Towards State-of-the-Art Object Detection: A Multistage Improvement of Single Shot Multibox Detector

Manikandan R
Research and Development Center
Hitachi India Pvt Ltd
Bangalore, India
manikandan@hitachi.co.in

***Abstract*—Object detection is a fundamental problem in computer vision with wide range of applications, from image tagging and indexing to applications related to surveillance, autonomous vehicles, and robotics. We address this problem by proposing practical improvements to an existing state-of-the-art object detection method i.e. Single Shot Detector (SSD) in multiple stages with focus on plurality of application level requirements. As the first contribution, we propose to improve performance efficiency by automatically selecting the scales of the default boxes. The scales of the default boxes determine the absolute size of the objects being searched for at different layers of the network, respectively, and hence are better off being tuned for the specific statistics of the object and the current data under inspection. As a second contribution, we focus on compressing the detection model to accommodate it for the low memory applications with limited compromise in accuracy. As a third contribution, we focus on reducing computations by pruning the search space using semantically-nearest neighbor images from the training set. We show the advantages of these contributions through quantitative evaluations of multiple state of the art datasets across multiple application domains with, the first one leads to improved performance, the second and third contribution pavying way for reduced memory requirement and higher detection speeds.**

***Index Terms*—Single Shot Detector, Compression, Scale Selection, Covex Optimization**

I. Introduction

Visual recognition technologies such as image classification, i.e. predicting the objects, scene, attributes etc. present in an image, and object detection, i.e. predicting the exact location of a given object in an image with a bounding box, have graduated from proof of concept level to being productizable. Starting from the seminal work by [1], proposing the first high performance deep convolutional neural network (CNN) for image classification, the algorithms and systems have very far, with better performing networks for image classification, such as the GoogLeNet [2], VGGNet [3] and ResNet [4], as well as networks for object detection, such as Fast-RCNN [5], Faster-RCNN [7], R-FCN [8], SSD and its variants [9], [10] , YOLO and its variants [11], RetinaNet [12] and very recently Mask-RCNN [13]. The high performing object detection modules now find applications in upcoming autonomous cars, visual surviellance etc.

A. Contributions

With the target of industrial applications, in the present paper, we propose to improve the implementation of the popular and high performing Single Shot Detector (SSD) by [9] with three contributions. As the contribution, we propose to automatize the selection of some key parameters, i.e. the scales of the default boxes that control what sized objects will be searched for at the different resolution CNN layers. As a second contribution we propose a multilayer pruning framework to compress the model catering lower memory needs. As the final contribution, we aim to improve the detection speed of the object detector in two phases initially by retrieving semantically similar images from the training set and using the spatial distribution of the object in the retrieved images as a prior for the search space in the current image. All these stages are generic and could be used with any of the existing detectors with some minor changes. We validate the improvements over four benchmark data sets namely KITTI [14], TSDB [15]–[17], RDDB [18] and PASCAL VOC [19]. We show empirically that (i) automatic tuning of default boxes leads to improvements in performances over the default parameter settings of SSD and (ii) the Nearest Neighbor based pruning allows us to obtain a trade-off between accuracy and speed of prediction and iii) Multi layer pruning improves the speed of the detector with limited drop in accuracy.

B. Related Work

In this section we present, existing literature related to Object detection and Model Compression in brief.

Object Detection and its Improvements: Object detection has a long history in computer vision with the initial works based on bag words features with nonlinear SVMs [20] and Histogram of Gradient (HOG) features [21], [22]. The current state-of-the-art object detection systems are now exclusively based on deep Convolutional Neural Networks (CNN). Some representative works include Region based CNN and its fast [5] and faster [7] variants, Single Shot Detector (SSD) [9] and You Only Look Once [11]. While the intial detectors focused on using existing deep networks [1] to classify a limited

TABLE I: Dataset statistics of KITTI object detection benchmark used in this work. All the results reported are on the Modified set.

	Training	Validation	Testing
Set-Original	7481	0	7518
Set-Modified	5983	1497	7518

number of object proposals per image, the later methods tried to do both, the object proposal prediction and classification, together in one architecture [5], [7]. In the current generation methods, the architectures do not have object proposals, but do prediction for a large number of possible object positions [9]–[11]. These methods obtain the best performances on public benchmark datasets while being very fast as well. We use the current generation of methods in the current work and build on them.

Object detection and Compression: While there is a large body of works on compression of classification based architectures, very few works are done till date on compression of object detection based deep learning models. Most notable are works by [23] which focuses on compressing SSD by using deconvolution to analyze kernels and active neurons, thereby removing inactive neurons and redundant kernels. Alternatively, [24] proposed a trainable framework for multiclass object detection through knowledge distillation via the introduction of weighed cross entropy and teacher bounded regression for knowledge distillation. Finally, in the very recent work [25], focus on eliminating channels by sampling using multiple methods and fine-tuning the resulting model on the given task. In conclusion, works that focus on SSD, concentrate either only a few layers of the SSD architecture or use compact base networks which result in reduced performance. In our work, we focus on compressing original SSD with VGG16 as its base network.

II. Dataset

In this section we describe various datasets that are used as part of this work.

A. KITTI object detection benchmark

To evaluate on autonomous driving applications we use the well-known KITTI Vision Benchmark Suite [14] for training and evaluating our detection models. The dataset consists of synchronised stereo camera and lidar frames recorded from a moving vehicle with annotations for eight different object classes, showing a wide variety of road scenes with different appearances. We only use the 2D detection dataset train and test the models III. There are 7,518 frames in the KITTI test set whose labels are not publicly available. The labelled training data consist of 7,481 frames which we split into two sets for training and validation (80% and 20% respectively). While the object detection benchmark considers three classes for evaluation: cars, pedestrians and cyclists with 28,742, 4,487, and 1,627 training labels respectively. In this work, we focus on the first two classes respectively.

TABLE II: Dataset statistics of standard Traffic Sign Detection Benchmarks

	Training	Testing
GTSDB [15]	600	300
BTSDB [16]	5905	3101
STSDB [17]	4000	500

B. Traffic Sign detection Benchmark

In addition we also use three standard TSDR benchmarks where the problem detection and classification is harder than KITTI. Table II shows each of the standard benchmark datasets used in this work in terms of their training and testing split sizes. As we can see, compared to GTSDB, BTSDB and STSDB have more annotated frames.

In this work, we have selected 39 unique traffic sign classes as shown in Table III, owing to their widespread usage across the European countries. For our experiments, we use a total of 32 classes of traffic signs from GTSDB, 15 and 27 classes from STSDB and BTSDB. The more in depth description of the class selection could be found in [26].

C. Road Damage Detection Benchmark

With construction applications in target we evaluate on Road Damage Detection Benchmark (RDDB). RDDB dataset used for this work consists of images recorded using mobile smartphone previously by [18] and was made available as part of the 2018 IEEE Big Data Cup. The images recorded are made in a zenith angle taken from inside of the vehicle on looking the road ahead. The dataset statistics is as shown in Table I. The dataset consists of road damage images collected across 7 municipalities in Japan with 8 road damage categories [18]. While the published work suggests collection of the dataset with more than 9K frames, the released dataset consists of total of 7.6K images with 5787 train images and 1813 test images. Since the released dataset doesn't provide any validation split, during initial model development, validation and parameter search stages we further divide the dataset into two splits namely **Set-Original** a.k.a the original dataset and **Set-Modified** involving a training set of 3974, validation set of 1813 and test set of 1813 images. More details on the dataset could be found in [18].

D. PASCAL VOC Benchmark

With general domain in target, we also experiment on PASCAL VOC dataset. The Pascal Visual Object Classes (VOC) Challenge has been an annual event since 2006. The challenge consists of two components: (i) a publicly available dataset of images obtained from the Flickr web site (2013), together with ground truth annotation and standardized evaluation software; and (ii) an annual competition and workshop. There are three principal challenges: classification does the image contain any instances of a particular object class? (where object classes include cars, people, dogs, etc.), detection where are the instances of a particular object class in the image (if any)?, and segmentation to which class does each pixel belong?. In addition, there are two subsidiary challenges (tasters): action

TABLE III: Dataset statistics of Road Damage Detection Benchmark used in this work. All the results reported are on the Modified set.

	Training	Validation	Testing
Set-Original	5787	0	1813
Set-Modified	3974	1813	1813

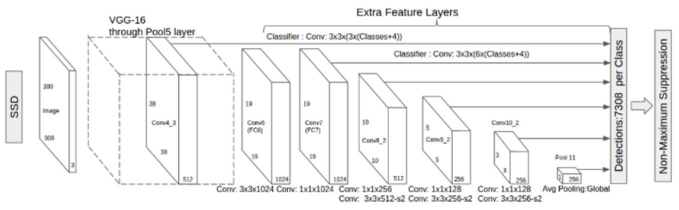


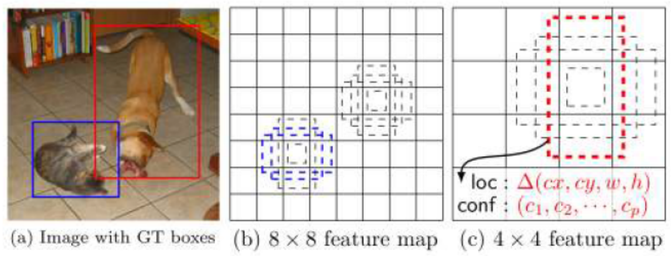
Fig. 1: Architecture of SSD [11]

classification what action is being performed by an indicated person in this image? (where actions include jumping, phoning, riding a bike, etc.) and person layout where are the head, hands and feet of people in this image?. The challenges are issued with deadlines each year, and a workshop held to compare and discuss that years results and methods. We use the detection tasks dataset of the year 2012 and 2007 as these contain the diverse set of images when put together. The object detection estimation dataset in particular consists of 16000 training images and 5000 test images, comprising 90000-labelled objects spanning across twenty different classes including car pedestrian, bike etc. All images are color and saved as jpg.

III. Single Shot Multibox Detector

SSD [11] (shown in Figure 1) predicts object bounding boxes by predicting the offsets and class scores relative to a set of predefined default boxes of different scales1 and aspect ratios (as shown in Figure 2). The scale of the default box varies across multiple convolution layers of the SSD network. When the feature layer used is the conv4_3 (first one to have a classifier layer in Figure 1), the sizes of the default boxes, and hence the predicted objects, will be smaller.

This is also demonstrated in Figure 2, where it can be seen that the 88 feature map can (only) predict smaller objects cf.



loc : $\Delta(cx, cy, w, h)$
conf : $(c_1, c_2, \cdots, c_p)$
(a) Image with GT boxes
(b) 8×8 feature map
(c) 4×4 feature map

Fig. 2: Detection at multiple scales and aspect ratios using feature maps of different sizes and default boxes of different aspect ratios [11]

the 44 feature map. Thus, default boxes in the earlier layers have smaller scales than the ones further down in the network. Another related aspect is that the earlier layers will have larger number of default boxes; this can also be seen in 2, where the number of default boxes in 88 layer will be 82/42 = 4 times more than that in the 44 layer, with other parameters same for both. Hence changing the distribution of the scales of default boxes changes the size of the objects, that we are effectively making SSD find, as well as the number of potential locations for the objects. While the default configuration of SSD default boxes performs reasonably well out of the box, it should be adjusted according to the specific statistics of the current task, i.e. as an extreme example, if the objects that we are interested in always appear in large sizes, then having small default boxes is not only inefficient but might introduce unwanted false positives as well.

IV. Multistage Improvement

In this section, we present the improvements made to SSD. More specifically in section IV-A, IV-B and IV-C we present our approaches for improving Performance, Memory and time efficiency. Please note that each of these improvements could be combined in multiple ways and need not be always used together.

A. Improving Performance Efficiency by Scale Selection

As the first contribution, we focus on improving performance efficiency by improving scale selection. In an SSD network with L convolutional layers contributing to detection, let

$$S = S_1, ...S_L \tag{1}$$

denote the scale configuration of default boxes with $S_1 < S2 < ... < S_L$. By assuming that the scale of default boxes increase linearly from one layer to next, scale configuration S can be

$$S = \left\{ s_i | s_i = s_{min} + \frac{i-1}{L-1}(S_{max} - S_{min})1, ..., L \right\} \tag{2}$$

thus reducing the number of parameters to be optimized to two. Given a training set of images I annotated with object bounding boxes, the optimal scale configuration potentially depends on the weighted combination of two factors, and could be found by solving the following optimization problem.

$$S^* = \text{argmax}_S \left\{ \alpha \underset{\theta}{\text{Recall}}(G_I, D_S) + \beta Precision(S)) \right\} \tag{3}$$

where G_I is the set of all ground-truth bounding boxes in I, D_S is the set of anchor boxes from all L layers of the network with scale configuration S. $Recall_\theta(G_I, D_S)$, is defined as the fraction of boxes in G_I with IoU overlap greater than threshold θ with at least one of the boxes in D_S. α and β are parameters used for the trade-off between precision and recall. $Precision(S)$ is the average-precision performance of the network trained with the default anchor box configuration S on a validation set I. We vary the values of S_{min}, S_{max} in range of [0.1 0.8] respectively.

With a representative validation set, we would expect only the $Precision(S)$ to be important, akin to cross-validation, but doing that is costly as it requires separate training of the network for each element of S. Here, we thus use the $Recall_{\theta}(G_I, D_S)$ to do an aggressive pruning of the scale ranges and then finally select only a small number of scale ranges to do the validation experiments requiring full re-training of the network. The first stage thus requires calculating the different default boxes generated as a result of the different scale ranges, which is substantially cheaper in computation

B. Compression using Multilayer Pruning

As a second contribution, we focus on improving memory efficiency of SSD using compression with limited compromise on accuracy. The core of our compression framework is based on the principle: "Evaluate the importance of the filters and remove the unimportant ones" in line with previous works [28].

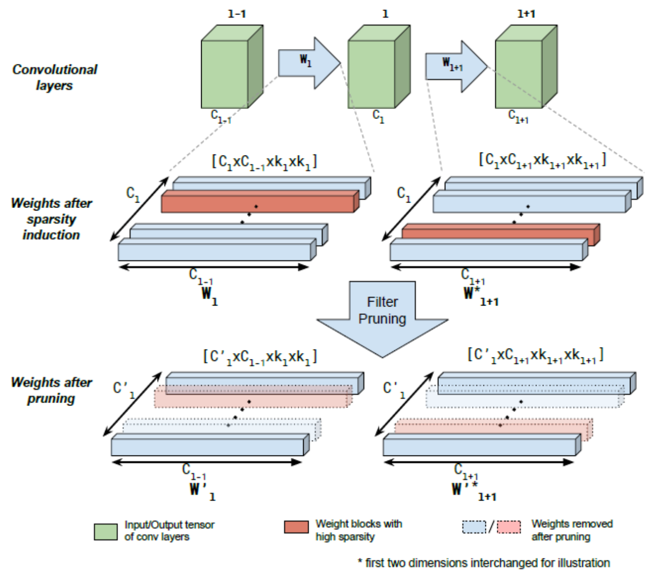
Let I_l and W_l denote the input tensor and parameters of l-th convolutional layer. Here $I_l \in \mathbb{R}^{c_{l-1} \times h_l \times w_l}$ has c_{l-1} channels, h_l rows and w_l columns. The weight tensor $W_l \in \mathbb{R}^{c_l \times c_{l-1} \times k_l \times k_l}$ is a set of c_l filters of $c_{l-1} \times k_l \times k_l$ size each. This convolutional layer produces the output tensor $O_l \in \mathbb{R}^{c_l \times h_{l+1} \times w_{l+1}}$, which is a set of c_l feature maps.

Our goal is to remove the unimportant filters in W_l. Let $c'_l \times c_{l-1} \times k_l \times k_l$ be the new size of W_l, where c'_l is the number of remaining filters after pruning of the unimportant ones. Since the number of filters is modified in layer l, the size of output tensor O_l and equivalently the size of input tensor of next layer I_{l+1} also reduces from $c_l \times h_{l+1} \times w_{l+1}$ to $c'_l \times h_{l+1} \times w_{l+1}$. Hence, the corresponding weights in W_{l+1} also need to be removed, which would in turn reduce the size of W_{l+1} from $c_{l+1} \times c_l \times k_{l+1} \times k_{l+1}$ to $c_{l+1} \times c'_l \times k_{l+1} \times k_{l+1}$.

In our multilayer pruning framework, we prune architecture such as SSD in multiple phases. In each phase, we target a different set of consecutive layers. First, we introduce sparsity in these layers and prune them aggressively in a single shot. Then, we recover the performance of the model by fine-tuning it. Unlike previous approaches [30] where pruning is done iterative on full network or layerwise, our method prunes a set of consecutive convolution layers at a time.

Figure 3 shows a schematic of our filter pruning strategy. Given a pre-trained model with parameters Θ and a set of consecutive layers L to be pruned. The entire framework executes in the following steps.

1) **Sparsity induction:** In the first step, trained the SSD model with loss-function with L1-norm resulting in the sparse model Θ_{L1}. Next, we set all the weights in L with an absolute value smaller than a threshold to zero resulting in the model Θ_{L1}^{th}. The value of the threshold is chosen based on the performance of Θ_{L1}^{th} on the validation set. Unlike previous approaches [**?**] that select layer-wise thresholds, we use a single global threshold for the whole layer set L. This allows us to prune filters in L in a single step. We explain this in detail in Section IV-B1.
2) **Filter selection:** To evaluate the importance of a filter in layer $l \in L$, we use filter sparsity statistics of layer l and $l+1$ in model Θ_{L1}^{th}. Unlike previous approaches that use data-driven methods to figure out the importance, we use only weight statistics. The key idea here is to remove all the filters in l that have a large fraction of zero weights in them as well as those filters in l that have a large fraction of zero weights corresponding to them in the filters of the following layer $l+1$ as illustrated in Figure 3. At the end of this step, we obtain a list of filters in layer l that are deemed as removable from the model. We repeat step 2 for each layer in L. A more detailed explanation is presented in Section IV-B2.
3) **Pruning:** Filters selected in step 2 for layer l and corresponding weights in layer $l+1$ associated with the output of these filters are removed from model Θ_{L1}. This is repeated for each layer in L sequentially.
4) **Retraining:** Finally, we retrain the pruned network using the original loss (without L1 regularization) to recover the performance drop due to sparsity induction in step 1.



Convolutional layers
l-1
l
l+1
W_l
W_{l+1}
C_{l-1}
C_l
C_{l+1}
Weights after sparsity induction
$[C_l x C_{l-1} x k_l x k_l]$
$[C_l x C_{l+1} x k_{l+1} x k_{l+1}]$
C_l
C_{l-1}
C_{l+1}
W_l
W^*_{l+1}
Filter Pruning
Weights after pruning
$[C'_l x C_{l-1} x k_l x k_l]$
$[C'_l x C_{l+1} x k_{l+1} x k_{l+1}]$
C'_l
W'_l
W'^*_{l+1}
Input/Output tensor of conv layers
Weight blocks with high sparsity
Weights removed after pruning
* first two dimensions interchanged for illustration

Fig. 3: Illustration of filter selection and pruning of convolutional filters in our pruning framework. After introducing sparsity in network weights by training with L1 regularization and thresholding, we examine the filter weights W_l and W_{l+1} of two consecutive convolutional layers l and $l+1$ respectively. Based on sparsity in these weights, we select the filters in W_l to be pruned. Note that corresponding weights in W_{l+1} also get removed. Removing these weights reduces the number of output channels in l-th layer from c_l to c'_l.

1) Sparsity Induction: Let $D = \{(x_0, y_0), (x_1, y_2), \ldots, (x_n, y_n)\}$ be the training set, where

x_i and y_i are input and target label. The parameters Θ of the original model are optimized to minimize the cost $C_D(\Theta)$.

$$\Theta = \text{argmin}\{C_D(\Theta)\} \tag{4}$$

The form of this cost function depends on the task to be solved by the original network. For instance, we use multibox loss function for SSD.

Let $L = \{l_{\text{start}}, \dots, l_{\text{end}}\}$ be the set of consecutive layers to be pruned. In order to induce sparsity in parameters of layers in L, we add L1-norm of parameters of these layers to the original cost function and train the network initialised with Θ.

$$\Theta_{L1} = \text{argmin}\left\{C_D(\Theta) + \alpha \sum_{l=l_{\text{start}}}^{l_{\text{end}}} ||W_l||_1\right\} \tag{5}$$

We choose the regularisation factor α such that the performance of the model with new parameters on a validation set D_{val} is close to the original performance, i.e. $P_{D_{\text{val}}}(\Theta_{L1}) \geq P_{D_{\text{val}}}(\Theta) - \epsilon_1$. Here, ϵ_1 is the tolerance constant that allows us to control the degree of sparsity in the parameters of layers in L. $P_{D_{\text{val}}}(\Theta)$ is the performance (eg. accuracy, AP etc.) of model with parameters Θ on the validation set D_{val}.

After obtaining Θ_{L1}, we set all the weights in L with absolute values smaller than a threshold t to zero. This gives us the parameters Θ_{L1}^{th}. We search for optimal t in a range proportional to standard deviation of weights in L, such that $P_{D_{\text{val}}}(\Theta_{L1}^{th}) \geq P_{D_{\text{val}}}(\Theta_{L1}) - \epsilon_2$. The constant ϵ_2 provides us the additional control on the number of zero weights in L. At this point, our network parameters are ready for filter pruning in layers in L, which is described in next section.

*2) **Filter pruning**:* In this step, we determine filter importance corresponding to layer $l \in L$. This is executed by examining sparsity statistics of W_l and W_{l+1} in Θ_{L1}^{th}. Let $F_i \in \mathbb{R}^{c_{l-1} \times k_l \times k_l}$ be the i-th of c_l filters in W_l. Note that in the next layer, the output feature map of filter F_i is connected to a slice of tensor W_{l+1} of size $c_{l+1} \times 1 \times k_{l+1} \times k_{l+1}$, we call this slice of weights G_i. Essentially, our filter pruning strategy is to remove the i-th filter if either one of the tensors F_i and G_i has a very large fraction of zero weights (see Figure 3). We implement this strategy using three thresholds s_F, s'_F and s_G with $s_F > s'_F$. For each $i \in \{1, \dots, c_l\}$, we evaluate following conditions and remove the i-th if atleast one of them holds true:

1) sparsity statistics in F_i is higher than s_F
2) sparsity statistics in F_i is higher than s'_F and sparsity statistics in G_i is higher than s_G.

Intuitively, the first condition selects the filters that have a very high level of sparsity ($> s_F$). These filters can be safely removed because their output activation maps are very weak. The second condition further selects those filters whose output activations are stronger than those selected by the first condition ($s_F >$ sparsity level $> s'_F$) but have an overall low contribution because of high sparsity in the weights connected to them in the next layer (sparsity level $> s_G$). Our method for computing sparsity statistics is unlike previous works [**?**] where the method is data-driven and measures sparsity on layers activation (e.g. output of ReLU) rather than its filter weights. To compute the sparsity level, we use the concept of zero row. A row in a filter $k_l \times k_l$ is considered as a zero row (size $1 \times k_l$) if all elements in the row are zero. Sparsity level is simply the percentage of zero rows in a filter.

$$\text{Sparsity level}(F_i) = \frac{\text{Number of zero rows}}{c_{l-1} \times k_l}$$

$$\text{Sparsity level}(G_i) = \frac{\text{Number of zero rows}}{c_{l+1} \times k_{l+1}}$$

Let Θ_c be the parameters of the network after filter pruning. We empirically found that taking the values of parameters in Θ_c from Θ_{L1} rather than from Θ_{L1}^{th} results in improved performance. The reason for improved performance is due to the restoration of values of zero weights that did not get pruned. Θ_{L1}^{th} only serve as a guide for the selection of filters to be pruned. Algorithm 1 describes the full procedure of filter pruning in detail.

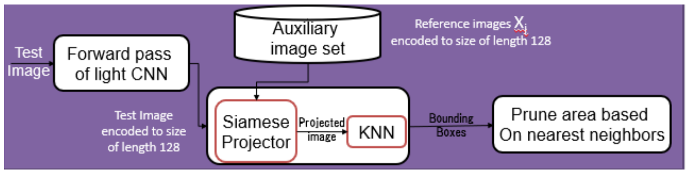
Finally, we retrain the pruned network without L1 regularization to restore the performance drop due to sparsity induction.

Algorithm 1 Algorithm for filter pruning

```
0: Inputs: Parameters of network after sparsity induction, $\Theta_{L1}$
   and $\Theta_{L1}^{th}$; set of layers to be pruned $L$.
0: Output: : New compressed model $M_c$ with weights $\Theta_c$.
0: filterindex = empty-list();
0: $\Theta_c$ = $\Theta_{L1}$
0: for layer $l$ in $L$ do
0:   for filter $i$ in $\{1, \dots, c_l\}$ do
0:     Extract $F_i$ and $G_i$ from $\Theta_L^{th}$
0:     splevelF = Sparsity-level($F_i$);
0:     splevelG = Sparsity-level($G_i$);
0:     if splevelF $>=$ $s'_F$ then
0:       if splevelF $>=$ $s_F$ then
0:         filterindex.add($i$)
0:       else
0:         if splevelG $>=$ $s_G$ then
0:           filterindex.add($i$)
0:         end if
0:       end if
0:     end if
0:   end for
0:   for filter $i$ in filterindex do
0:     $\Theta_c$ = Remove $F_i$ from $W_l$, $G_i$ from $W_{l+1}$ in $\Theta_c$
0:     $\Theta_L^{th}$ = Remove $F_i$ from $W_l$, $G_i$ from $W_{l+1}$ in $\Theta_L^{th}$
0:   end for
0: end for
0: $M_c$ = Redefine model with the remaining parameters $\Theta_c$
0: Initialize $M_c$ with $\Theta_c$
0: Return Model $M_c$ and $\Theta_c$ =0
```

C. Improving Time Efficiency through Semantic Retrieval

As a final contribution, we aimed at optimizing the test time of a trained model by actively pruning the search area. The



Test Image
Forward pass of light CNN
Auxiliary image set
Reference images X_j encoded to size of length 128
Test Image encoded to size of length 128
Siamese Projector
Projected image
KNN
Bounding Boxes
Prune area based On nearest neighbors

Fig. 4: Semantic Retrieval using Siamese Neural Network and Nearest Neighbor [27]

pruning stategy was based on retrieving images with similar semantic appearances and using their bounding boxes to prune the search area in the current image. To do this effectively, we proposed to use a small and fast CNN network to encode a set of dataset images offline. At test time, two operations need to be done, (i) the test image has to be encoded with the same fast CNN and (ii) the nearest neighbors are then retrieved in that embedding space. The bounding boxes from the retrieved images are then overlayed on the current image and are used to prune the search area by running detection only on the covering rectangle, and not the entire image. By doing so, we hoped to decrease the run time of the detection, while possibly trading off some performance.

However, we found that a generic similarity (based on an ImageNet pre-trained CNN) does not work effectively as it is overly biased by non-relevant artifacts in the image. In particular with VGG Fast CNN3 [27], pre-trained on the ImageNet dataset and compared using cosine similarity between l_2 normalized last fc activations.

To avoid such interference and obtain retrievals which reflect the semantics based on likely objects locations in the images, we then proposed to train Siamese network to learn the embedding. These embeddings would then be used instead of the embeddings obtained from ImageNet pretrained CNN. This can be seen as learning conditional priors, on object locations, from the training data. For each test image, similar images retrieved from the training data based on such embeddings would be expected to have similar object locations. The embeddings for the training images are computed offline and stored and only needs to be calculated for the test image at run time. The top retrieved images are then used to infer the possible object locations in the current image.

To train the Siamese network, we use a pairwise loss formulation similar to [29]. The method requires two sets of pairs of images, one where the pair (x_j, x_j) are similar, and have corresponding y_{ij} = +1, and the other where the pairs are dissimilar with y_{ij}= 1. The similarity and dissimilarity in our case is defined by the bounding boxes (of actual objects, of a certain class, present in the image) overlap the images with high bounding box overlap are considered similar, while those for which bounding boxes do not overlap sufficiently are dissimilar. As shown in figure 4 when a new test image comes, we project its feature using siamese network and do Nearest Neighbor (NNs) with the training images. We retrieve k NNs, using Eclidean distance after projection, and use their ground truth bounding boxes to have a pruned search area as the minimum enclosing rectangle for all those bounding boxes, with an extra padding of 10 pixels for some context.

TABLE IV: Comparison of Performances across experiments. The value signifies mAP measure

	PASCAL VOC	KITTI	GTSDB	STSDB	BTSDB	RDDB
SSD	79.8	53.9	76.8	71.34	85.5	48.16
SCR	77.5	68.5	82.44	88.4	91.6	49.13
CC	77.94	65.2	89.5	64.2	80.4	37.2
CC_SCR	77.94	65.2	89.5	63.7	81.4	38.4

TABLE V: Comparison of Speed across experiments. The results signifies speed in FPS measured using TitanX GPU using Pytorch Framework.

	PASCAL VOC	KITTI	GTSDB	STSDB	BTSDB	RDDB
SSD	19	19	19	19	19	19
SCR	22	22	22	22	22	22
CC	142	142	142	142	142	142
CC_SCR	140	140	140	140	140	140

D. Experiments and Results

In this section, we present results for each of the improvements made to SSD across all the previously mentioned benchmark datasets. Unless specified explicitly all the results report are for SSD512. More specifically in this section, we present results of accuracy in terms of mean average precision (mAP), memory consumption in mega bytes (MB) and speed in frames per second (FPS) tested on TitanX GPU in Tables IV,V and VI for SSD with **Scale Selection with Semantic Retrieval (SCR), Compression (CC), Compression with Scale Selection and Semantic Retrieval (CC_SCR)** respectively. Please note that the results in table V corresponds to 16 Nearest Neighbors fetched during test time, the corresponding mAP and total memory size are mentioned in table IV and VI. As such there is 1-1 correspondence between the results of tables IV,V and VI.

E. Discussion on Results

In previous sections we described the improvements developed for SSD to improve its performance and speed at the same time reduce memory consumption which is visible from Tables IV,V and VI. In this section, we will describe results and some ablation study done on the said work. Considering the results in Tables IV,V and VI we present analysis and ablation study considering GTSDB dataset, which showed improvement in all aspects of performance, speed and memory.

*1) **Analysis of Performance Improvement**:* Figure 5 shows the histogram of scale of the bounding boxes in GTSDB

TABLE VI: Comparison of Model size across experiments.The value signifies size of model in MB.

	PASCAL VOC	KITTI	GTSDB	STSDB	BTSDB	RDDB
SSD	108	108	108	108	108	108
SCR	113	113	113	113	113	113
CC	22	22	3.8	3.8	3.8	70
CC_SCR	20.7	20.7	8.8	8.8	8.8	78.8

TABLE VII: Performance and average scale of default boxes across various layers of SSD with different selected scales. * indicates scale selected by our scale selection approach.

Default Scale range of Boxes $\rightarrow$	**10-80**	**20-75**	**25-90**	**5-150***
Layer	**Scales of default boxes (in px)**			
conv4_3	33.0	41.2	22.0	10.2
fc7	65.7	77.9	59.6	41.2
conv6_2	104.5	114.6	97.2	85.2
conv7_2	140.2	151.2	134.7	129.2
conv8_2	176.0	187.9	172.3	173.2
pool16	247.5	261.2	247.5	261.2
Results (mAP)	78.5	76.8	80.2	82.44

dataset. We observe that even in a dataset with only three classes of objects, i.e. Mandatory, Prohibitory and Danger, the distribution of the object scales vary while there are many objects with scales in range of 20 to 80px, there are not many in the higher scales. Thus in practice, for optimal performance, we should aim for a distribution of default boxes which is tuned wrt. the actual distribution in out target set of images. Such distribution could be potentially tuned to the object class, at the expense of linear scaling the complexity (as it will require as many different models as there are classes).

Table VII shows the empirical comparison of these different scale settings. First, the table shows the scale of the the default boxes across layers. As we can see that it drops very quickly from the earlier layers to later, as evident from conv4_3 to conv7_2. The default box scale range parametrization in SSD controls the sizes of the default boxes used at each such layer (which has a classifier connection) thereby impacting the result. Also the table shows the results for each of the setting including the result obtained on scale selected by our approach. As we can see with broader scale selection of the default boxes, the results does improve signficantly. We see similar results across the datasets of KITTI, BTSDB and STSDB, where with scale selection the results improve drastically. However one would argue that from Table IV that the result for other datasets namely PASCAL VOC and RDDB are limited. Further in case of PASCAL VOC the results drop and in case of RDDB the results improvement is approx 1%. We conjecture this as an effect of the dataset where in case of the PASCAL VOC, the dataset of train and test are random images from flickr where the objects sizes vary significantly across the test and train set leading to varying scale distribution pattern that couldn't be taken advantage of by our method. Secondly in case of RDDB the results improvement is limited mostly because of the problem of class overlap among the damage types.

2) ***Analysis of reduction in Memory***: SSD consists of a large number of convolutional layers, where multiple convolutional layers contribute to the detections at various scales. The complex dependency of output detections on convolutional layers makes it challenging to compress. Previous works, only attempt to compress the base network of SSD. In our experiments, we compress the full network including the detection layers. However, we observed that compressing the entire network together in a single step massively degrades

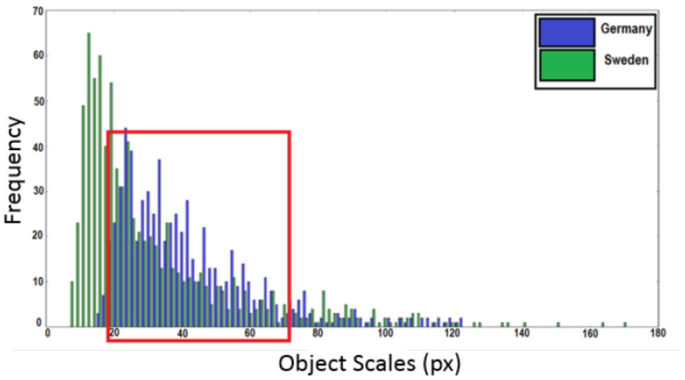


Fig. 5: Histogram of scale of the bounding boxes in TSDB dataset, for GTSDB (blue) and STSDB (green) classes. Scale is the geometric mean of the width and the height. [11]

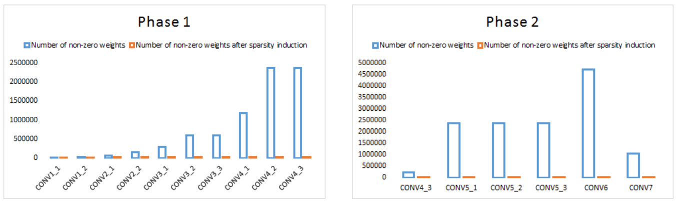


Fig. 6: Number of non-zero parameters in SSD convolutional layers before and after different phases of our sparsity induction method (GTSDB).

the performance. Therefore, we apply our multilayer pruning strategy and target the network in chunks of layers in multiple phases.

Settings: In all of our experiments, we use SSD with base network VGG16. For pruning of SSD, we employ our method in 4 phases where were select set of convolutions namely: conv1_1 - conv4_3, conv4_3 - conv7, conv7 - conv8_2 and conv8_2 - conv12_2. We also include the conv layers that perform the localization and classification in these groups of layers. Regularization constant α is chosen in such a way that performance loss $\epsilon_1 \in [2,3]$. Experimentally, we found that if $\epsilon_1 \geq 5$ then at a later stage we can not recover the performance loss due to the pruning of the important connections. Recall that Θ_{L1} and Θ_{L1}^{th} denote the model parameters obtained after applying regularization and global thresholding (with threshold t) respectively. The threshold t is selected such that there is a further performance drop of $\epsilon_2 \in [5,7]$ after thresholding. Note that Θ_{L1}^{th} is only used for selecting unimportant filters while the actual filter pruning is done on the model with parameters Θ_{L1}. This allows us to afford a relatively high-performance drop after thresholding and explains the high value of ϵ_2. Experimentally, we found that $\epsilon_2 \geq 10$ results in too much sparsity in Θ_{L1}^{th} and consequently affects the filter selection procedure by selecting important filters for pruning. This results in an irrecoverable loss in performance.

We use grid search for selecting hyperparameters s_F, s'_F and s_G in the range $[0.70, 0.99]$ and found $s_F = 0.9$, $s'_F = 0.85$ and $s_G = 0.95$ to work best for ϵ_1 and ϵ_2 in the ranges

	CONV Layers																								
SSD 512	**1_1**	**1_2**	**2_1**	**2_2**	**3_1**	**3_2**	**3_3**	**4_1**	**4_2**	**4_3**	**5_1**	**5_2**	**5_3**	**6**	**7**	**8_1**	**8_2**	**9_1**	**9_2**	**10_1**	**10_2**	**11_1**	**11_2**	**12_1**	**12_2**
original	64	64	128	128	256	256	256	512	512	512	512	512	512	1024	1024	256	512	128	256	128	256	128	256	128	256
pruned-det1	64	64	121	123	206	84	113	59	50	189															

TABLE VIII: Layer-wise pruning results in SSD512 model on GTSDB dataset.

Model	**AP**				**Size**	**Total Parameters**
	prohibitory	**mandatory**	**danger**	**mAP**		
SSD300-original	38.61	31.07	47.2	38.96	96.1MB	24.0M
SD512-original	88.67	76.32	82.34	82.44	98.7 MB	24.7M
SSD512-pruned-1det	97.95	83.82	86.83	**89.53**	**3.8 MB (26X)**	**938.8K (3.8%)**

TABLE IX: AP for each class with original SSD and pruned SSD models on GTSDB.

TABLE X: Speedup achieved by Semantic Retrieval for GTSDB.

No of Nearest Neighbor, K	**2**	**4**	**8**	**16**	**No Pruning**
CNN forward pass time (ms)	0.93	0.93	0.93	0.93	
Average prior retrieval time (ms)	0.65	0.67	0.77	0.92	
Average SSD forward pass time (ms)	20.8	25.0	28.7	31.8	39.6
Total time (ms)	22.4	26.6	30.4	33.5	39.6
Speedup achieved	1.8x	1.5x	1.3x	1.2x	
Speedup expected (theoretical)	2.7x	2.1x	1.7x	1.5x	
mAP for GTSDB	70.3	71.4	80.2	81.4	82.44

mentioned above.

We initally trained both SSD300 and SSD512 on GTSDB and obtained 38.96% and 82.44% mAP on test set respectively. Due to lower results, in this paper we only used SSD512 for our compression experiments. Figure 6 shows the number of non-zero parameters in convolutional layers after sparsity induction in first two phases. We observed that most of the parameters in layers after conv4_3 become zero (see Figure 6). For instance, in conv7 only 3 out of 1M remain non-zero after thresholding. This is expected as only the early layers in SSD are responsible for detecting of smaller objects. We compress the layers in the first phase and remove all layers after conv4_3 because of the extremely high sparsity. The resulting model SSD512-pruned-1det which has only one detection layer is 26X smaller than the original model. Table VIII shows the layer-wise pruning results. Table IX shows performance and model size of original SSD and SSD512-pruned-1det.

However, in table VI we can see that while models of TSDB are compressed at higher rate, unlike the case of PASCAL VOC and KITTI data sets. We show one such example of compressing SSD over PASCAL VOC dataset in Table XI. This suggests that amount of compression of the model is indeed data dependent, which in turn means that our methods tries to model the max capacity of the neural network at that compressed size.

*3) **Analysis of Improvement in Speed:*** Table X shows the relative speed up achieved in case of GTSDB dataset. We show the time consumption with respect to figure 4. As we can see from table for V with addition of semantic retrieval we increase the speed by additional 3 FPS across all our experiments. Here we describe in the detail the results obtained for GTSDB dataset inline with results from previous section. As we can see from table X the number of nearest neighbors is proportional to the mAP and inversely related to additional speed gained by the detector. With more nearest neighbors we achieve the better results, at the same time it reduces the FPS owing to more time spent in retrieval of neighbors. However we can see that the result is not as expected through our theoretical calculation. We believe this mostly has to do with the implementation of the solution and with optimized implementation we expect to achieve similar results. While one would argue that compressed model alone would suffice for speed up as well as memory reduction. We believe the Semantic Retrieval based pruning would be useful in area where the higher performance of the system is critical, which cannot be obtained via compressed model.

F. Conclusion

In this paper, we have proposed a multistage improvement to SSD to improve its speed, memory efficienct and performance. To do this we introduces Scale selection, Semantic Retrieval and multilayer pruning. We present various results across multiple benchmarks namely PASCAL VOC, TSDB, RDDB and KITTI in which we found the improvements to be fruitful. Further we achieved a significant amount of compression and performance improvements. At the same time Semantic Retrieval Indeed helped us in gained the last ounce of speed left in the SSD. In the future, we would be interested in exploring this approach for other object detection models and architecture variants of SSD.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In CVPR, 2015.

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for largescale image recognition. In ICLR, 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In CVPR, 2016.

[5] Girshick, Ross B.. Fast R-CNN. 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1440-1448.

[6] Ren, Shaoqing, Kaiming He, Ross B. Girshick and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. CoRR abs/1506.01497 (2015): n. pag.

[7] Ren, Shaoqing, Kaiming He, Ross B. Girshick and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. CoRR abs/1506.01497 (2015): n. pag

[8] Dai, Jifeng, Yi Li, Kaiming He and Jian Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks. NIPS (2016).

[9] Liu, Wei, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu and Alexander C. Berg. SSD: Single Shot MultiBox Detector. ECCV (2016).

[10] Fu, Cheng-Yang, Wei Liu, Ananth Ranga, Ambrish Tyagi and Alexander C. Berg. DSSD : Deconvolutional Single Shot Detector. CoRR abs/1701.06659 (2017): n. pag.

		SSD300 on PASCAL VOC			SSD512 on PASCAL VOC		
		`original`	`pruned-3det`	`pruned-6det`	`original`	`pruned-4det`	`pruned-7det`
Input Size		300X300X3	300X300X3	300X300X3	512X512X3	512X512X3	512X512X3
Layers	**CONV1_1**	3X3 CONV, 64	3X3 CONV, 64	3X3 CONV, 64	3X3 CONV, 64	3X3 CONV, 64	3X3 CONV, 64
	CONV1_2	3X3 CONV, 64	3X3 CONV, 56	3X3 CONV, 56	3X3 CONV, 64	3X3 CONV, 61	3X3 CONV, 61
	CONV2_1	3X3 CONV, 128	3X3 CONV, 107	3X3 CONV, 107	3X3 CONV, 128	3X3 CONV, 119	3X3 CONV, 119
	CONV2_2	3X3 CONV, 128	3X3 CONV, 121	3X3 CONV, 121	3X3 CONV, 128	3X3 CONV, 122	3X3 CONV, 122
	CONV3_1	3X3 CONV, 256	3X3 CONV, 193	3X3 CONV, 193	3X3 CONV, 256	3X3 CONV, 215	3X3 CONV, 215
	CONV3_2	3X3 CONV, 256	3X3 CONV, 158	3X3 CONV, 158	3X3 CONV, 256	3X3 CONV, 160	3X3 CONV, 160
	CONV3_3	3X3 CONV, 256	3X3 CONV, 195	3X3 CONV, 195	3X3 CONV, 256	3X3 CONV, 187	3X3 CONV, 187
	CONV4_1	3X3 CONV, 512	3X3 CONV, 263	3X3 CONV, 263	3X3 CONV, 512	3X3 CONV, 252	3X3 CONV, 252
	CONV4_2	3X3 CONV, 512	3X3 CONV, 181	3X3 CONV, 181	3X3 CONV, 512	3X3 CONV, 172	3X3 CONV, 172
	CONV4_3 D1	3X3 CONV, 512	3X3 CONV, 331	3X3 CONV, 331	3X3 CONV, 512	3X3 CONV, 313	3X3 CONV, 313
	CONV5_1	3X3 CONV, 512	3X3 CONV, 98	3X3 CONV, 98	3X3 CONV, 512	3X3 CONV, 170	3X3 CONV, 170
	CONV5_2	3X3 CONV, 512	3X3 CONV, 108	3X3 CONV, 108	3X3 CONV, 512	3X3 CONV, 134	3X3 CONV, 134
	CONV5_3	3X3 CONV, 512	3X3 CONV, 78	3X3 CONV, 78	3X3 CONV, 512	3X3 CONV, 117	3X3 CONV, 117
	CONV6	3X3 CONV, 1024	3X3 CONV, 146	3X3 CONV, 146	3X3 CONV, 1024	3X3 CONV, 305	3X3 CONV, 305
	CONV7 D2	1X1 CONV, 1024	1X1 CONV, 106	1X1 CONV, 106	1X1 CONV, 1024	1X1 CONV, 332	1X1 CONV, 332
	CONV8_1	1X1 CONV, 256	1X1 CONV, 34	1X1 CONV, 34	1X1 CONV, 256	1X1 CONV, 122	1X1 CONV, 122
	CONV8_2 D3	3X3 CONV, 512	3X3 CONV, 198	3X3 CONV, 198	3X3 CONV, 512	3X3 CONV, 133	3X3 CONV, 133
	CONV9_1	1X1 CONV, 128		1X1 CONV, 12	1X1 CONV, 128	1X1 CONV, 89	1X1 CONV, 89
	CONV9_2 D4	3X3 CONV, 256		3X3 CONV, 36	3X3 CONV, 256	3X3 CONV, 168	3X3 CONV, 168
	CONV10_1	1X1 CONV, 128		1X1 CONV, 22	1X1 CONV, 128		1X1 CONV, 81
	CONV10_2 D5	3X3 CONV, 256		3X3 CONV, 41	3X3 CONV, 256		3X3 CONV, 92
	CONV11_1	1X1 CONV, 128		1X1 CONV, 22	1X1 CONV, 128		1X1 CONV, 40
	CONV11_2 D6	3X3 CONV, 256		3X3 CONV, 66	3X3 CONV, 256		3X3 CONV, 84
	CONV12_1				1X1 CONV, 128		1X1 CONV, 40
	CONV12_2 D7				4X4 CONV, 256		4X4 CONV, 80
Total Parameters		26.3M	3.7M (14.1%)	3.9M (14.8%)	27.2M	5.1M (18.8%)	5.5M (20.2%)
Model Size		105.2 MB	15 MB (7X)	15.7 MB (6.7X)	108.8 MB	20.3 MB (5.4X)	22 MB (4.9X)
Mean AP		77.16	72.15	75.07	79.52	75.59	77.94

TABLE XI: Layer-wise pruning results in SSD300 and SSD512 models on PASCAL VOC dataset. CONVX_DN denotes the N-th layer that is connected to detection layer (conf and loc layer).

[11] Redmon, Joseph and Ali Farhadi. YOLO9000: Better, Faster, Stronger. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017): 6517-6525.

[12] Lin, Tsung-Yi, Priya Goyal, Ross B. Girshick, Kaiming He and Piotr Dollr. Focal Loss for Dense Object Detection. 2017 IEEE International Conference on Computer Vision (ICCV) (2017): 2999-3007.

[13] He, Kaiming, Georgia Gkioxari, Piotr Dollr and Ross B. Girshick. Mask R-CNN. 2017 IEEE International Conference on Computer Vision (ICCV) (2017): 2980-2988.

[14] 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.

[15] Houben, Sebastian, Johannes Stallkamp, Jan Salmen, Marc Schlipsing and Christian Igel. "Detection of traffic signs in real-world images: The German traffic sign detection benchmark." The 2013 International Joint Conference on Neural Networks (IJCNN) (2013): 1-8.

[16] Timofte, Radu, Karel Zimmermann and Luc Van Gool. "Multi-view traffic sign detection, recognition, and 3D localisation." 2009 Workshop on Applications of Computer Vision (WACV) (2009): 1-8.

[17] Larsson, Fredrik and Michael Felsberg. "Using Fourier Descriptors and Spatial Models for Traffic Sign Recognition." SCIA (2011).

[18] Maeda, Hiroya, Yoshihide Sekimoto, Toshikazu Seto, Takehiro Kashiyama and Hiroshi Omata. Road Damage Detection Using Deep Neural Networks with Images Captured Through a Smartphone. CoRR abs/1801.09454 (2018).

[19] M. Everingham, S. M. A. Eslami, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. International Journal of Computer Vision, 111:98136, 2014.

[20] Andrea Vedaldi, Varun Gulshan, Manik Varma, and Andrew Zisserman. Multiple kernels for object detection. In CVPR, 2009.

[21] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In CVPR, 2005.

[22] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9):16271645, 2010.

[23] Xie, X., Han, X., Liao, Q., & Shi, G. (2017). Visualization and Pruning of SSD with the base network VGG16. ICDLT '17.

[24] Chen, G., Choi, W., Yu, X., Han, T.X., Chandraker, M.K. Learning Efficient Object Detection Models with Knowledge Distillation. NIPS 2017

[25] Anisimov, D., Khanova, T. (2017). Towards lightweight convolutional neural networks for object detection. 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), 1-8.

[26] Manikandan R, Traffic Sign Detection - How well does SSD fare up? An Empirical Study, 47th Annual IEEE AIPR, Ubiquitous Imaging ,Washington, D.C., USA, 2018

[27] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In British Machine Vision Conference (BMVC), 2014.

[28] Luo, J., Wu, J., Lin, W. (2017). ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression. 2017 IEEE International Conference on Computer Vision (ICCV), 5068-5076.

[29] Binod Bhattarai, Gaurav Sharma, and Frederic Jurie. CP-mtML: Coupled projection multi-task metric learning for large scale face retrieval. In CVPR, 2016.

[30] Han, S., Mao, H., Dally, W.J. (2015). Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding. CoRR, abs/1510.00149.