UAV-Net: A Fast Aerial Vehicle Detector for Mobile Platforms

Tobias Ringwald[†] Lars Sommer[‡*] Arne Schumann[*] Jürgen Beyerer[*‡] Rainer Stiefelhagen[†]

[†]CV:HCI
Karlsruhe Institute of Technology
Karlsruhe, Germany

[‡]Vision and Fusion Lab
Karlsruhe Institute of Technology
Karlsruhe, Germany

[*]Fraunhofer IOSB
Fraunhoferstraße 1
Karlsruhe, Germany

{firstname.lastname}@{kit.edu, iosb.fraunhofer.de}

Abstract

Vehicle detection in aerial imagery is a challenging task due to small object sizes, high object density and partial occlusions. While past research mostly focused on improving detection accuracy, inference speed is another important factor when using CNN object detectors in a real life scenario – especially when targeting mobile platforms like unmanned aerial vehicles (UAVs). In this work, we compare several established detection frameworks in terms of their accuracy-speed trade-off and show that the Single Shot MultiBox Detector (SSD) offers the best compromise. We subsequently undertake a thorough evaluation of several design choices to further increase detection speed while sacrificing little to no accuracy. This includes the choice of base network architecture, improved prediction layers and an automatic model pruning approach. Given our evaluation results, we finally construct UAV-Net – a novel aerial vehicle detector that has a model size of less than 0.4 MiB and is more than 16 times faster than current top performing approaches. UAV-Net is well suited for on-board processing and operates in real time on a Jetson TX2 platform. Nevertheless, its accuracy is on par with state-of-the-art approaches on the DLR 3K, VEDAI and UAVDT datasets. Code and models are available on the project website.[1]

1. Introduction

In recent years, vehicle detection in aerial images received significant attention, as it is crucial for many applications in the civil and military domain, *e.g.* traffic monitoring, surveillance, disaster relief, as well as search and rescue tasks. However, the relatively small object sizes, varying vehicle types, partial occlusions, and intricate backgrounds impede the detection task (see Figure 1).

Recent approaches [1, 2, 3, 4, 5, 6, 7, 8] for vehicle detection in aerial imagery are based on deep learning detection frameworks such as Faster R-CNN [9] and SSD [10]. Although inference time plays an important role when using deep learning based object detectors in a real life scenario on mobile platforms like UAVs, past research on vehicle detection in aerial imagery mostly focused on improving detection accuracy [4, 5, 6, 7, 8]. Thus, most existing approaches are limited to offline processing or deployment on powerful hardware in a ground control station.

Figure 1: Example image from the DLR 3K dataset [11]. Object size and density are vastly different from commonly used detection datasets.

Our target scenario in this work is on-board processing of imagery during the flight time of a UAV. In such an aerial setting, increased inference speed does not only relate to the frame rate at which a video can be processed but also to the area of ground that can be analyzed in a fixed time frame. This may allow aerial platforms to fly faster or at higher altitudes.

In the wider image classification literature, novel computation-efficient CNN architectures, such as MobileNet [12], ShuffleNet [13], or SqueezeNet [14], have recently been proposed for use on mobile platforms with limited resources. Adopting these CNN architectures as base

[1] https://gitlab.com/tringwald/uav-net

networks for object detection frameworks leads to high detection accuracy while inference time and memory footprint are considerably reduced [12, 15].

In this work, we propose a computation-efficient and lightweight vehicle detector called UAV-Net, which is based on SSD and adapted to the unique characteristics of aerial imagery. For this purpose, we systematically assess modifications to the key stages of the detector with regard to their influence on inference time and detection accuracy. The impact of multiple computation-efficient CNN architectures and their variants are analyzed as base networks for the vehicle detection task. We further propose a novel filter pruning approach that automatically condenses the trained networks in an iterative manner. Furthermore, the impact of employed feature maps, activation functions, and filters used for regression and classification stages is evaluated. Based on our findings, we finally construct UAV-Net, which has a model size of less than 0.4 MiB and is more than 16 times faster than current top performing approaches. UAV-Net is well suited for on-board processing and operates in real time on a Jetson TX2 platform, while its accuracy is on par with the state-of-the-art approaches on the DLR 3K [11], VEDAI [16] and UAVDT [17] datasets.

In summary, our contributions are threefold: i) We evaluate and compare several important design choices to optimize the accuracy-speed trade-off for aerial vehicle detection, ii) we propose a novel filter pruning approach, and iii) we compose a new, fast, yet accurate vehicle detector for aerial imagery based on the most promising design choices and provide it to the community to inspire further research.

2. Related Work

In literature, a large variety of deep learning based detection frameworks has been proposed, which can be categorized into two-stage and one-stage approaches. Two-stage approaches like Faster R-CNN [9] and R-FCN [18] typically comprise an initial stage to generate region proposals, which are then classified in the subsequent stage. One-stage approaches like SSD [10] and YOLOv2 [19] predict object classes and locations in a single step, which generally results in improved inference speed compared to two-stage approaches. While multiple modifications, *e.g*. multi-layer exploitation [10, 20] and top-down-networks [21, 20, 22] have been proposed to improve the detection accuracy, less research focused on lightweight object detection methods suited for mobile platforms.

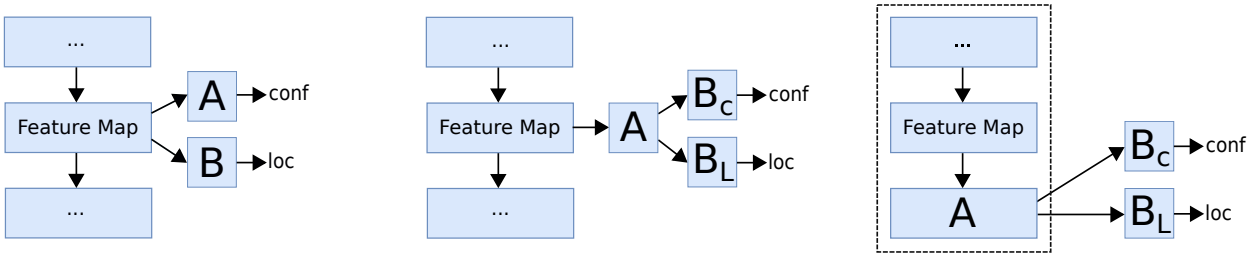
Several authors proposed to employ lightweight architectures as base networks for different object detection frameworks. Howard *et al*. [12] proposed MobileNet, which comprises depthwise separable convolutions, as base network for SSD. MobileNet-SSD achieves a comparable detection accuracy on the MS COCO dataset with a significantly reduced parameter and FLOP count. SqueezeDet [15], which is inspired by the YOLO detection pipeline, employs SqueezeNet [14] as base network. State-of-the-art detection accuracy is achieved on the KITTI object detection dataset while the inference speed is clearly reduced. Similar to SqueezeDet, Tiny SSD [23] comprises a base network that leverages the fire modules introduced in [14]. Wang *et al*. [24] outperform MobileNet-SSD on PASCAL VOC and MS COCO by employing PeleeNet [24] as base network for SSD. Recently, Li *et al*. [25] proposed Tiny-DSOD, which is based on SSD [10] and DSOD [26]. The base network is replaced by a depthwise dense block based architecture. In contrast to the other lightweight detectors, a depthwise feature pyramid network is used to add more semantic information. Instead of replacing the base network, Li *et al*. [27] proposed a light-head R-CNN to speed up two-stage approaches. For this, the number of feature map channels is reduced and the classification head is replaced by a single fully-connected layer.

Deep learning based detection frameworks have been applied to various domains, including vehicle detection in aerial images. Several modifications have been proposed to account for the characteristics of aerial imagery. For instance, Sakla *et al*. [1] removed the last two sequences of convolutional layers in order to increase the feature map resolution for Faster R-CNN. In [2], the authors additionally adapted the size of anchor boxes to the size of present vehicles. In [3], the authors analyze adaptations of Faster R-CNN, R-FCN and SSD. Following the general literature, several modifications, such as top-down networks [4, 5, 6] and multi-layer exploitation [7, 8], have been adopted for vehicle detection in aerial imagery. But very few works focus on improving the inference speed or applying lightweight networks. The authors of [28] propose ShuffleDet, which employs ShuffleNet as base network for a modified variant of SSD and apply it for car detection in parking lots. Experimental results show a good trade-off between accuracy and speed on mobile platforms.

3. Fundamentals

3.1. Single Shot MultiBox Detector (SSD)

SSD is often used as base detection framework due to its good trade-off between detection accuracy and inference speed. SSD is a fully convolutional network that can use any CNN base network as a feature extractor. The outputs of multiple convolutional layers with different dimensions are used as feature maps to predict detections at multiple scales. For this purpose, a set of default boxes used for bounding box regression is associated with each feature map location. The positions of the default boxes are fixed relative to the corresponding feature map location. $(C{+}4)K$ convolutional filters with kernel size $3{\times}3$ are applied at each feature map location to predict four bounding box offsets



(a) Normal SSD setup with separate convolution blocks A and B.

(b) SSD setup with shared convolution block A and separate blocks B_C and B_L.

(c) For a single feature map, block A can also be seen as an extension to the base network.

Figure 2: Different setups for the auxiliary classification and bounding box regression filters.

relative to the default boxes and C confidence scores, where C is the number of object classes and K is the number of default boxes. The predictions from all feature maps are then combined. Finally, redundant predictions are filtered out via non-maximum suppression (NMS).

3.2. Mobile Networks

With the upcoming need for inference on mobile platforms, earlier network architectures like VGG-16 [29] are not suitable anymore due to the lack of computational power. Therefore, research extended to the area of mobile networks, trying to find a good trade-off between computational requirements and accurate output. Most modern CNNs for mobile platforms share common design guidelines like reducing the channel depth or avoiding larger convolution kernels wherever possible.

MobileNet [12] uses depthwise separable convolution (DSC) as its main building block instead of 3×3 convolutions. The DSC block is comprised of a 3×3 depthwise convolution followed by a 1×1 pointwise convolution. For non-trivial image dimensions and channel depths, a DSC block is often an order of magnitude more efficient than normal 3×3 convolutions while only requiring a fraction of their parameters.

ShuffleNet [13] further refines this approach by also replacing the 1×1 convolutions with cheaper 1×1 group convolutions. Depending on the group count g, a group convolution filter only considers the $\frac{C}{g}$ channels of its respective group instead of the full channel depth C. This further lowers the parameter count and computational complexity of the group convolution operation and makes ShuffleNet one of the smallest networks evaluated in this paper.

While DSCs and group convolutions are more efficient in theory, framework implementations are often unoptimized or not even present. *SqueezeNet* [14] does not make use of these special convolution types and instead leverages the fire module building block that only consists of standard convolutions. The "squeeze" part of a fire module reduces the number of channels for the mixed 1×1 and 3×3 "expand" convolutions by compressing the channel depth with cheaper 1×1 convolutions first.

ZynqNet [30] further improves the SqueezeNet architecture by replacing pooling with strided convolution, different downsampling steps and the alternating use of 3×3 and 1×1 kernels for the "squeeze" layer.

4. Optimization Strategy

4.1. Adaptation to Aerial Imagery

The employed feature maps and default box settings are essential design parameters for tuning the detection accuracy [1, 2]. High feature map resolutions are necessary to precisely locate small object instances, especially for vehicles in aerial imagery [3]. Thus, we only use the output of the last layer with an approximate downsampling factor of 8 as feature map, *e.g.* `conv4_3` in case of VGG-16. Deeper layers are not considered due to the low spatial resolution of feature maps. Note that renouncing multi-layer exploitation is only suited in case of a constant ground sampling distance which results in negligible variation in object sizes. In case of small object instances, the best detection accuracy is achieved for default boxes in the range of the object sizes. To provide suitable default box sizes, we apply the clustering approach from [21] to the training data. Similarly, $max(\frac{H}{W}, \frac{W}{H})$ is used as the clustering feature for the aspect ratio and $\sqrt{W \times H}$ for the default box size. This method yields aspect ratios $\{1.3, 1.9\}$ and box sizes $\{28, 35\}$ px for the DLR 3K dataset. As vehicles can be arbitrarily rotated, the respective reciprocal ratios are also added.

4.2. Auxiliary SSD Layers

In the SSD framework, classification and bounding box regression filters (auxiliary or prediction layers) are applied to different layers of the base network to predict detections (see Figure 2a). For this step, standard 3×3 convolutions are usually used for blocks A and B. Motivated by the modifications proposed in [24] and [31], different building blocks are evaluated as replacements for the expensive 3×3 convolutions. Additionally, the setup in Figure 2b is used to share an intermediate stage for both subsequent classification and regression filters. This can then be used to split a DSC into a pure depthwise part (block A) and a pure point-

wise part in block B_C and B_L. Note that for a SSD setup with a single feature map, Figure 2b could also be seen as an extension to the base network as shown by the dashed line in Figure 2c, whereas the separate setup can be seen as Figure 2c with an identity transformation as block A.

4.3. Filter Pruning

Nowadays, base networks like VGG or MobileNet are commonly pretrained on ImageNet classification with 1000 classes. Thus, these networks often have a large number of parameters which is unnecessary for detection tasks with only a few classes. We therefore propose an automatic pruning approach to iteratively prune an existing network by removing filter kernels. Our approach is based on the one-shot pruning method by Li *et al*. [32], which prunes a convolutional layer by calculating the ℓ_1 norm for every filter and then removing the filters with the lowest value until only a percentage δ of filters remain. However, this requires to either use a fixed percentage δ for all layers or find an δ_n for every layer $l_n \in L$, where L is the set of all convolutional layers in a network. Using a fixed δ leads to inferior results, as certain layers are more sensitive to pruning than others, e.g. convolutional layers close to the input as shown in [32]. A layerwise δ_n involves the retraining of many different network architectures in order to find an optimal combination. This requires an excessive amount of resources and can take multiple GPU-months – especially when training on large aerial images. We therefore propose an automatic pruning algorithm that requires little to none human intervention and few resources.

Our automatic pruning approach only requires the training of a fully-sized architecture followed by a single finetuning step. The pruning process starts by considering every convolutional layer $l_n \in L$ of the already trained network. Similar to [32], the ℓ_1 norm for every filter f_i in layer l_n is calculated as $\sum_{w \in f_i} |w|$ where $w \in f_i$ denotes all weights in a filter f_i. We then remove a fixed amount of k filters with the lowest ℓ_1 norm from l_n and also the corresponding k channels of the convolutional filters in the next layer. The network is evaluated on a small validation set in order to judge the impact of the removed filters. For our detection task, we monitor the sensitivity metric $S = \frac{TP}{P}$, as this best describes the network's abilities to detect objects while being more stable than average precision. This process is repeated for every layer l_n, yielding tuples of the form (S, l_n, k). The best tuple with regard to S is then chosen (prefering later layers in case of a draw) and used to create a new network with k fewer filters in layer l_n for the next pruning iteration. This process can be seen as a greedy tree search and is visualized in Figure 3. The algorithm terminates when only a predetermined percentage φ of filters remains in the network. The final network is then finetuned to recover the lost knowledge.

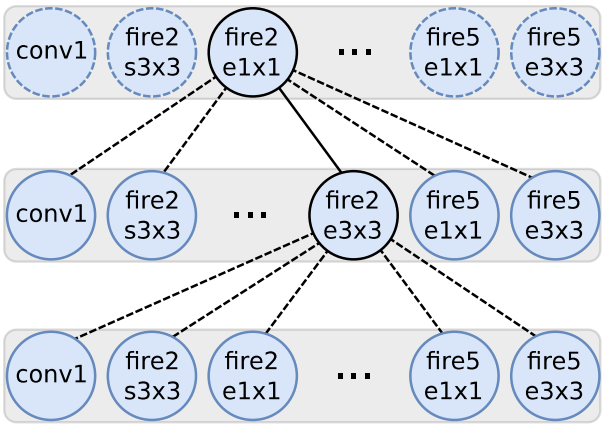


Figure 3: Visualization of the proposed autopruning approach. Dashed lines indicate considered pruning steps, solid black lines indicate chosen pruning steps.

Compared to other approaches like [32] and [33], our proposed algorithm only requires two hyperparameters φ and k, where φ determines the accuracy-speed trade-off and k defines the algorithm's convergence speed at the cost of a suboptimal pruning step. Furthermore, it does not require retraining after every pruning iteration and can therefore be completed in only a few hours on a single GPU. Unlike one-shot approaches, our iterative pruning approach also considers prior pruning steps when calculating the ℓ_1 norm and is therefore able to react to these changes. Additionally, the algorithm learns to avoid sensitive layers by monitoring the chosen metric.

4.4. Batch Normalization

Another important part of modern CNN architectures is the batch normalization layer proposed in [34]. During training, batch normalization computes mean μ and variance σ^2 values with which the input is normalized. A small value ϵ is used to prevent a division by zero. Scale γ and shift β are also added to enable the network to learn an identity transformation. Batch normalization is then given by Equation 1.

$$y = \gamma \frac{(Wx + b) - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta. \tag{1}$$

As μ, σ, γ and β are constant during inference, they can be merged into the weights and biases of the previous convolutional layer:

$$\hat{y} = \hat{W}x + \hat{b} \tag{2}$$

$$\hat{W} = \gamma\left(\frac{W}{\sqrt{\sigma^2 + \epsilon}}\right) \tag{3}$$

$$\hat{b} = \gamma\left(\frac{b - \mu}{\sqrt{\sigma^2 + \epsilon}}\right) + \beta \tag{4}$$

where $\hat{W}$ and $\hat{b}$ are the rescaled weights and biases. This procedure was also used in [21] and can help to achieve a

Value	Titan X	GTX 1060	Jetson TX2
CPU	48× Intel Xeon E5-2650 v4	12× Intel Xeon E649	4× ARM Cortex-A57 + 2× NVIDIA Denver2
RAM	256 GB	128 GB	8 GB (shared with GPU)
GPU	TITAN X (Pascal), 12 GB	GTX 1060 (Pascal), 6 GB	Tegra X2 (Pascal), 8 GB (shared with RAM)

Table 1: Platforms used for benchmarking.

large speedup without losing accuracy.

5. Experimental Results

The *DLR 3K Munich Vehicle Aerial Image Dataset* [11] was used for the initial experiments. The dataset contains 20 aerial images with a resolution of 5616×3744 px and a ground sampling distance (GSD) of 13 cm. The images are split into 10 training and 10 test samples. Due to the large extents, every image was cut into tiles of size 936×624 px. Only tiles containing at least one object were considered for our experiments. The annotations contain seven different vehicle types given as oriented bounding boxes. Due to the lack of annotations for most vehicle types, only the `car` and `van` labels were considered and merged into a single `vehicle` class. Furthermore, all oriented bounding boxes were converted to axis-aligned bounding boxes according to [1, 2, 7].

All SSD experiments were trained and evaluated with the original Caffe SSD implementation [10]. For training, random crops and rotations were introduced in addition to the photometric data augmentations of the SSD framework, as the dataset contains only 10 full size training images. For training, we use the Adam optimizer [35], an initial learning rate of 10^{-3} and a mini-batch size of 16.

In terms of detection accuracy, the models were evaluated by plotting the precision-recall curves and calculating the area under the curve, which is known as the average precision (AP) metric. Every model was then benchmarked on three different platforms, representing a server and desktop GPU, corresponding to ground-control-station or offline processing, and the NVIDIA Jetson TX2, which can be integrated into UAVs for on-board processing (see Table 1). The `MAX-N` power preset was used, allowing for the highest inference performance at the cost of a higher power consumption (up to 15 W). Inference speed is reported in frames per second (FPS) averaged over 500 forward passes. The employed image input sizes are 936×624 px for DLR 3K. Note that the benchmarks do not include the NMS stage (unless otherwise noted) to better judge the architectural changes. Please refer to the supplementary material for benchmarks including the NMS stage.

5.1. Baseline Experiments

Initially, different detection meta-architectures are compared in Table 2. For SSD and Faster R-CNN, we use VGG-16 as base network and employ `conv4_3` as feature map. The default box settings described in Section 4.1 are used for SSD. For Faster R-CNN, we adopt the anchor settings from [3]. In case of YOLOv2, we employ the 10th convolutional layer of Darknet-19 [19] as feature map in order to provide equal feature map resolutions. SSD and Faster R-CNN reach a similar AP of over 97% while YOLOv2 only achieves 95.7%. In terms of inference speed, Faster R-CNN was the slowest meta-architecture due to the overhead of its two-stage design. When running on the Jetson setup, Faster R-CNN could only process a frame every two seconds. Note that for YOLOv2, the original Darknet framework was used for benchmarking. This could achieve 2.7 frames per second on the on-board platform. However, SSD was faster for the desktop and server setup while also being more accurate.

Detection Framework	AP (%)	Inference Speed (FPS)		
		Titan X	GTX 1060	Jetson TX2
Faster R-CNN	97.2	6.0	3.0	0.5
SSD	**97.3**	**24.7**	**10.1**	1.2
YOLOv2	95.7	10.9	5.9	**2.7**

Table 2: Comparison of different detection meta-architectures (including NMS stage). Note that YOLOv2 benchmarks are conducted with the Darknet framework.

Boxes	conv4_3	fc7	conv6_2	conv7_2	conv8_2	conv9_2	AP (%)	Inference Speed (FPS)		
								Titan X	GTX 1060	Jetson TX2
2	✓	✗	✗	✗	✗	✗	**97.3**	26.7	10.7	1.2
1	✓	✗	✗	✗	✗	✗	97.2	**27.9**	**11.7**	**1.3**
1	✓	✓	✗	✗	✗	✗	96.6	22.1	9.0	1.1
1	✓	✓	✓	✗	✗	✗	96.1	21.6	8.8	1.1
1	✓	✓	✓	✓	✗	✗	96.1	21.3	8.7	1.1
1	✓	✓	✓	✓	✓	✗	96.0	21.1	8.6	1.1
1	✓	✓	✓	✓	✓	✓	96.1	20.8	8.5	1.1

Table 3: Baseline results for VGG-SSD (including NMS stage): Different feature maps and number of default box sizes in comparison.

Overall, SSD yielded the best trade-off between detection accuracy and speed and was therefore chosen as the meta-architecture for further experiments. Table 3 depicts the impact of the number of default box sizes on the detection performance. For this purpose, the number of default box sizes was reduced to one by simply using the mean of the box size distribution (31px). Using only one default box size yielded a comparable detection accuracy, while being slightly faster on all benchmarking platforms.

Table 3 also depicts the effect on AP when using multiple feature maps with different resolutions. All experiments clearly show that no layer after `conv4_3` could help to improve the detection accuracy. This is mostly caused by the constant ground sampling distance and vehicle size of the dataset as discussed in Chapter 4.1. For the benchmarks, adding `fc7` causes a noticeable drop in inference speed due

Network Architecture	AP (%)	Inference Speed (FPS)		
		Titan X	GTX 1060	Jetson TX2
MobileNet$^{BN}_{\alpha=1.00}$	**97.2**	70.9	35.3	6.8
MobileNet$_{\alpha=1.00}$	**97.2**	124.9	67.1	11.7
MobileNet$_{\alpha=0.75}$	96.8	150.4	78.9	14.3
MobileNet$_{\alpha=0.50}$	94.1	**199.9**	**102.7**	**19.6**
ShuffleNet^{BN}	96.4	55.2	35.7	10.8
ShuffleNet	96.4	91.0	59.7	14.8
SqueezeNet v1.0	**97.2**	135.9	60.9	10.2
SqueezeNet v1.1	**97.2**	183.7	81.2	14.6
ZynqNet	**97.2**	184.9	81.9	14.7

Table 4: Results for the MobileNet, Shufflenet, SqueezeNet and ZynqNet architectures. Models marked with *BN* also include the batch normalization layers during the benchmark.

to its large dimensions. Later feature maps like `conv7_2` have already been pooled to a spatial size of 15×10 or less and do therefore not contribute to the inference time in a relevant way. Overall, using the VGG model with a single feature map and one default box size yielded the best trade-off and is therefore used as baseline for experiments involving mobile networks.

5.2. Base Network Architectures

Furthermore, evaluation results for all mobile network architectures introduced in Chapter 3.2 are given in Table 4. Only one default box size was used for all network architectures. For ShuffleNet, the ShuffleNet 1×, g=3 model was used, as it provides the best trade-off between accuracy and inference speed according to its authors [13]. For each network architecture, we employ the output of the last layer with an approximate downsampling factor of 8 as feature map. Thus, the feature maps exhibit similar resolutions compared to `conv4_3` of VGG-16. MobileNet and ShuffleNet are commonly trained with batch normalization layers. Hence, benchmarks are reported for both architectures with and without these layers, using the merging process described in Chapter 4.4. In both cases, merging the batch normalization led to a vast gain in inference speed, with an improvement of up to 54 frames per second on the server setup. The merging process does not affect AP as it is an identity transformation.

For MobileNet, the impact of the channel depth multiplier α is evaluated. For a given layer, the channel depth multiplier $\alpha \in (0, 1]$ reduces the number of input channels M to αM and the number of output channels N to αN [12]. This is done before training and leads to a fixed, smaller architecture. In comparison, MobileNet retained a high AP even for the smallest model with $\alpha = 0.5$. Unfortunately, MobileNet is currently thwarted by the depthwise convolution implementation, often leading to inferior throughput when compared to models that only rely on standard convolutions.

All fully-sized networks of the SqueezeNet family could reach VGG-level AP while running up to 11 times faster. Expectedly, ShuffleNet could not reach the 97.2% AP mark, due to its fast downsampling approach. Overall, ZynqNet delivers the highest inference speed while also providing detection results close to the VGG baseline. Additionally, it only consists of standard convolutional and concatenation layers and does not require special layers (*e.g.* a channel shuffle layer). Therefore, ZynqNet is chosen as the base network architecture for further refinement experiments.

Network Architecture	AP (%)	Inference Speed (FPS)		
		Titan X	GTX 1060	Jetson TX2
ReLU	97.2	184.9	81.9	14.7
no squeeze ReLU	**97.3**	185.5	82.5	14.7
PReLU	97.1	157.8	70.2	12.6
leaky ReLU	97.1	185.6	82.4	**14.9**
ELU	**97.3**	**186.4**	**82.7**	14.7

Table 5: Comparison of Zynqnet with different activations.

Network Architecture	AP (%)	Inference Speed (FPS)		
		Titan X	GTX 1060	Jetson TX2
Normal 3×3 convolutions	97.2	184.9	81.9	14.7
Separate 1×1 convolutions	97.2	188.4	83.3	**15.5**
Separate 1×1 group convolutions	97.2	**188.7**	**83.5**	**15.5**
Separate DSC block	**97.3**	162.0	76.3	13.8
Separate squeeze/expand block	97.2	182.7	80.9	15.0
Shared DSC block	97.2	174.7	79.8	14.6
Shared squeeze/expand block	97.1	184.8	82.1	15.2

Table 6: Building blocks for auxiliary regression and classification layers.

5.3. Activations

Motivated by the results of Dong *et al.* [36] and Treml *et al.* [37], we first evaluate the effect of the applied activations. ZynqNet usually uses the ReLU activation after every convolution block. Table 5 shows both AP and inference speed when replacing ReLU with the related activation functions PReLU, leaky ReLU and ELU. Additionally, the currently unpublished SqueezeNet v1.2 idea is transfered to ZynqNet by removing the ReLU activations after the squeeze blocks. All experiments arrived at a similar AP close to the baseline ReLU model with ZynqNet v1.2 and ELU slightly improving the AP by 0.1%. In terms of runtime, removing ReLU activations led to a measurable improvement for the server setup while results on the on-board setup stayed the same. For leaky ReLU and ELU, runtime improved due to implementation details of the Caffe framework. Using PReLU degraded performance due to the overhead associated with parameter loading and the more complex computation.

5.4. Auxiliary SSD Layers

We further evaluate the effect of replacing the normal 3×3 convolutions in the auxiliary SSD layers with more efficient building blocks in Table 6. For the separate setup (see Figure 2a), blocks A and B are set to either simple 1×1

convolutions, 1×1 group convolutions, a full DSC block for both A and B or a "squeeze" block, using 1×1 convolutions for channel depth reduction followed by 3×3 "expand" convolutions. The number of groups was set to 2 for the experiment involving group convolutions, as this is the only choice that evenly divides the number of input and output channels. For the shared setup (see Figure 2b), the DSC was split into a depthwise block A and pointwise blocks B_C and B_L. The squeeze/expand block was divided similarly, with a shared "squeeze" block A.

The results indicate that every building block could replace the normal 3×3 kernels without loss of accuracy. In terms of inference speed, only a slight speedup could be achieved for 1×1 (group) convolutions, as the number of default box sizes and feature maps was already optimized beforehand (see Table 3). Models involving depthwise separable convolutions experienced a small drop in throughput due to the currently inefficient implementation. Even 1×1 (group) convolutions could reach the 97.2% AP of the original model. As the receptive field is already noticeable larger than the size of present vehicles, the additional spatial context due to an auxiliary 3×3 filter is not required. Thus, we propose that simple 1×1 (group) convolutions are sufficient as auxiliary heads.

5.5. UAV-Net Architecture

For the UAV-Net architecture, all the improvements of the previous sections are now combined. The main speedup is achieved by replacing the initial VGG-16 base network with our modified ZynqNet. The regression and classification filters are replaced by simple 1×1 convolutions and the ReLU activations after the squeeze layers are removed. The remaining activations are replaced by ELU. Retraining this modified network yields the same test performance as the original network, as shown in Table 7. However, inference speed improves, resulting in a gain of 1.2 FPS for the on-board and 9.2 FPS for the server setup. Although 1×1 group convolutions also lead to a similar result in Table 6, normal 1×1 convolutions were chosen for UAV-Net, as group convolutions would impose an additional symmetry constraint for the proposed pruning approach discussed in Chapter 5.6. Please refer to the supplementary material for a full network description.

5.6. Filter Pruning

In this section, we evaluate our proposed automatic pruning approach. The pruning algorithm described in Chapter 4.3 is applied to the fully-sized UAV-Net$_{\varphi\,=\,1.00}$. Empirically, we found that setting $k = 4$ provides a large speedup while not impairing the pruning process in a noticeable way. In total, the pruning UAV-Net took less than a day on a single GPU. Note that the pruning process cannot remove whole convolutional layers by pruning all filters. Therefore, at least k filters will remain in every layer. Results for characteristic φ values are provided in Table 7. Even when setting $\varphi = 0.5$, UAV-Net can reach an AP value close to the VGG reference, while being more than 21 FPS faster on the Jetson TX2. Pruning more than 50% of the filters in UAV-Net leads to a larger degradation in AP with up to 38 FPS on the mobile platform. However, setting $\varphi = 0.075$ results in a huge drop in terms of AP. The reason for this is the squeeze/expand structure of UAV-Net. For $\varphi = 0.075$, almost all expand layers have already been pruned until only k filters remain, therefore leaving our algorithm no choice but to prune the squeeze layers. As even the fully-sized UAV-Net has only 16 filters in the upper squeeze layers, pruning additional filters leads to the observed collapse. However, our algorithm learned to defer this suboptimal pruning step until no other choice remained.

Network	AP (%)	Inference Speed (FPS)		
		Titan X	GTX 1060	Jetson TX2
VGG	**97.2**	27.9	11.7	1.3
ZynqNet	**97.2**	184.9	81.9	14.7
UAV-Net$_{\varphi\,=\,1.000}$	**97.2**	194.1	83.8	15.9
UAV-Net$_{\varphi\,=\,0.750}$	**97.2**	225.8	98.8	18.8
UAV-Net$_{\varphi\,=\,0.500}$	97.1	265.2	116.2	22.7
UAV-Net$_{\varphi\,=\,0.250}$	95.4	342.6	153.3	31.3
UAV-Net$_{\varphi\,=\,0.150}$	91.3	410.0	181.2	38.2
UAV-Net$_{\varphi\,=\,0.075}$	11.1	**426.8**	**203.5**	**43.1**

Table 7: UAV-Net on DLR 3K with different φ values in comparison to selected baseline models.

Network architecture	Model Size	Parameter Count	Relative Size
VGG, 2 box sizes	30.19 MiB	7,912,316	100.0%
ZynqNet	0.89 MiB	230,782	2.9%
UAV-Net$_{\varphi\,=\,0.50}$	0.39 MiB	101,934	1.3%
UAV-Net$_{\varphi\,=\,0.15}$	**0.07 MiB**	**17,146**	**0.2%**

Table 8: Model size (on disk) and parameter counts for selected network architectures.

In addition to a speedup, filter pruning also leads to smaller model sizes. Table 8 shows the model size of selected networks in comparison. For $\varphi = 0.50$, UAV-Net reaches VGG-level AP with only 1.3% of its model size. UAV-Net$_{\varphi\,=\,0.15}$ still reaches 91.3% AP at 0.2% of VGG-SSD's model size.

Qualitative detection results for UAV-Net$_{\varphi=0.50}$ are provided in Figure 4. As usual in aerial vehicle detection, false positives are mostly caused by roof structures. Please refer to the supplementary material for further visualizations.

5.7. Transfer to other Datasets

To identify an optimized model for on-board deployment under various conditions, we conduct experiments on the VEDAI and UAVDT datasets. VEDAI is comprised of 1268 images of size 1024×1024 px and a GSD of 12.5 cm, which is similar to DLR 3K. According to [1], only small vehicles,

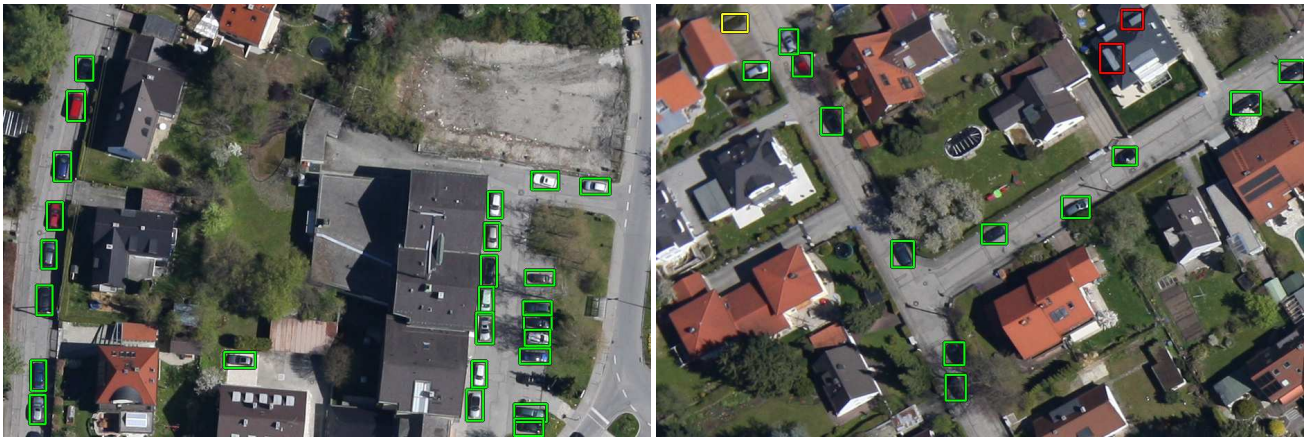


Figure 4: Qualitative results for UAV-Net$_{\varphi=0.50}$ on DLR 3K. A confidence threshold of 50% was used to generate the detections (green boxes = true positives, yellow boxes = false negatives, red boxes = false positives).

Dataset	Model	AP (%)	Inference Speed (FPS)		
			Titan X	GTX 1060	Jetson TX2
VEDAI	VGG	**96.4**	16.7	5.8	0.7
VEDAI	UAV-Net$_{\varphi=1.00}$	95.7	123.5	50.2	9.9
VEDAI	UAV-Net$_{\varphi=0.50}$	95.2	168.0	73.8	13.9
VEDAI	UAV-Net$_{\varphi=0.15}$	93.5	**256.4**	**125.9**	**22.9**
UAVDT	R-FCN [17]	34.35	4.7	–	–
UAVDT	SSD [17]	33.62	41.6	–	–
UAVDT	Faster R-CNN [17]	22.32	2.8	–	–
UAVDT	RON [17]	21.59	11.1	–	–
UAVDT	UAV-Net$^{1\times1,c=1}_{\varphi=1.00}$	26.21	**214.0**	**98.8**	**18.3**
UAVDT	UAV-Net$^{5\times5,c=5}_{\varphi=1.00}$	**34.52**	80.1	34.7	6.6
UAVDT	UAV-Net$^{3\times3,c=4}_{\varphi=1.00}$	32.76	112.2	51.5	9.0
UAVDT	UAV-Net$^{3\times3,c=4}_{\varphi=0.50}$	31.82	132.5	69.2	11.4

Table 9: UAV-Net results on VEDAI-1024 and UAVDT. Models from [17] were benchmarked differently. c is the number of clustered box sizes.

namely the classes `car`, `pickup`, and `van`, are considered for evaluation. UAVDT comprises roughly 80k images of size 1024×540 px. In contrast to DLR 3k and VEDAI, images are recorded at different flying altitudes with different camera angles. In order to show the general applicability of our UAV-Net detector, we finetune the DLR 3K pretrained models with a 10 times lower learning rate and report the results in Table 9. Please note that the reference UAVDT benchmarks reported in [17] were conducted on a workstation with an Intel i9-7900X CPU and a NVIDIA GTX 1080 Ti GPU with a different benchmarking protocol. AP for UAVDT is calculated with the official MATLAB evaluation script. To measure the inference time, image input sizes are 1024×1024 px for VEDAI and 1024×540 px for UAVDT.

On VEDAI, even UAV-Net$_{\varphi\,=\,0.15}$ could reach an AP close to the VGG baseline while being more than 22 FPS faster on the Jetson TX2. For UAVDT, our assumption from Chapter 5.4 does not hold anymore, as UAVDT exhibits large variations in vehicle scales due to different camera angles and various GSDs. Furthermore, the fixed box size from Chapter 4.1 does not fit the various object sizes anymore. We therefore also provide results for UAV-Net with modified prediction layers for a larger receptive field and more default box sizes found by clustering the training set boxes (denoted as c). Using 3×3 convolutions and four default box sizes shows a good trade-off between detection accuracy and inference speed. UAV-Net$^{5\times5,c=5}_{\varphi=1.00}$ even surpasses the best result reported in [17]. Please refer to the supplementary material for a detailed ablation study.

6. Conclusion

In this paper, multiple state-of-the-art deep learning object detectors were adjusted for the task of vehicle detection in aerial UAV imagery with on-board deployment in mind. The most promising architectures were further examined and improved by changes to the activations, auxiliary prediction layers and an automatic pruning approach. With these changes in mind, we propose UAV-Net, a novel vehicle detector for on-board inference that can run in real time on a Jetson TX2 while retaining an AP close to the reference baselines on the DLR 3K, VEDAI and UAVDT datasets.

Although a multitude of different approaches and aspects were evaluated, room for further optimization remains. The recently proposed ShuffleNet v2 [38] and MobileNet v2 [31] architectures could be investigated as alternative base networks and might yield an additional speedup. Finally, model quantization like FP16 or INT8 inference could be used to further lower the model's memory footprint and achieve a higher inference speed.

References

[1] W. Sakla, G. Konjevod, and T. N. Mundhenk. Deep multimodal vehicle detection in aerial ISR imagery. In *WACV*. IEEE, 2017. 1, 2, 3, 5, 7

[2] L. W. Sommer, T. Schuchert, and J. Beyerer. Fast deep vehicle detection in aerial images. In *WACV*. IEEE, 2017. 1, 2, 3, 5

[3] L. Sommer, L. Steinmann, A. Schumann, and J. Beyerer. Systematic evaluation of deep learning based detection frameworks for aerial imagery. In *Automatic Target Recognition XXVIII*, volume 10648. SPIE, 2018. 1, 2, 3, 5

[4] Q. Li, L. Mou, Q. Xu, and Y. Zhang. R3-Net: A Deep Network for Multi-oriented Vehicle Detection in Aerial Images and Videos. *arXiv preprint arXiv:1808.05560*, 2018. 1, 2

[5] M. Y. Yang, W. Liao, X. Li, and B. Rosenhahn. Deep Learning for Vehicle Detection in Aerial Images. In *ICIP*. IEEE, 2018. 1, 2

[6] L. Sommer, A. Schumann, T. Schuchert, and J. Beyerer. Multi Feature Deconvolutional Faster R-CNN for Precise Vehicle Detection in Aerial Imagery. In *WACV*. IEEE, 2018. 1, 2

[7] T. Tang, S. Zhou, Z. Deng, H. Zou, and L. Lei. Vehicle detection in aerial images based on region convolutional neural networks and hard negative example mining. *Sensors*, 17(2):336, 2017. 1, 2, 5

[8] Z. Deng, H. Sun, S. Zhou, J. Zhao, and H. Zou. Toward fast and accurate vehicle detection in aerial images using coupled region-based convolutional neural networks. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(8):3652–3664, 2017. 1, 2

[9] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, 2015. 1, 2

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot multibox detector. In *ECCV*, 2016. 1, 2, 5

[11] K. Liu and G. Mattyus. Fast Multiclass Vehicle Detection on Aerial Images. *IEEE Geosci. Remote Sensing Lett.*, 12(9):1938–1942, 2015. 1, 2, 5

[12] 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. 1, 2, 3, 6

[13] X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *CVPR*, 2018. 1, 3, 6

[14] 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 <0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016. 1, 2, 3

[15] B. Wu, F. N. Iandola, P. H. Jin, and K. Keutzer. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving. In *CVPR Workshops*, 2017. 2

[16] S. Razakarivony and F. Jurie. Vehicle detection in aerial imagery: a small target detection benchmark. *Journal of Visual Communication and Image Representation*, 34:187–203, 2016. 2

[17] D. Du, Y. Qi, H. Yu, Y. Yang, K. Duan, G. Li, W. Zhang, Q. Huang, and Q. Tian. The unmanned aerial vehicle benchmark: object detection and tracking. In *ECCV*, 2018. 2, 8

[18] J. Dai, Y. Li, K. He, and J. Sun. R-FCN: Object detection via region-based fully convolutional networks. In *NIPS*, 2016. 2

[19] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CVPR*, 2017. 2, 5

[20] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature Pyramid Networks for Object Detection. In *CVPR*, 2017. 2

[21] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. DSSD: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017. 2, 3, 4

[22] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li. Single-Shot Refinement Neural Network for Object Detection. In *CVPR*, 2018. 2

[23] A. Wong, M. J. Shafiee, F. Li, and B. Chwyl. Tiny SSD: A Tiny Single-shot Detection Deep Convolutional Neural Network for Real-time Embedded Object Detection. In *CRV*, 2018. 2

[24] R. J. Wang, X. Li, S. Ao, and C. X. Ling. Pelee: A Real-Time Object Detection System on Mobile Devices. In *NeurIPS*, 2018. 2, 3

[25] Y. Li, J. Li, W. Lin, and J. Li. Tiny-DSOD: Lightweight Object Detection for Resource-Restricted Usages. In *BMVC*, 2018. 2

[26] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue. DSOD: Learning deeply supervised object detectors from scratch. In *ICCV*, 2017. 2

[27] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun. Light-Head R-CNN: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017. 2

[28] S. M. Azimi. ShuffleDet: Real-Time Vehicle Detection Network in On-board Embedded UAV Imagery. In *ECCV*, 2018. 2

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. 3

[30] D. Gschwend. ZynqNet: An FPGA-accelerated embedded convolutional neural network. *Master thesis, ETH-Zurich*, 2016. 3

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018. 3, 8

[32] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *ICLR*, 2016. 4

[33] S. Han, J. Pool, and J. Tran. Learning both weights and connections for efficient neural network. In *NIPS*, 2015. 4

[34] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 4

[35] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 5

[36] X. Dong, G. Kang, K. Zhan, and Y. Yang. EraseReLU: a simple way to ease the training of deep convolution neural networks. *arXiv preprint arXiv:1709.07634*, 2017. 6

[37] M. Treml, J. Arjona-Medina, T. Unterthiner, et al. Speeding up semantic segmentation for autonomous driving. In *MLITS, NIPS Workshop*, 2016. 6

[38] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. *ECCV*, 2018. 8