus it is extremely fast. Our model is fully-convolutional, which leads to small model size and better energy efficiency. Finally, our experiments show that our model is very accurate, achieving state-of-the-art accuracy on the KITTI [10] benchmark. The source code of SqueezeDet is open-source released. [1]

1. Introduction

A safe and robust autonomous driving system relies on accurate perception of the environment. To be more specific, an autonomous vehicle needs to accurately detect cars, pedestrians, cyclists, road signs, and other objects in real-time in order to make the right control decisions that ensure safety. Moreover, to be economical and widely deployable, this object detector must operate on embedded processors that dissipate far less power than powerful GPUs (Graphics Processing Unit) used for benchmarking in typical computer vision experiments.

Object detection is a crucial task for autonomous driving. Different autonomous vehicle solutions may have different combinations of perception sensors, but image based object detection is almost irreplaceable. Image sensors are inexpensive compared with others such as LIDAR. Image data (including video) are much more abundant than, for example, LIDAR cloud points, and are much easier to collect and annotate. Recent progress in deep learning shows a promising trend that with more and more data that cover all kinds of long-tail scenarios, we can always design more powerful neural networks with more parameters to digest the data and become more accurate and robust.

While recent research has been primarily focused on improving accuracy, for actual deployment in an autonomous vehicle, there are other issues of image object detection that are equally critical. For autonomous driving some basic requirements for image object detectors include the following: a) Accuracy. More specifically, the detector ideally should achieve 100% recall with high precision on objects of interest. b) Speed. The detector should have real-time or faster inference speed to reduce the latency of the vehicle control loop. c) Small model size. As discussed in [18], smaller model size brings benefits of more efficient distributed training, less communication overhead to export new models to clients through wireless update, less energy consumption and more feasible embedded system deployment. d) Energy efficiency. Desktop and rack systems may have the luxury of burning 250W of power for neural network computation, but embedded processors targeting automotive market must fit within a much smaller power and energy envelope. While precise figures vary, the new Xavier[2] processor from Nvidia, for example, is targeting a 20W thermal design point. Processors targeting mobile applications have an even smaller power budget and must fit in the 3W–10W range. Without addressing the problems of a) accuracy, b) speed, c) small model size, and d) energy efficiency, we won't be able to truly leverage the power of deep neural networks for autonomous driving.

In this paper, we address the above issues by presenting *SqueezeDet*, a fully convolutional neural network for object detection. The detection pipeline of SqueezeDet is inspired by [24]: first, we use stacked convolution filters to extract a high dimensional, low resolution feature map for the input image. Then, we use *ConvDet*, a convolutional layer to take the feature map as input and compute a large amount of object bounding boxes and predict their categories. Finally, we filter these bounding boxes to ob-

[1] https://github.com/BichenWuUCB/squeezeDet

[2] https://blogs.nvidia.com/blog/2016/09/28/xavier/

tain final detections. The "backbone" convolutional neural net (CNN) architecture of our network is SqueezeNet [18], which achieves AlexNet level imageNet accuracy with a model size of less than 5MB that can be further compressed to 0.5MB. After strengthening the SqueezeNet model with additional layers followed by *ConvDet*, the total model size is still less than 8MB. The inference speed of our model can reach 57.2 FPS[3] (frames per second) with input image resolution of 1242x375. Benefiting from the small model size and activation size, SqueezeDet has a much smaller memory footprint and requires fewer DRAM accesses, thus it consumes only 1.4J of energy per image on a TITAN X GPU, which is about 84X less than a Faster R-CNN model described in [2]. SqueezeDet is also very accurate. One of our trained SqueezeDet models achieved the best average precision in all three difficulty levels of cyclist detection in the KITTI object detection challenge [10].

The rest of the paper is organized as follows. We first review related work in section 2. Then, we introduce our detection pipeline, the *ConvDet* layer, the training protocol and network design of SqueezeDet in section 3. In section 4, we report our experiments on the KITTI dataset, and discuss accuracy, speed, parameter size of our model. Due to limited page length, we put energy efficiency discussion in the supplementary material to this paper. We conclude the paper in section 5.

2. Related Work

2.1. CNNs for object detection

From 2005 to 2013, various techniques were applied to advance the accuracy of object detection on datasets such as PASCAL [8]. In most of these years, variants of HOG (Histogram of Oriented Gradients) + SVM (Support Vector Machine) [6] or DPM (Deformable Part Models) [9] were used to define the state-of-art accuracy on these datasets. However, in 2013, Girshick *et al*. proposed Region-based Convolutional Neural Networks (R-CNN) [12], which led to substantial gains in object detection accuracy. The R-CNN approach begins by identifying region proposals (i.e. regions of interest that are likely to contain objects) and then classifying these regions using a CNN. One disadvantage of R-CNN is that it computes the CNN independently on each region proposal, leading to time-consuming ($\leq$ 1 fps) and energy-inefficient ($\geq$ 200 J/frame) computation. To remedy this, Girshick *et al*. experimented with a number of strategies to amortize computation across the region proposals [13, 19, 11], culminating in *Faster R-CNN* [25].Another model, R-FCN (Region based Fully Convolutional Network), is fully-convolutional and delivers accuracy that is competitive with R-CNN. Its fully-convolutional architecture allows it to amortize more computation across the region proposals.

There have been a number of works that have adapted the R-CNN approach to address object detection for autonomous driving. Almost all the top-ranked published methods on the KITTI leader board are based on Faster R-CNN. [2] modified the CNN architecture to use shallower networks to improve accuracy. [4, 28] on the other hand focused on generating better region proposals. Most of these methods focused on better accuracy, but to our knowledge, no previous methods have reported real-time inference speeds on KITTI dataset.

Region proposals are a cornerstone in all of the object detection methods that we have discussed so far. However, in YOLO (You Only Look Once) [24], region proposition and classification are integrated into one single stage. Compared with R-CNN and Faster R-CNN based methods, YOLO's single stage detection pipeline is extremely fast, making YOLO the first CNN based, general-purpose object detection model that achieved real-time speed.

2.2. Small CNN models

Given a particular accuracy level on a computer vision benchmark, it is natural to investigate the question: what is the smallest model size that can achieve that level of accuracy? SqueezeNet [18] was the result of one such investigation. It achieved the same level of accuracy as AlexNet [20] on ImageNet [7] image classification with less than 5MB of parameters: a reduction of 50x relative to AlexNet. After SqueezeNet, several works continued to search for more compact network structures. ENet [23] explored spatial decomposition of convolutional kernels. Together with other techniques, ENet achieved SegNet [3] level accuracy for semantic segmentation with $79X$ less parameters. Recently MobileNet [17] explored channel-wise decomposition of convolutional kernels, and was applied to several mobile vision tasks including object detection, fine-grain classification, face attributes and landmark recognition.

2.3. Fully convolutional networks

Fully-convolutional networks (FCN) were popularized by Long *et al*., who applied them to the semantic segmentation domain [22]. FCN defines a broad class of CNNs, where the output of the final parameterized layer is a grid rather than a vector. This is useful in semantic segmentation, where each location in the grid corresponds to the predicted class of a pixel.

FCN models have been applied in other areas as well. To address the image classification problem, a CNN needs to output a 1-dimensional vector of class probabilities. One common approach is to have one or more *fully-connected layers*, which by definition output a 1D vector – $1\times1\times$Channels (e.g. [20, 26]). However, one alternative approach used in FCN models (e.g. [18, 21]) is to have

[3]Standard camera frame rate is 30 FPS, which is regarded as the benchmark of the real-time speed.

the final parameterized layer be a convolutional layer that outputs a grid (H×W×Channels), and to then use average-pooling to downsample the grid to a vector of class probabilities with the shape of 1×1×Channels. Finally, the R-FCN method that we mentioned earlier in this section is also a fully-convolutional network.

3. Method Description

3.1. Detection Pipeline

Inspired by YOLO [24], we also adopt a single-stage detection pipeline in which region proposal and classification is performed by one single network simultaneously. As shown in Fig. 1, a convolutional neural network first takes an image as input and extracts a low-resolution, high dimensional feature map from the image. Then, the feature map is fed into the *ConvDet* layer to compute bounding boxes centered around $W \times H$ uniformly distributed spatial grids. Here, W and H are number of grid centers along horizontal and vertical axes.

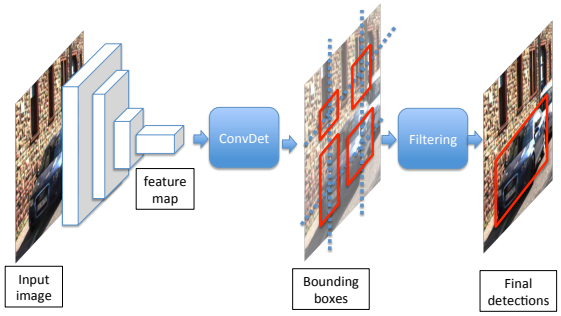


Figure 1. SqueezeDet detection pipeline. A convolutional neural network extracts a feature map from the input image and feeds it into the *ConvDet* layer. The *ConvDet* layer then computes bounding boxes centered around $W \times H$ uniformly distributed grid centers. Each bounding box is associated with 1 confidence score and C conditional class probabilities. Then, we keep the top N bounding boxes with highest confidence and use NMS (Non-Maximum Suppression) to filter them to get the final detections.

Each bounding box is associated with $C + 1$ values, where C is the number of classes to distinguish, and the extra 1 is for the confidence score. Similarly to YOLO [24], we define the confidence score as $Pr(\text{Object}) * \text{IOU}^{pred}_{truth}$. The $Pr(\text{Object})$ encodes the probability that an bounding box contains an object. $\text{IOU}^{pred}_{truth}$ is the intersection-over-union between the predicted bounding box and the ground truth bounding box. A high confidence score implies a high probability that an object of interest does exist and that the overlap between the predicted bounding box and the ground truth is high. The other C scalars represents the conditional class probability distribution given that the object exists within the bounding box. More formally, we denote the conditional probabilities as $Pr(\text{class}_c|\text{Object}), c \in [1, C]$. We assign the label associated with the highest conditional probability to this bounding box and we use

$$\max_c Pr(\text{class}_c|\text{Object}) * Pr(\text{Object}) * \text{IOU}^{pred}_{truth}$$

as the metric to estimate the confidence of the bounding box prediction.

Finally, we keep the top N bounding boxes with the highest confidence and use Non-Maximum Suppression (NMS) to filter redundant bounding boxes to obtain the final detections. During inference, the entire detection pipeline consists of only one forward pass of one neural network with minimal post-processing.

3.2. ConvDet

The *SqueezeDet* detection pipeline is inspired by YOLO [24]; however, as we will describe in this section, the design of the *ConvDet* layer enables SqueezeDet to generate tens-of-thousands of region proposals with many fewer model parameters compared to YOLO.

ConvDet is essentially a convolutional layer that is trained to output bounding box coordinates and class probabilities. It works as a sliding window that moves through each spatial position on the feature map. At each position, it computes $K \times (4 + 1 + C)$ values that encode the bounding box predictions. Here, K is the number of reference bounding boxes with pre-selected shapes. 1 is for the confidence score and 4 is the number of bounding box coordinates. Using the notation from [25], we call these reference bounding boxes as anchors. Each position on the feature map corresponds to a grid center in the original image, so each anchor can be described by 4 scalars as $(\hat{x}_i, \hat{y}_j, \hat{w}_k, \hat{h}_k), i \in [1, W], j \in [1, H], k \in [1, K]$. Here $\hat{x}_i, \hat{y}_i$ are spatial coordinates of the reference grid center (i, j). $\hat{w}_k, \hat{h}_k$ are the width and height of the k-th reference bounding box. We used the K-means based method described by [2] to select reference bounding box shapes to match the data distribution.

For each anchor (i, j, k), we compute 4 relative coordinates $(\delta x_{ijk}, \delta y_{ijk}, \delta w_{ijk}, \delta h_{ijk})$ to transform the anchor into a predicted bounding box, as shown in Fig. 2. Following [14], the transformation is described by

$$\begin{aligned} x^p_i &= \hat{x}_i + \hat{w}_k \delta x_{ijk}, \quad & y^p_j &= \hat{y}_j + \hat{h}_k \delta y_{ijk}, \\ w^p_k &= \hat{w}_k \exp(\delta w_{ijk}), \quad & h^p_k &= \hat{h}_k \exp(\delta h_{ijk}), \end{aligned} \tag{1}$$

where $x^p_i, y^p_j, w^p_k, h^p_k$ are predicted bounding box coordinates.

ConvDet is similar to the last layer of RPN in Faster R-CNN [25]. The major difference is that, RPN is regarded as a "weak" detector that is only responsible for detecting whether an object exists and generating bounding box proposals for the object. The classification is handed over to

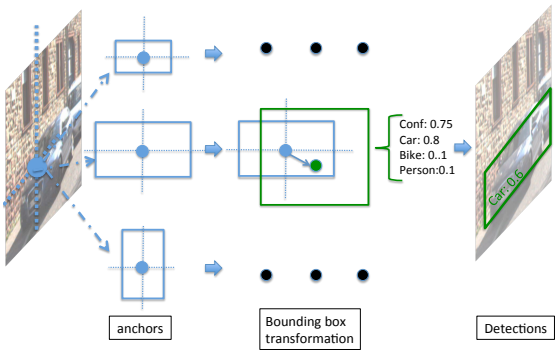


Figure 2. Bounding box transformation. Each grid center has K anchors with pre-selected shapes. Each anchor is transformed to its new position and shape using the relative coordinates computed by the *ConvDet* layer. Each anchor is associated with a confidence score and class probabilities to predict the category of the object within the bounding box.

fully connected layers, which are regarded as a "strong" classifier. However, we exploit that fact that convolutional layers are "strong" enough to detect, localize, and classify objects at the same time.

For simplicity, we denote the detection layers of YOLO [24] as *FcDet* (only counting the last two fully connected layers). Compared with *FcDet*, the *ConvDet* layer has orders of magnitude fewer parameters and is still able to generate more region proposals with higher spatial resolution. The comparison between *ConvDet* and *FcDet* is illustrated in Fig. 3.

Assume that the input feature map is of size (W_f, H_f, Ch_f), W_f is the width of the feature map, H_f is the height, and Ch_f is the number of input channels to the detection layer. Denote *ConvDet*'s filter width as F_w and height as F_h. With proper padding/striding strategy, the output of *ConvDet* keeps the same spatial dimension as the feature map. To compute $K \times (4 + 1 + C)$ outputs for each reference grid, the number of parameters required by the *ConvDet* layer is $F_w F_h \text{Ch}_f K(5 + C)$.

The *FcDet* layer described in [24] is comprised of two fully connected layers. Using the same notation for the input feature map and assuming the number of outputs of the $fc1$ layer is F_{fc1}, then the number of parameters in the $fc1$ layer is $W_f H_f \text{Ch}_f F_{fc1}$. The second fully connected layer in [24] generates C class probabilities as well as $K \times (4+1)$ bounding box coordinates and confidence scores for each of the $W_o \times H_o$ grids. Thus, the number of parameters in the $fc2$ layer is $F_{fc1} W_o H_o (5K + C)$. The total number of parameters in these two fully connected layers is $F_{fc1}(W_f H_f \text{Ch}_f + W_o H_o (5K + C))$.

In [24], the input feature map is of size 7x7x1024.

	RP	cls	#Parameter
RPN	✓	✗	$\text{Ch}_f K(4+1)$
ConvDet	✓	✓	$F_w F_h \text{Ch}_f K(5+C)$
FcDet	✓	✓	$F_{fc1}(W_f H_f \text{Ch}_f + W_o H_o (5K + C))$

Table 1. Comparison between RPN, *ConvDet* and *FcDet*. RP stands for region proposition. cls stands for classification.

$F_{fc1} = 4096$, $K = 2$, $C = 20$, $W_o = H_o = 7$, thus the total number of parameters required by the two fully connected layers is approximately 212×10^6. If we keep the feature map sizes, number of output grid centers, classes, and anchors the same, and use 3x3 *ConvDet*, it would only require $3 \times 3 \times 1024 \times 2 \times 25 \approx 0.46 \times 10^6$ parameters, which is 460X smaller than *FcDet*. The comparison of RPN, *ConvDet* and *FcDet* is illustrated in Fig. 3 and summarized in Table 1.

3.3. Training protocol

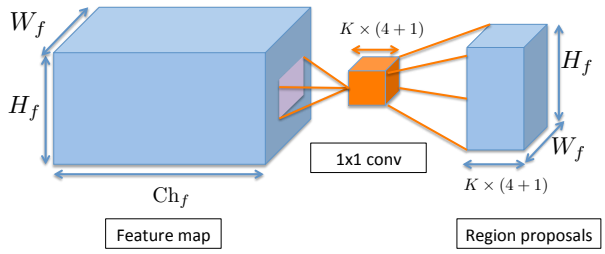
Unlike Faster R-CNN [25], which deploys a (4-step) alternating training strategy to train RPN and detector network, our SqueezeDet detection network can be trained end-to-end, similarly to YOLO [24]. To train the *ConvDet* layer to learn detection, localization, and classification simultaneously, we define a multi-task loss function:

$$\begin{aligned} &\frac{\lambda_{bbox}}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} [(\delta x_{ijk} - \delta x_{ijk}^G)^2 + (\delta y_{ijk} - \delta y_{ijk}^G)^2 \\ &\qquad + (\delta w_{ijk} - \delta w_{ijk}^G)^2 + (\delta h_{ijk} - \delta h_{ijk}^G)^2] \\ &+ \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \frac{\lambda_{conf}^+}{N_{obj}} I_{ijk} (\gamma_{ijk} - \gamma_{ijk}^G)^2 + \frac{\lambda_{conf}^-}{WHK - N_{obj}} \bar{I}_{ijk} \gamma_{ijk}^2 \\ &\qquad + \frac{1}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \sum_{c=1}^{C} I_{ijk} l_c^G \log(p_c). \end{aligned} \tag{2}$$

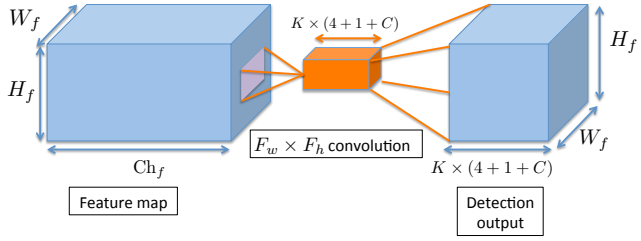
The first part of the loss function is the bounding box regression. $(\delta x_{ijk}, \delta y_{ijk}, \delta w_{ijk}, \delta h_{ijk})$ corresponds to the relative coordinates of anchor-k located at grid center-(i, j). They are outputs of the *ConvDet* layer. The ground truth bounding box δ_{ijk}^G, or $(\delta x_{ijk}^G, \delta y_{ijk}^G, \delta w_{ijk}^G, \delta h_{ijk}^G)$, is computed as:

$$\begin{aligned} \delta x_{ijk}^G &= (x^G - \hat{x}_i)/\hat{w}_k, \quad \delta y_{ijk}^G = (y^G - \hat{y}_j)/\hat{h}_k, \\ \delta w_{ijk}^G &= \log(w^G/\hat{w}_k), \quad \delta h_{ijk}^G = \log(h^G/\hat{h}_k). \end{aligned} \tag{3}$$

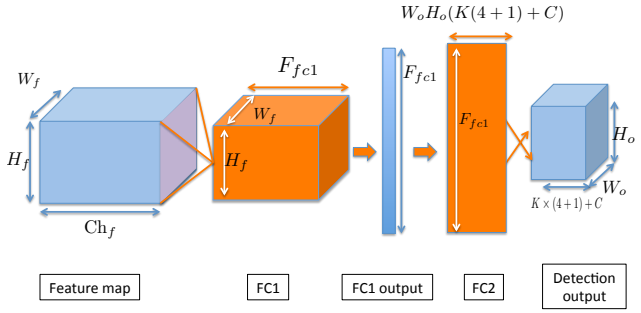
Note that Equation 3 is essentially the inverse transformation of Equation 1. (x^G, y^G, w^G, h^G) are coordinates of a ground truth bounding box. During training, we compare ground truth bounding boxes with all anchors and assign them to the anchors that have the largest overlap (IOU) with each of them. The reason is that we want to select the "closest" anchor to match the ground truth box such that the



(a) Last layer of Region Proposal Network (RPN) is a 1x1 convolution with $K \times (4+1)$ outputs. 4 is the number of relative coordinates, and 1 is the confidence score. It's only responsible for generating region proposals. The parameter size for this layer is $\mathrm{Ch}_f \times K \times 5$.



(b) The *ConvDet* layer is a $F_w \times F_h$ convolution with output size of $K \times (5+C)$. It's responsible for both computing bounding boxes and classifying the object within. The parameter size for this layer is $F_w F_h \mathrm{Ch}_f K(5+C)$.



(c) The detection layer of YOLO [24] contains 2 fully connected layers. The first one is of size $W_f H_f \mathrm{Ch}_f F_{fc1}$. The second one is of size $F_{fc1} W_o H_o K(5+C)$.

Figure 3. Comparing RPN, *ConvDet* and the detection layer of YOLO [24]. Activations are represented as blue cubes and layers (and their parameters) are represented as orange ones. Activation and parameter dimensions are also annotated.

transformation needed is reduced to minimum. I_{ijk} evaluates to 1 if the k-th anchor at position-(i,j) has the largest overlap with a ground truth box, and to 0 if no ground truth is assigned to it. This way, we only include the loss generated by the "responsible" anchors. As there can be multiple objects per image, we normalize the loss by dividing it by the number of objects.

The second part of the loss function is confidence score regression. γ_{ijk} is the output from the *ConvDet* layer, representing the predicted confidence score for anchor-k at position-(i,j). γ^G_{ijk} is obtained by computing the IOU of the predicted bounding box with the ground truth bounding box. Same as above, we only include the loss generated by the anchor box with the largest overlap with the ground truth. For anchors that are not "responsible" for the detection, we penalize their confidence scores with the $\bar{I}_{ijk}\gamma^2_{ijk}$ term, where $\bar{I}_{ijk} = 1 - I_{ijk}$. Usually, there are much more anchors that are not assigned to any object. In order to balance their influence, we use λ^+_{conf} and λ^-_{conf} to adjust the weight of these two loss components. By definition, the confidence score's range is [0, 1]. To guarantee that γ_{ijk} falls into that range, we feed the corresponding *ConvDet* output into a *sigmoid* function to normalize it.

The last part of the loss function is just cross-entropy loss for classification. $l^G_c \in \{0,1\}$ is the ground truth label and $p_c \in [0,1], c \in [1,C]$ is the probability distribution predicted by the neural net. We used *softmax* to normalize the corresponding *ConvDet* output to make sure that p_c is ranged between $[0,1]$.

The hyper-parameters in Equation 2 are selected empirically. In our experiments, we set $\lambda_{bbox} = 5, \lambda^+_{conf} = 75, \lambda^-_{conf} = 100$. This loss function can be optimized directly using back-propagation.

3.4. Neural Network Design

So far in this section, we described the single-stage detection pipeline, the *ConvDet* layer, and the end-to-end training protocol. These parts are universal and can work with various CNN architectures, including VGG16[27], ResNet[16], etc. When choosing the "backbone" CNN structure, our focus is mainly on model size and energy efficiency, and SqueezeNet[18] is our top candidate.

Model size. SqueezeNet is built out of *Fire Modules*, which are comprised of a *squeeze* layer as input, and two parallel *expand* layers as output[18]. The *squeeze* layer is a 1x1 convolutional layer that compresses an input tensor with large channel size to one with the same batch and spatial dimension, but smaller channel size. The *expand* layer is a mixture of 1x1 and 3x3 convolution filters that takes the compressed tensor as input, retrieves the rich features and outputs an activation tensor with large channel size. The alternating *squeeze* and *expand* layers effectively reduces parameter size without losing too much accuracy.

Energy efficiency. Different operations involved in neural network inference have varying energy needs. The most expensive operation is off-chip DRAM access, which uses 100 times more energy than an on-chip SRAM access and floating point operations [15]. Thus, we want to reduce off-chip DRAM accesses as much as possible.

The most straightforward strategy to reduce off-chip

DRAM accesses is to use small models which reduce memory accesses for parameters. An effective way to reduce parameter size is to use convolutional layers instead of fully connected layers when possible. Convolution parameters can be accessed once and reused across all neighborhoods of all data items (if batch$>$1) of the input data. However, fully-connected layers only exposes parameter reuse opportunities in the "batch" dimension, and each parameter is only used on one neighborhood of the input data. Besides model size, another important aspect is to control the size of intermediate activations. Assume the on-chip SRAM size of the computing hardware is 16MB, the SqueezeDet model size is 8MB. If the size of activation output of any two consecutive layers is less than 8MB, then all the memory accesses can be completed in SRAM, no DRAM accesses are needed. A detailed energy efficiency discussion will be provided as supplementary material to this paper.

In this paper, we adopted two versions of the SqueezeNet architecture. The first one is the SqueezeNet v1.1 model[4] with 4.72MB of model size and $> 80.3\%$ ImageNet top-5 accuracy. The second one is a more powerful SqueezeNet variation with squeeze ratio of 0.75, 86.0% of ImageNet accuracy and 19MB of model size [18]. In this paper, we denote the first model as SqueezeDet and the second one as SqueezeDet+. We pre-train these two models for ImageNet classification, then we add two fire modules with randomly initialized weight on top of the pretrained model, and connect to the *ConvDet* layer.

4. Experiments

We evaluated our model on the KITTI [10] object detection dataset, which is designed with autonomous driving in mind. We analyzed our model's accuracy measured by average precision (AP), recall, speed and model size, and then compare with other top ranked methods on the KITTI leader board. Next, we analyzed the trade-off between accuracy and cost in terms of model size, FLOPS and activation size by tuning several key hyperparameters. We implemented our model's training, evaluation, error analysis and visualization pipeline using Tensorflow [1], compiled with the cuDNN [5] computational kernels. The energy efficiency experiments of our model will be reported in the supplementary material.

4.1. KITTI object detection

Experimental setup. In our experiments, unless specified otherwise, we scaled all the input images to 1242x375. We randomly split the 7381 training images in half into a training set and a validation set. Our average precision (AP) results are on the validation set. We used Stochastic Gradient Descent with momentum to optimize the loss function. We set the initial learning rate to 0.01, learning rate decay factor to 0.5 and decay step size to 10000. Instead of using a fixed number of steps, we trained our model all the way until the mean average precision (mAP)[5] on the training set converges, and then evaluate the model on the validation set. Unless specifically specified, we used batch size of 20. We adopted data augmentation techniques such as random cropping and flipping to reduce overfitting. We trained our model to detect 3 categories of object, car, cyclist, pedestrian and used 9 anchors for each grid in our model. At the inference stage, we only kept the top 64 detections with highest confidence, and use NMS to filter the bounding boxes. We used NVIDIA TITAN X GPUs for our experiments.

Average Precision. The detection accuracy, measured by average precision is shown in Table 2. Our proposed SqueezeDet+ model achieved the best AP in all three difficulty levels of cyclist detection on the KITTI leader board. Its mean average precision of all 3 difficulty levels in 3 categories outperforms the top published methods [28, 4]. To evaluate whether *ConvDet* can be applied to other backbone CNN architectures, we appended *ConvDet* to the convolutional trunk of the VGG16 and ResNet50 models. In Table 2, observe that both of these models achieved competitive AP especially on car and cyclist detection. Example of error detections in different types are visualized in Fig. 4.

Recall. Recall is essential for the safety of autonomous vehicles, so we now analyze the recall of our proposed models. For each image with resolution of 1242x375, SqueezeDet generates in total 15048 bounding box predictions. It's intractable to perform non-maximum suppression (NMS) on this many bounding boxes because of the quadratic time complexity of NMS with respect to number of bounding boxes. Thus we only kept the top 64 predictions to feed into NMS. An interesting question to ask is, how does the number of bounding boxes kept affect recall? We tested this with the following experiment: First, we collect all the bounding box predictions and sort them by their confidence. Next, for each image, we choose the top N_{box} bounding box predictions, and sweep N_{box} from 8 to 15048. Then, we evaluate the overall recall for all difficulty levels of all categories. The Recall-N_{box} curve is plotted in Fig. 5. As we could see, for SqueezeDet and its strengthened model, the top 64 bounding boxes' overall recall is already larger than 80%. If using all the bounding boxes, the SqueezeDet models can achieve 91% and 92% overall recall. Increasing the image size by 1.5X, the total number of bounding boxes increased to $35,190$ and the maximum recall using all bounding boxes increases to 95%.

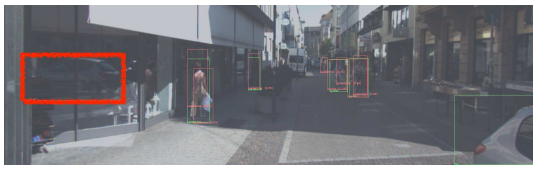
Speed. Our models are the first to achieve real-time inference speed on KITTI dataset. To better understand the

[4] https://github.com/DeepScale/SqueezeNet/

[5] Mean of average precision in 3 difficulty levels (easy, medium, hard) of 3 categories (car, cyclist, pedestrian).

method	car			cyclist			pedestrian			mAP	model size	speed
	E	M	H	E	M	H	E	M	H		(MB)	(FPS)
SubCNN [28]	90.8	**89.0**	79.3	79.5	71.1	62.7	83.3	71.3	66.4	77.0	-	0.2
MS-CNN [4]	90.0	89.0	76.1	84.1	75.5	66.1	83.9	73.7	68.3	78.5	-	2.5
PNET*	81.8	83.6	74.2	74.3	58.6	51.7	77.2	64.7	60.4	69.6	-	10
Pie*	89.4	89.2	74.2	84.6	76.3	67.6	**84.9**	**73.2**	67.6	78.6	-	0.83
FRCN+VGG16 [2]	92.9	87.9	77.3	-	-	-	-	-	-	-	485	1.7
FRCN+Alex [2]	**94.7**	84.8	68.3	-	-	-	-	-	-	-	240	2.9
SqueezeDet (ours)	90.2	84.7	73.9	82.9	75.4	72.1	77.1	68.3	65.8	76.7	**7.9**	**57.2**
SqueezeDet+ (ours)	90.4	87.1	78.9	**87.6**	**80.3**	**78.1**	81.4	71.3	**68.5**	**80.4**	26.8	32.1
VGG16 + *ConvDet* (ours)	93.5	88.1	79.2	85.2	78.4	75.2	77.9	69.1	65.1	79.1	57.4	16.6
ResNet50 + *ConvDet* (ours)	92.9	87.9	**79.4**	85.0	78.5	76.6	67.3	61.6	55.6	76.1	35.1	22.5

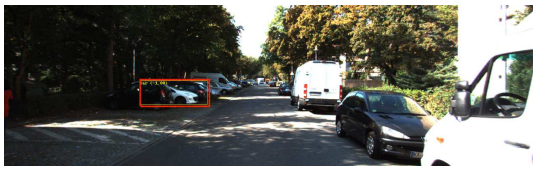
Table 2. Summary of detection accuracy, model size and inference speed of different models on KITTI object detection challenge. Our average precision results are on the validation set. * denotes that it is from an anonymous submissions, thus citation is not available.



(a) Example of a background error. The detector is confused by a car mirrored in the window.

(b) Classification error. The detector predicts a cyclist to be a pedestrian.

(c) Localization error. The predicted bounding box doesn't have an IOU > 0.7 with the ground truth.



(d) Missed object. The missed car is highly truncated and overlapped with other cars.

Figure 4. Example of detection errors.

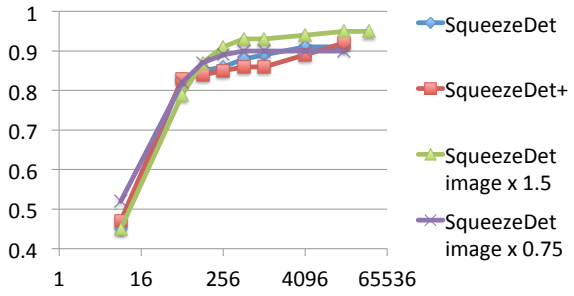


Figure 5. Overall recall vs N_{obj} for SqueezeDet and SqueezeDet+ models. We also tried to re-scale the input image by 1.5X and 0.75X. SqueezeDet and SqueezeDet+ model achieved the best recall of 0.91 and 0.92 with all bounding boxes. SqueezeDet with 1.5X image resolution achieved 0.95. SqueezeDet with 0.75X image resolution achieved 0.90.

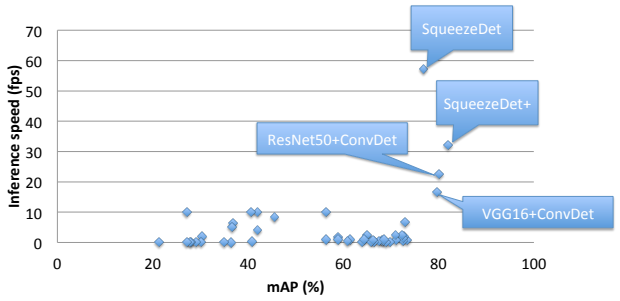
landscape, we collected statistics of 40 submissions of cyclist detection on the KITTI dataset, plotted their inference speed vs mean average precision of three difficulty levels of the cyclist category in Fig.6(a). At the time when this paper was written, the fastest model on the KITTI leader board is an anonymous submission named PNET with 10FPS of inference speed. Our proposed SqueezeDet model achieved 57.2 FPS with much better accuracy compared with PNET. With the stronger SqueezeDet+ model, we still obtained a speed of 32.1 FPS. With the VGG16 and ResNet50 models, augmented by ConvDet, the inference speed is slightly slower, but still faster than all the existing KITTI submissions, as can be seen in Table 2 and Fig.6(a).

Model size. As model size is not reported on the KITTI leader board, We compare our proposed models with Faster-RCNN based models from [2]. We plotted the model size and their mean average precision for 3 difficulty levels of the car category in Fig. 6(b) and summarized them in Table 2. As can be seen in Table 2, the SqueezeDet model is 61X smaller than the *Faster R-CNN + VGG16* model, and it is 30X smaller than the *Faster R-CNN + AlexNet* model. In fact, almost 80% of the parameters of the VGG16 model are from the fully connected layers. Thus, after we replace the fully connected layers and RPN layer with *ConvDet*, the model size is only 57.4MB. Compared with YOLO [24]

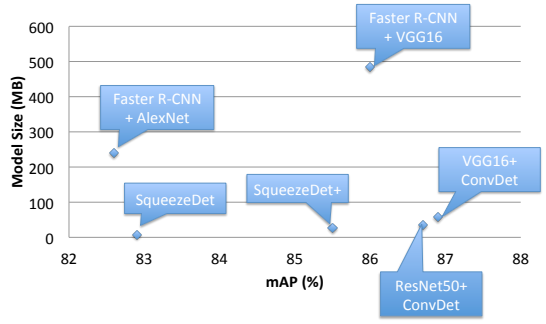
DSE	mAP (%)	Speed (FPS)	FLOPs $\times 10^9$	Model Size (MB)	Activation Memory Footprint (MB)
SqueezeDet	76.7	57.2	9.7	7.9	117.0
scale-up	72.4	31.3	22.5	7.9	263.3
scale-down	73.2	92.5	5.3	7.9	65.8
16 anchors	66.9	51.4	11.0	9.4	117.4
SqueezeDet+	80.4	32.1	77.2	26.8	252.7

Table 3. Design space exploration for SqueezeDet. Different approaches with their accuracy, FLOPs per image, inference speed, model size and activation memory footprint. The speed, FLOPS and activation memory footprint are measured for batch size of 1. We used mean average precision (mAP) to evaluate the overall accuracy on the KITTI object detection task.

which is comprised of 24 convolutional layers, two fully connected layers with a model size of 753MB, SqueezeDet, without any compression, is 95X smaller.



(a) Inference speed vs mean average precision for cyclist detection. Each point represents one method's speed-vs-accuracy tradeoff.



(b) Model size vs. mean average precision for car detection. Each point on this plane represents a method's model size and accuracy tradeoff.

Figure 6. Comparison of different methods' model size, inference speed, and accuracy on the KITTI dataset.

4.2. Design space exploration

We conducted design space exploration to evaluate the influence of some key hyper-parameters on our model's overall detection accuracy (measured in mAP). Meanwhile, we also investigated the "cost" of these variations in terms of FLOPs, inference speed, model size and memory footprint. The results are summarized in Table 3, where the first row is our SqueezeDet architecture, subsequent rows are modifications to SqueezeDet, and the final row is SqueezeDet+.

Image resolution. For object detection, increasing image resolution is often an effective approach to improve detection accuracy [2], but larger images lead to larger activations, more FLOPs, longer training times, etc. To evaluate some of these tradeoffs, in our experiments we scaled the image resolution by 1.5X and 0.75X receptively. With larger input images, the training becomes much slower, so we reduced the batch size to 10. As we can see in Table 3, scaling up the input image actually decreases the mAP and also leads to more FLOPs, lower speed, and larger memory footprint. We also performed an experiment with decreasing the image size. Scaling down the image leads to an impressive 92.5 FPS of inference speed and a smaller memory footprint, though it suffers from a 3 percentage point drop in mean-average precision.

Number of anchors. Another hyper-parameter to tune is the number of anchors. Intuitively, the more anchors used, the more bounding box proposals are generated, and this should result in a better accuracy.In our experiments illustrated in Table 3, using more anchors actually leads to lower accuracy; however, it also shows that for models that use *ConvDet*, increasing the number of anchors only modestly increases the model size, FLOPs, and memory footprint.

Model architecture. As we discussed before, by using a more powerful backbone model with more parameters significantly improved accuracy (See Table 3), but this modification also costs substantially more in terms of FLOPs, model size and memory footprint.

5. Conclusion

We proposed SqueezeDet, a fully convolutional neural network for real-time object detection. We integrated the region proposition and classification into *ConvDet*, which is orders of magnitude smaller than its fully-connected counterpart. With the constraints of autonomous driving in mind, our proposed SqueezeDet and SqueezeDet+ models are designed to be small, fast, energy efficient, and accurate. On all of these metrics, our models advance the state-of-the-art.

Acknowledgments

Research partially funded by the DARPA Award Number HR0011-12-2-0016, Berkeley Deep Drive (BDD) Industry Consortium, Samsung Global Research Outreach (GRO) program, and BMW.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] K. Ashraf, B. Wu, F. N. Iandola, M. W. Moskewicz, and K. Keutzer. Shallow networks for high-accuracy road object-detection. *arXiv:1606.01561*, 2016.

[3] V. Badrinarayanan, A. Handa, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for robust semantic pixel-wise labelling. *arXiv preprint arXiv:1505.07293*, 2015.

[4] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European Conference on Computer Vision*, pages 354–370. Springer, 2016.

[5] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cuDNN: efficient primitives for deep learning. *arXiv:1410.0759*, 2014.

[6] N. Dalal and B. 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.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. 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.

[8] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IJCV*, 2010.

[9] 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] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

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

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

[13] R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 437–446, 2015.

[14] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Supplementary material: Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[15] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: efficient inference engine on compressed deep neural network. *arXiv:1602.01528*, 2016.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.

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

[18] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*, 2016.

[19] F. N. Iandola, M. W. Moskewicz, S. Karayev, R. B. Girshick, T. Darrell, and K. Keutzer. DenseNet: implementing efficient convnet descriptor pyramids. *arXiv:1404.1869*, 2014.

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

[21] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[22] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[23] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[24] J. Redmon, S. Divvala, R. Girshick, and A. 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.

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

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

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

[28] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. Subcategory-aware convolutional neural networks for object proposals and detection. *arXiv:1604.04693*, 2016.