Project Paper
Realtime Object Detection on the NAO Robot

Humboldt-University of Berlin

Department of Computer Science
Adaptive Systems Group

Anh Thu Nguyen
March 5, 2020

Declaration

I declare that this research project was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Anh Thu Nguyen*)

Contents

Introduction

The human visual cognition allows us to look at an image and instantly know which objects are in the image, where they are and what they do. This skill enables us to make decisions and perform complex tasks in real-time such as playing football. The goal of a real-time object detection system is to enable a robot to do the same. However, many current object detection methods are focused on improving the detection accuracy above all and are therefore very resource intensive. Those systems can not operate on embedded devices like smartphones and robots such as the NAO. Those devices have usually only very limited memory and computing capacities. Often they are not equipped with a GPU. Therefore we need a real-time object detection system which also works on a CPU with little memory usage.

In this work we will take a look at two state-of-the-art real-time object detection solutions based on neural networks which are specifically designed for embedded devices like the NAO robot. The first network is Tiny-DSOD by Li, Yuxi, et al., 2018[14] and the second network is SqueezeDet by Wu, Bichen, et al., 2017[26]. We will analyze both networks and try to fuse them in order to get an even better result than each of the single networks. Afterwards we will evaluate the fused network on the dataset KITTI[6]. Finally we will compare the result of the combined network with the results of the original networks.

In section 2 and 3 we present the core ideas of Tiny-DSOD and SqueezeDet respectively. Our experiment and results on the fused network are presented in section 4. Finally we compare the results and conclude our work in section 5.

Tiny-DSOD

The network Tiny-DSOD by Li, Yuxi, et al.[14] is a lightweight object detector for resource-restricted usages, which is trained from scratch. The architecture of Tiny-DSOD is based on the Deeply Supervised Object Detection (DSOD)[20] framework and consists of the depth-wise Dense Block (DDB) based backbone part and depth-wise Feature Pyramid Network (D-FPN) based front-end part. On the Pascal VOC 2007 dataset the network achieves 72.1% mAP with only 0.95M parameters and 1.06B FLOPs. For comparison Faster-RCNN has 134.70M and YOLO has 188.25M parameters and achieve similar results on the same dataset.

2.1 Backbone

The backbone part of Tiny-DSOD is inspired by DenseNet[9]. The idea of DenseNet is to exploit the potential of the network through feature reuse. These networks have some advantages such as reduced number of parameters and no need to learn redundant feature maps. DenseNet is divided into Dense Blocks, a group of connected layers which contain batch normalization, ReLU and convolution layers. In these blocks, the layers are densely connected together, which means that each layer receives all previous feature maps as input. Figure 2.2 shows one Dense Block in DenseNet. The dimensions of the feature maps remain constant within a block and only the number of filters changes. The growth rate defines the number of outputs. Consequently, the growth rate regulates how much new information each layer contributes to the global state. The layers between the blocks are called Transition Layers and take care of the downsampling by applying batch normalization and pooling layers. Figure 2.1 presents the pipeline in DenseNet with three Dense Blocks.

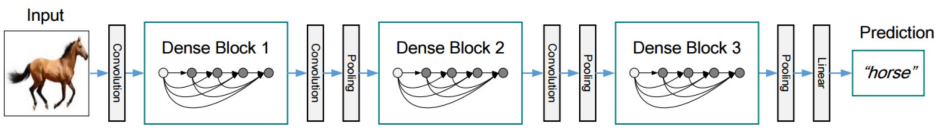


Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

Figure 2.1: A deep DenseNet with 3 Dense Blocks.[9]

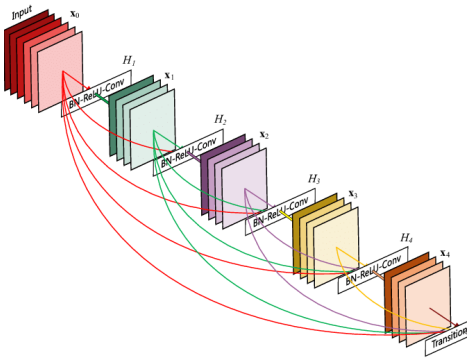


Figure 2.2: A 5-layer dense block with a growth rate of k = 4.[9]

Tiny-DSOD utilizes depth-wise Dense Blocks (DDB), which is a combination of the Dense Block and the depth-wise Separable Convolution (S-CONV). It processes a normal convolution in two parts to increase the number of channels for the output. The first part is depth-wise convolution and the second part is point-wise convolution. The concept is shown in Figure 2.3. The main advantage is to transform the image only once and not over and over again, thus computing power is reduced.

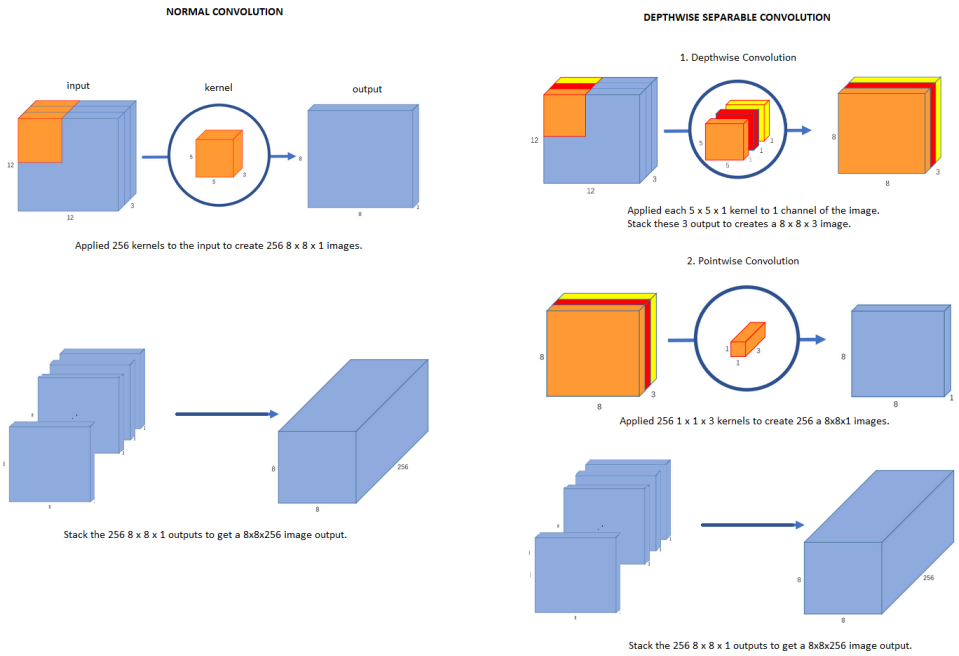
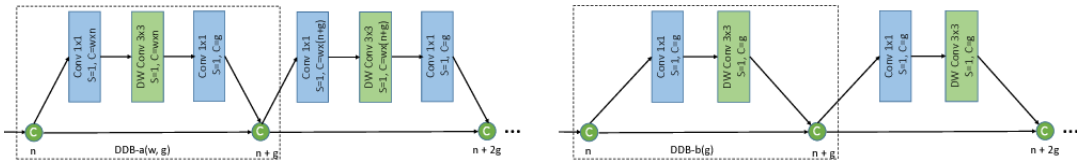


Figure 2.3: depth-wise Separable Convolution[24]

During the process of development of Tiny-DSOD two types of DDB units are introduced: DDB-a and DDB-b. DDB-a is similar to the Bottleneck Residual Block from MobileNet-v2[19], which contains an expansion 1x1 convolution layer, a 3x3 depth-wise convolution layer and a linear 1x1 convolution layer. However, the DDB-a has two main disadvantages: high complexity and redundancy among model parameters. To mitigate these problems the DDB-b was introduced. This is shown in Figure 2.4, containing only a 1x1 convolution layer and a 3x3 depth-wise convolution layer. DDB-b is more efficient and accurate than DDB-a, which is verified in experiments from Li, Yuxi, et al. in their paper[14], section 4.2. Moreover, the complexity of stacked DDB-b blocks is smaller than the ones of DDB-a.

The growth rate g affects the resource consumption and therefore should be small. However, small growth rate g has negative consequence on the discrimination function. Tiny-DSOD adopts the variational growth rate strategy based on CondenseNet[8] by allocating a smaller g to shallower stages with large dimension size, and increasing g linearly when the stage goes deeper.



(a) stacked DDB-a(w, g) (b) stacked DDB-b(g)

Figure 2.4: Illustration of the DDB with (a) is stacked DDB-a parameterized by growth rate g and expand ratio w. (b) is stacked DDB-b parameterized by growth rate g. n is the block input channel number, S means the stride of convolution and C means the number of output channels. Numbers under the concatenating node (green C) means the number of output channels after concatenation.[14]

The Table 2.1 shows the structure of the backbone with DDB-b(g). Each convolution layer is followed by a batch normalization and a ReLU layer. In the extractor part there are four dense stages, each stage containing several DDB-b blocks. For computing and parameter efficiency each dense stage is followed by one transition layer, which fuses channel-wise information from the last stage.

	Module name	Output size	Component
Stem	Convolution	64 x 150 x 150	3 x 3 conv, stride 2
	Convolution	64 x 150 x 150	1 x 1 conv, stride 1
	Depth-wise convolution	64 x 150 x 150	3 x 3 dwconv, stride 1
	Convolution	128 x 150 x 150	1 x 1 conv, stride 1
	Depth-wise convolution	128 x 150 x 150	3 x 3 dwconv, stride 1
	Pooling	128 x 75 x 75	1 x 1 conv, stride 2
Extractor	Dense stage 0	256 x 75 x 75	DDB-b(32) * 4
	Transition layer 0	128 x 38 x 38	1 x 1 conv, stride 1 2 x 2 max pool, stride 2
	Dense stage 1	416 x 38 x 38	DDB-b(48) * 6
	Transition layer 1	128 x 19 x 19	1 x 1 conv, stride 1 2 x 2 max pool, stride 2
	Dense stage 2	512 x 19 x 19	DDB-b(64) * 6
	Transition layer 2	256 x 19 x 19	1 x 1 conv, stride 1
	Dense stage 3	736 x 19 x 19	DDB-b(80) * 6
	Transition layer 3	64 x 19 x 19	1 x 1 conv, stride 1

Table 2.1: Tiny-DSOD backbone with input 3 x 300 x 300. In the "Component" column, the symbol " * " after block names indicates that block repeats number times.[14]

2.2 Front-end

The front-end part includes a lightweight FPN named depth-wise FPN (D-FPN), which is based on the idea of the Feature Pyramide Network[15]. Figure 2.5 illustrates the structure of the D-FPN structure, which consists of downsampling and upsampling.
The reverse-path upsampling has been demonstrated to be very helpful for small objects' detection in many works, for example in Feature Pyramide Network[15], in Deconvolutional Single Shot Detector (DSSD) [5] or in Context-Aware Single-Shot Detector (CSSD)[27] and it is usually implemented by means of deconvolution with transposed convolutional layer, which greatly increases the model complexity. To overcome this problem, Li, Yuxi, et al. propose in Tiny-DSOD a cost-efficient solution, which is shown in top-right of Figure 2.5. This operation could be formulated in the equation (2.1) as follows:

$$F_c(x, y) = W_c * \sum_{(m,n)\in\Omega} U_c(m, n)\tau(m, sx)\tau(n, sy), \tag{2.1}$$

where F_c is the c-th channel of the output feature map and U_c is the corresponding channel of the input. W_c is the c-th kernel of depth-wise convolution and * denotes the spatial convolution. W is the co-ordinate set of input features and s is the resampling coefficient in this layer. $\tau(a, b) = max(0, 1 - |a - b|)$ is the differentiable bilinear operator.[14]

There is an uncertain point in the paper. As illustrated in Figure 2.5 the down- and upsampling x2 is not fluently symmetric by starting with layer size 128 x 38 x 38. The downsampling from 128 x 19 x 19 is 128 x 9 x 9. This conflict can be handle by using different paddings.

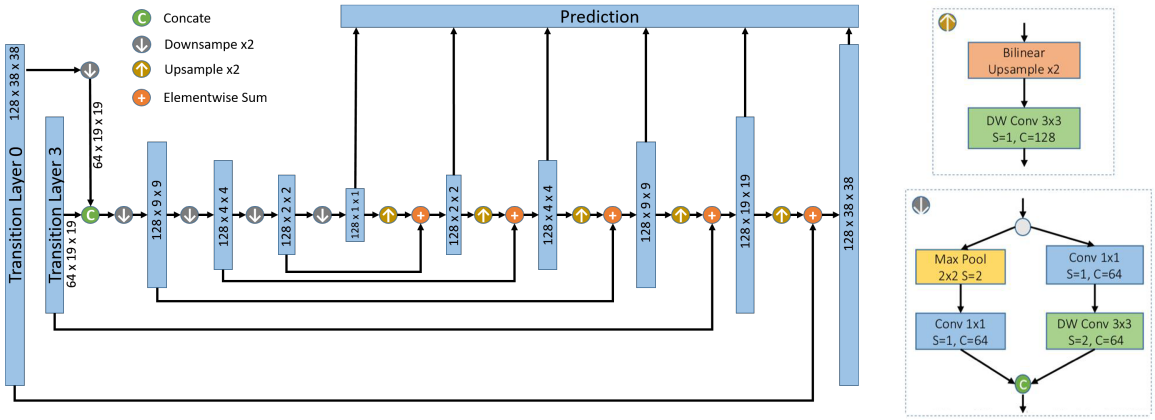
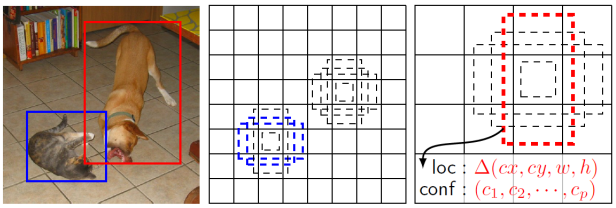


Figure 2.5: The left part is the structure of D-FPN, while the right part further depicts the details of the upsampling (top-right) and downsampling (bottom-right) modules in D-FPN. Note that both samplings are by factor 2; S is the stride of convolution and C is the number of output channels.[14]

2.3 Prediction Layer

The prediction layer was not explicitly described in the paper, but Tiny-DSOD is inspired by Deeply Supervised Object Detector (DSOD)[20]. In their GitHub implementation [28] the authors confirmed that the code is based on the SSD and DSOD framework. It was developed following the Single-Shot Detection (SSD)[16] framework. However, DSOD and SSD have the same prediction method. SSD uses multi-scale feature maps to detect objects independently. After extracting a feature map n x m, SSD applies 3 x 3 convolution filters to each cell, which results in k bounding boxes with different sizes and aspect ratios. For each of the bounding boxes, c class scores and four offsets relative to the original default bounding box shape are computed. Finally SSD calculates $(c+4)k$ x n x m outputs for each feature map. The concept is shown in Figure 2.6.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 2.6: SSD: Multiple bounding boxes for localization (loc) and confidence (conf)[16]

The loss function is formulated in equation (2.2) and consists of two terms: Lconf and Lloc, where N is number of the matched default boxes.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + L_{loc}(x, l, g)) \tag{2.2}$$

The localization loss L_{loc} is similar to the one in Faster R-CNN and comprise the matched default boxes which is the smooth L1 loss between the predicted box (l) and the ground-truth box (g) parameters . These parameters include the offsets for the center point (cx, cy), width (w) and height (h) of the bounding box. Equation 2.3 shows the localization loss. [23]

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in cx,cy,w,h} x_{ij}^{k} \; smooth_{L1}(l_i^m - \hat{g}_j^m) \tag{2.3}$$

$$\text{where} \quad \hat{g}_j^{cx} = \frac{g_j^{cx} - d_i^{cx}}{d_i^w} \quad \hat{g}_j^{cy} = \frac{g_j^{cy} - d_i^{cy}}{d_i^h} \quad \hat{g}_j^w = log(\frac{g_j^w}{d_i^w}) \quad \hat{g}_j^h = log(\frac{g_j^h}{d_i^h})$$

$$\text{with} \quad smooth_{L1} = \begin{cases} 11|x|if|x| > \alpha \\ \frac{1}{\alpha}x^2 if|x| < \alpha \end{cases}$$

L1 and L2 are two loss functions used to minimize the error. L1 loss function represents Least Absolute Deviations and L2 loss function represents Least Square Errors. Smooth L1 loss behaves as L1 loss when the absolute value of the argument is high, and it behaves like L2 loss when the absolute value of the argument is close to zero. α is set to 1 by cross validation. [1]

The confidence loss L_{conf} which is the softmax loss over multiple classes confidences (c). $x_{ij}^p = \{1, 0\}$ is an indicator for matching i-th default box to the j-th ground truth box of category p.[23]

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} (x_{ij}^p log(\hat{c}_i^p)) - \sum_{i \in Neg} log(\hat{c}_i^0) \tag{2.4}$$

$$\text{where} \quad \hat{c}_i^p = \frac{exp(c_i^p)}{\sum_p exp(c_i^p)}$$

SqueezeDet

SqueezeDet is a fully convolutional neural network for object detection that aims to satisfy the following issues: high accuracy, extreme speed, small model size and good energy efficiency. Inspired by YOLO[17] SqueezeDet is a single-stage detection pipeline that does region proposal and classification of multiple objects within the image by one single network. SqueezeDet is achieving the same accuracy as previous much larger models. Its model size is 30x smaller than the Faster R-CNN+AlexNet model[12] and the network consumes 84x less energy than the Faster R-CNN+VGG16 model[21]. The more powerful version of the SqueezeDet is SqueezeDet+, which is 19.7x faster than Faster R-CNN+VGG16.[26]. The architecture of SqueezeDet is based on two parts. The first part is the backbone CNN SqueezeNet[10] with two additional layers to extract the feature maps from input images. The second part is ConvDet[1] to compute a large amount of object bounding boxes called anchors and to predict their categories.

3.1 Backbone

The backbone CNN is selected by its model size and its energy efficiency structure. SqueezeNet is based on fire modules, which allow to reduce the parameter size without any significant accuracy loss. The fire module is shown in Figure 3.1 and contains a squeeze layer and two parallel expand layers. The squeeze layer is an 1 x 1 convolution layer and the expand layers is a combination of 3 x 3 and 1 x 1 filters, whose results are concatenated.

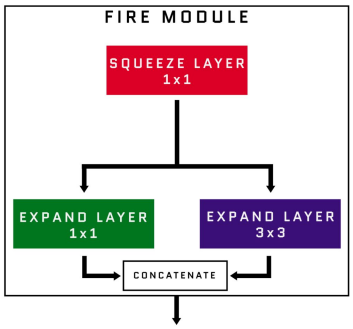


Figure 3.1: Fire module in [10]

There are two implemented versions of the SqueezeNet architecture. The first one is SqueezeNet and the second one is SqueezeNet+. Both versions were pre-trained on

[1]The ConvDet part is similar to the one of Region Proposal Network (RPN) in Faster R-CNN[18].

ImageNet[4], then two fire modules with randomly initialized weight on their top were added. SqueezeNet has 4.72MB of model size and $> 80.3\%$ accuracy on ImageNet[4]. SqueezeNet+ is a more powerful SqueezeNet variation with 19MB of model size and 86.0% accuracy on ImageNet[4]. Tables 3.1 and 3.2 show layers details of the two backbones. Both backbones were connected to the ConvDet layer. The resulting networks are named accordingly SqueezeDet and SqueezeDet+.

Module name	Output size	Component
Convolution 1	64 x 624 x 192	3 x 3 conv, stride 2
Pooling 1	64 x 312 x 96	3 x 3 max pool, stride 2
Fire Module 2	128 x 312 x 96	1 x 1 conv, stride 1, filters 16 1 x 1 conv, stride 1, filters 64 3 x 3 conv, stride 1, filters 64
Fire Module 3	128 x 312 x 96	1 x 1 conv, stride 1, filters 16 1 x 1 conv, stride 1, filters 64 3 x 3 conv, stride 1, filters 64
Pooling 3	128 x 156 x 48	3 x 3 max pool, stride 2
Fire Module 4 Fire Module 5	256 x 156 x 48	1 x 1 conv, stride 1, filters 32 1 x 1 conv, stride 1, filters 128 3 x 3 conv, stride 1, filters 128
Pooling 5	256 x 78 x 24	3 x 3 max pool, stride 2
Fire Module 6 Fire Module 7	384 x 78 x 24	1 x 1 conv, stride 1, filters 48 1 x 1 conv, stride 1, filters 192 3 x 3 conv, stride 1, filters 192
Fire Module 8 Fire Module 9	512 x 78 x 24	1 x 1 conv, stride 1, filters 64 1 x 1 conv, stride 1, filters 256 3 x 3 conv, stride 1, filters 256
2 Extra Layers: Fire Module 10 Fire Module 11	768 x 78 x 24	1 x 1 conv, stride 1, filters 96 1 x 1 conv, stride 1, filters 384 3 x 3 conv, stride 1, filters 384

Table 3.1: SqueezeDet backbone with input 3 x 1248 x 384

Module name	Output size	Component
Convolution 1	96 x 618 x 185	7 x 7 conv, stride 2
Pooling 1	96 x 308 x 92	3 x 3 max pool, stride 2
Fire Module 2	128 x 308 x 92	1 x 1 conv, stride 1, filters 96
		1 x 1 conv, stride 1, filters 64
Fire Module 3		3 x 3 conv, stride 1, filters 64
Fire Module 5	256 x 153 x 45	1 x 1 conv, stride 1, filters 192
		1 x 1 conv, stride 1, filters 128
		3 x 3 conv, stride 1, filters 128
Fire Module 6	384 x 153 x 45	1 x 1 conv, stride 1, filters 288
		1 x 1 conv, stride 1, filters 192
Fire Module 7		3 x 3 conv, stride 1, filters 192
Fire Module 8	512 x 153 x 45	1 x 1 conv, stride 1, filters 384
		1 x 1 conv, stride 1, filters 256
		3 x 3 conv, stride 1, filters 256
Pooling 8	512 x 76 x 22	3 x 3 max pool, stride 2
Fire Module 9	512 x 76 x 22	1 x 1 conv, stride1, filters 384
		1 x 1 conv, stride 1, filters 256
		3 x 3 conv, stride 1, filters 256
2 Extra Layers:	512 x 76 x 22	1 x 1 conv, stride 1, filters 384
Fire Module 10		1 x 1 conv, stride 1, filters 256
Fire Module 11		3 x 3 conv, stride 1, filters 256

Table 3.2: SqueezeDet+ backbone with input 3 x 1242 x 375

3.2 Prediction

ConvDet layer is an F_w x H_h convolution layer, which is trained to output bounding box coordinates and class probabilities. It is similar to the last layer of Region Proposal Network (RPN) in Faster R-CNN, the difference being that it only generates region proposals. The outputs are four numbers of the relative coordinates and one confidence score. ConvDet can compute directly the detection output, such as bounding boxes, and classify the object within it. Picture 3.2 shows the difference between the last layer of RPN and ConvDet. In SqueezeDet and SqueezeDet+ ConvDet is applied to SqueezeNet, but it can also be applied to other backbone CNNs like VGG16[21] or ResNet50[7].

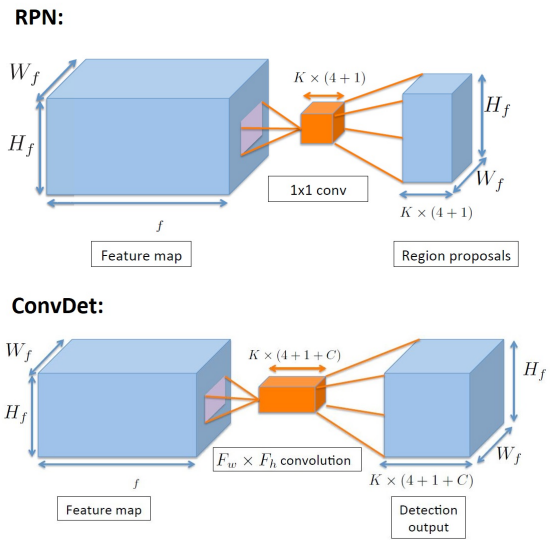


Figure 3.2: Last layer of Region Proposal Network (RPN) is an 1x1 convolution with K x $(4 + 1)$ outputs and ConvDet layer is an F_w x H_h convolution with K x $(4 + 1 +$ C) outputs. [26]

ConvDet works like a sliding window and computes K x $(4 + 1 + C)$ outputs at each position on the feature map. At each position there are K numbers of anchor boxes with pre-selected shapes[2]. ConvDet computes for each k-th anchor four relative coordinates and transforms the anchor to the new position and shape based on the ground truth boxes. Figure 3.3 illustrates the bounding box transformation and Equation (3.1) describes the transformation more formally.

$$\begin{aligned} x_i^p &= \hat{x}_i + \delta x_{ijk}, \\ y_j^p &= \hat{y}_i + \delta y_{ijk}, \\ w_k^p &= \hat{w}_k exp(\delta w_{ijk}), \\ h_k^p &= \hat{w}_k exp(\delta h_{ijk}) \end{aligned} \tag{3.1}$$

Each k-th anchor can be described as $(\hat{x}_i, \hat{y}_j, \hat{w}_k, \hat{h}_k), i \in [1, W], j \in [1, H], k \in [1, K]$, where $\hat{x}_i, \hat{y}_j$ is the coordinates of the grid center (i, j) and $(\hat{w}_k, \hat{h}_k)$ are the width and the height of the k-th anchor box. The four relative coordinates are $(\delta x_{ijk}, \delta y_{ijk}, \delta w_{ijk}, \delta h_{ijk})$

[2]The anchor box shapes tailored to the dataset are selected with the approach in [2].

and $x_i^p, y_i^p, w_i^p, h_i^p$ are the final predicted bounding box coordinates.

For each bounding box ConvDet calculates one confidence score and C conditional class probabilities, where C is the number of classes. A high confidence score means a high probability of an object from one of the desired classes. The results of the detection are transferred to the next stage in the pipeline of SqueezeDet (Figure 3.4) for the filtering of the predicted bounding boxes.

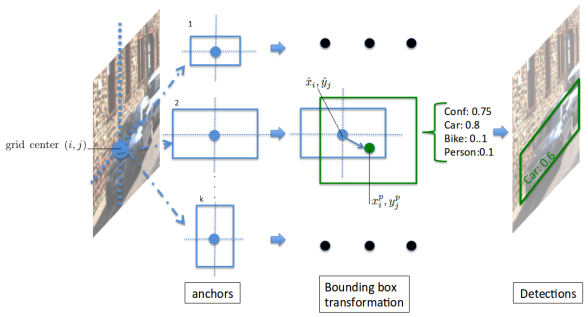


Figure 3.3: Bounding box transformation[26]

ConvDet computes multiple bounding boxes surrounding the objects in the image. To refine the predicted bounding boxes and calculate compact detection boxes, Non-Maximum Suppression (NMS) is used. NMS is a common post-processing technique for fusing all detections corresponding to the same object. NMS uses as input a list of bounding boxes, their corresponding confidence scores and an overlap threshold. The first step is to select the highest confidence score of the pre-prediction and pick that value as the confidence score. The second step is the comparison of all pre-predictions with the selected one and the calculation of the IOU. If the IOU is greater than the threshold, that bounding box will be discarded from the list.[11]. After NMS processing the final results of SqueezeDet are the top N bounding boxes with highest confidence score.

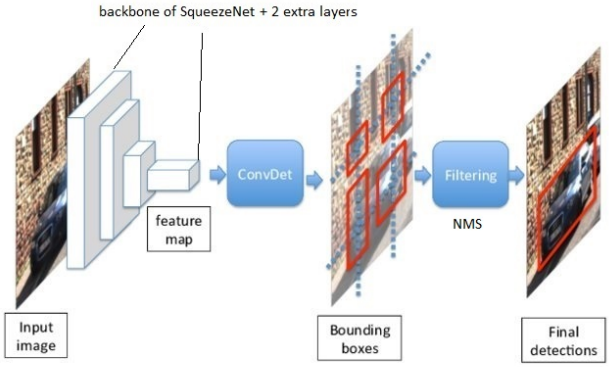


Figure 3.4: SqueezeDet detection pipeline[26]

The loss function of SqueezeDet is formulated in Equation 3.2. The first part is the regression of the scalars for the anchors. The second part is the confidence score regression which uses IOU of ground and predicted bounding boxes. The third part is the penalization of anchors which are not responsible for detection by dropping their confidence score. The last part is the cross entropy.

$$
\begin{aligned}
&\frac{\lambda_{bbox}}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk}[(\delta x_{ijk} - \delta x_{ijk}^{G})^2 + (\delta y_{ijk} - \delta y_{ijk}^{G})^2 + (\delta w_{ijk} - \delta w_{ijk}^{G})^2 + (\delta h_{ijk} - \delta hijk^{G})^2] \\
&+ \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} I_{ijk} \frac{\delta_{conf}^{+}}{N_{obj}} I_{ijk}(\gamma_{ijk} - \gamma_{ijk}^{G}))^2 \\
&+ \frac{\gamma_{conf}^{-}}{WHK - N_{obj}} \bar{I}_{ijk} \gamma_{ijk}^{2} \\
&+ \frac{1}{N_{obj}} \sum_{i=1}^{W} \sum_{j=1}^{H} \sum_{k=1}^{K} \sum_{c=1}^{C} I_{ijk} l_{c}^{G} log(p_c)
\end{aligned}
\tag{3.2}
$$

where $(\delta x_{ijk}, \delta y_{ijk}, \delta w_{ijk}, \delta h_{ijk})$ are the relative coordinates of k-anchor
and $\delta x_{ijk}^{G}, \delta y_{ijk}^{G}, \delta w_{ijk}^{G}, \delta h_{ijk}^{G}$ are the coordinates of the ground truth bounding boxes δ_{ijk}^{G}.

Experiment

After analyzing the two networks we take a look on both implementations from the authors published on GitHub: SqueezeDet in [25] and Tiny-DSOD in [28]. As stated by the authors of SqueezeDet, ConvDet can be applied to other backbone CNNs. Hence we concluded that the architecture backbone from Tiny-DSOD can also be used instead of SqueezeNet. Following this idea, we want to modify the implementation of SqueezeDet by replacing SqueezeNet backbone with Tiny-DSOD backbone. The Figure 4.1 shows the modified pipeline. SqueezeDet was implemented by its authors in Python 2.7 us-

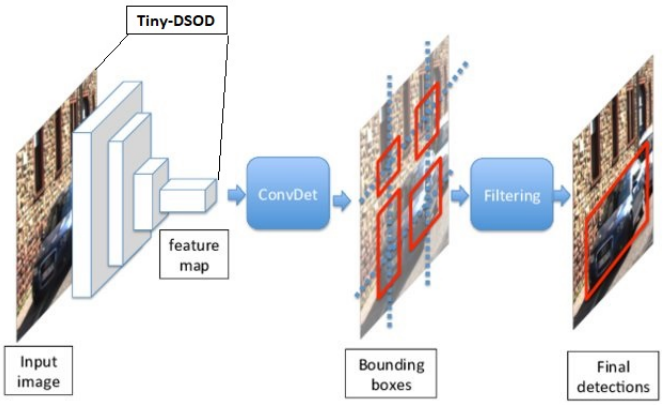


Figure 4.1: The modified SqueezeDet detection pipeline

ing the TensorFlow 1.0 framework[25]. We downloaded and run the code in order to reproduce the results from the paper. By using their pre-trained model of SqueezeDet and SqueezeDet+ we could reproduce their results. We found another implementation of SqueezeDet on Keras in GitHub [22], which is much simpler due to the removal of redundant or unnecessary parts e.g. code for benchmarking against different architectures. For our experiment we will use the implementation of SqueezeDet on Keras and modify it with the backbone of Tiny-DSOD. For the training and the evaluation of our combined network we will use KITTI dataset[6]. This dataset is composed of extremely wide images of size 1248 x 384. To prevent small objects from vanishing, we keep the size of the input image instead of 300 x 300 as in the original Tiny-DSOD backbone. The modified backbone is shown in Table 4.1.

	Module name	Output size	Component
Stem	Convolution	64 x 624 x 192	3 x 3 conv, stride 2
	Convolution	64 x 624 x 192	1 x 1 conv, stride 1
	Depth-wise convolution	64 x 624 x 192	3 x 3 dwconv, stride 1
	Convolution	128 x 624 x 192	1 x 1 conv, stride 1
	Depth-wise convolution	128 x 624 x 192	3 x 3 dwconv, stride 1
	Pooling	128 x 312 x 96	1 x 1 conv, stride 2
Extractor	Dense stage 0	256 x 312 x 96	DDB-b(32) * 4
	Transition layer 0	128 x 156 x 48	1 x 1 conv, stride 1 2 x 2 max pool, stride 2
	Dense stage 1	416 x 156 x 48	DDB-b(48) * 6
	Transition layer 1	128 x 78 x 24	1 x 1 conv, stride 1 2 x 2 max pool, stride 2
	Dense stage 2	512 x 78 x 24	DDB-b(64) * 6
	Transition layer 2	256 x 78 x 24	1 x 1 conv, stride 1
	Dense stage 3	736 x 78 x 24	DDB-b(80) * 6
	Transition layer 3	64 x 78 x 24	1 x 1 conv, stride 1

Table 4.1: The modified Tiny-DSOD backbone with input 3 x 1248 x 384.

4.1 Training and Results

For the training we kept the parameters from SqueezeDet on Keras for our combined network. Therefore, the number of anchor boxes is $K = 9$, the number of top bounding boxes is $N = 64$ and the threshold for dropout is 0.5. The input size is 1248 x 384 and the batch size is 16. The 7381 training images from the KITTI dataset is randomly split half into training set and half into validation set. We trained our model to detect three categories of objects: cyclist, pedestrian and car.
We took the results of Tiny-DSOD and SqueezeDet from the original papers for our comparison. In Table 4.2 we also added MS-CNN[3] and fire-FRD-CNN[13] to highlight the size of the two small networks. Tiny-DSOD and SqueezeDet achieves a competitive mAP result of 77.0% mAP and 76.6 % mAP, while both models have less than half of parameters. It should be noted that Tiny-DSOD achieves the highest accuracy on the category "cars", which are the main objects in the KITTI dataset.

The combined network achieves worse results in every class except cyclists, but with significantly less parameters and therefore much higher inference time. It is important to note that we trained the network for just 100 epochs due to time constraints. We suspect that with more training and perhaps data augmentation we can achieve comparable results to Tiny-DSOD with much less parameters.

Method	Params	FLOPs	car AP	cyclist AP	pedestrian AP	mAP
MS-CNN[3]	80M	-	85.0	75.2	75.3	78.5
fire-FRD-CNN[13]	14M	-	87.07	83.7	**76.73**	**82.5**
SqueezeDet[26]	1.98M	9.7B	82.9	76.8	70.4	76.7
SqueezeDet+[26]	6.71M	77.2B	85.5	**82.0**	73.7	80.4
Tiny-DSOD[14]	**0.85M**	4.1B	**88.3**	73.6	69.1	77.0
our Tiny-Det	774,344	1.613.730	69.1	96.2	29.3	64.87

Table 4.2: Results on KITTI 2D Object Detection (the models are trained on half KITTI trainval and test on the other half)

Conclusion

Tiny-DSOD and SqueezeDet used different approaches with the common goal, an efficient object detection network for less powerful systems like a NAO robot. While Tiny-DSOD is focused on the problem of training object detector from scratch, SqueezeDet focuses on the ConvDet layer, which is a convolutional layer that is trained to output bounding box coordinates and class probabilities. The end-to-end training protocol of SqueezeDet is universal and can work with various CNN architectures. Thus we combined the two methods to create a new object detector with usable results and much fewer parameters. In the future more experiments regarding hyper parameter tuning and data augmentation should be done in order to see if we can get even better results from such a small object detection network.

Bibliography

[1] 2018. https://stats.stackexchange.com/questions/351874/how-to-interpret-smooth-l1-loss.

[2] Khalid Ashraf, Bichen Wu, Forrest N Iandola, Mattthew W Moskewicz, and Kurt Keutzer. Shallow networks for high-accuracy road object-detection. *arXiv preprint arXiv:1606.01561*, 2016.

[3] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European conference on computer vision*, pages 354–370. Springer, 2016.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

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

[6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

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

[8] Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2752–2761, 2018.

[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[10] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[11] Sambasivarao. K, 2019. https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c.

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

[13] Wei Li, Kai Liu, Lin Yan, Fei Cheng, YunQiu Lv, and LiZhe Zhang. Frd-cnn: Object detection based on small-scale convolutional neural networks and feature reuse. *Scientific reports*, 9(1):1–12, 2019.

[14] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-dsod: Lightweight object detection for resource-restricted usages. *arXiv preprint arXiv:1807.11013*, 2018.

[15] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

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

[17] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[18] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[19] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[20] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod: Learning deeply supervised object detectors from scratch. In *Proceedings of the IEEE international conference on computer vision*, pages 1919–1927, 2017.

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

[22] Omnius Engineering Team. Squeezedet on keras, 2018. `https://github.com/omni-us/squeezedet-keras`.

[23] Sik-Ho Tsang. Review: Ssd — single shot detector (object detection), November 2018. `https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11`.

[24] Chi-Feng Wang. A basic introduction to separable convolutions, 2018. `https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728`.

[25] Bichen Wu. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. GitHub, 2017. `https://github.com/BichenWuUCB/squeezeDet`.

[26] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 129–137, 2017.

[27] Wei Xiang, Dong-Qing Zhang, Heather Yu, and Vassilis Athitsos. Context-aware single-shot detector. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1784–1793. IEEE, 2018.

[28] Jiuwei Li Yuxi Li, Jianguo Li and Weiyao Lin. Tiny-dsod: Lightweight object detection for resource restricted usage. github, 2018. `https://github.com/lyxok1/Tiny-DSOD`.