Analysis of efficient CNN design techniques for semantic segmentation

Alexandre Briot
GEEDS AI, Valeo Creteil
alexandre.briot@valeo.com

Prashanth Viswanath, Senthil Yogamani
Valeo Vision Systems, Ireland
prashanth.viswanath, senthil.yogamani@valeo.com

Abstract

Majority of CNN architecture design is aimed at achieving high accuracy in public benchmarks by increasing the complexity. Typically, they are over-specified by a large margin and can be optimized by a factor of 10-100x with only a small reduction in accuracy. In spite of the increase in computational power of embedded systems, these networks are still not suitable for embedded deployment. There is a large need to optimize for hardware and reduce the size of the network by orders of magnitude for computer vision applications. This has led to a growing community which is focused on designing efficient networks. However, CNN architectures are evolving rapidly and efficient architectures seem to lag behind. There is also a gap in understanding the hardware architecture details and incorporating it into the network design. The motivation of this paper is to systematically summarize efficient design techniques and provide guidelines for an application developer. We also perform a case study by benchmarking various semantic segmentation algorithms for autonomous driving.

1. Introduction

The complexity of Convolution Neural Networks (CNN) architectures have been growing consistently. However for industrial applications, there is a computational bound because of limited resources on embedded platforms. It is essential to design efficient models which fit the run-time budget of the system. There are many papers which demonstrate large runtime improvements with minimal loss of accuracy by using various techniques. An overview of efficient CNN from a hardware perspective is provided in [36] and guidelines for design of small networks is provided in [16]. A comparison of accuracy of networks normalized to complexity is presented in [3]. Figure 1 reproduced from this paper illustrates that there is large variability in the effective capacity of different networks. Huang et al [14] perform a detailed accuracy/performance trade-off comparison of various meta-architectures like Faster R-CNN, R-FCN and SSD.

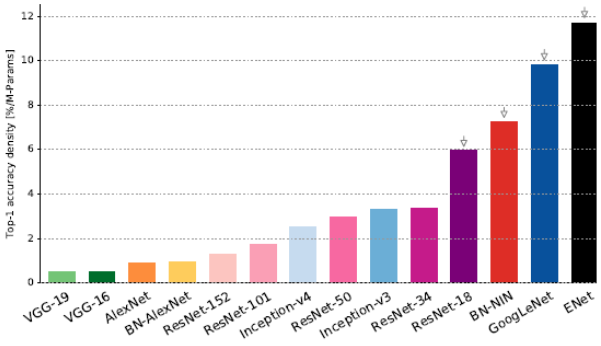

Figure 1. Illustration of large disparity in the efficiency of networks measured by an accuracy metric normalized to number of parameters. Figure is reproduced from [3].

The authors observe three main gaps in the current literature on efficient CNNs: (1) Most of these studies use relatively older networks like AlexNet or VGG16, (2) Different papers exploit different techniques and there is no systematic study of combination of optimization methods and (3) Most of the methods attempt to reduce the number of operations which may not lead to efficient mapping on hardware architectures. The motivation of this paper is to summarize efficient design techniques, share benchmarks of common network architectures and design guidelines for semantic segmentation.

The rest of the paper is structured as follows. Section 2 provides a short overview of the building blocks and design techniques of CNN. Section 3 provides a taxonomic survey of efficient CNN design and optimization techniques. Section 4 provides an overview of hardware (HW) architectures focusing on optimization trade-offs specific to hardware. Section 5 discusses a case study on semantic segmentation for autonomous driving and summarizes commonly used efficient design techniques. Finally, section 6 summarizes the paper and provides potential future directions.

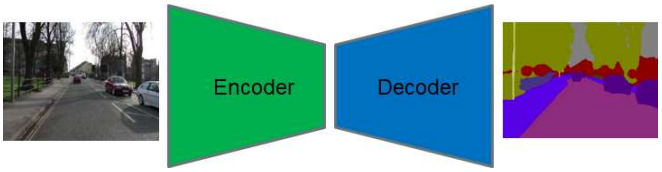

Figure 2. Typical encoder-decoder architecture of CNN based semantic segmentation network.

Figure 3. Baseline network with convolution blocks (/2 indicates downsampling, the last number indicates number of channels)

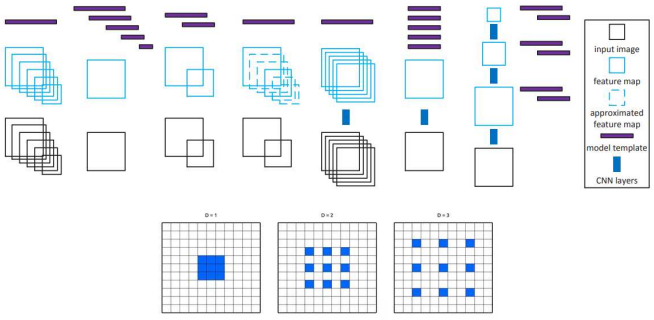

Figure 4. Illustration of various strategies for handling different sizes of objects. Figure on the top illustrating model variations is re-produced from [2]. Figure on the bottom illustrates dilated convolutions which is an alternative way to handle scale.

2. CNN architecture design concepts

In this section, we provide an overview of building blocks of CNN and their design aspects. Although the focus is on semantic segmentation architectures, most concepts are generic and applies to other visual perception tasks like bounding box object detection.

Encoder-Decoder architecture: Semantic segmentation architectures typically have an encoder and a decoder as shown in Figure 2. The encoder extracts features from the image which is then decoded to produce semantic segmentation output. ImageNet pre-trained networks are typically used as encoder. In early architectures [1] [31], decoder was a mirror image of encoder and had the same complexity. Newer architectures use a relatively smaller decoder. There can also be additional connections from encoder to decoder. For example, Segnet [1] passes max-pooling indices and Unet [31] passes intermediate feature maps to decoder.

Baseline FCN: The simplest CNN encoder is a linear cascade of convolution operators typically known as Fully Convolutional Networks (FCNs). The standard design is to progressively reduce the resolution of feature maps and increase the number of channels. A systematic empirical evaluation of all the hyper-parameters and design criteria of FCN is provided in [28]. Modular design of replicating building blocks is quite popular and it has been successfully demonstrated in ResNet and GoogleNet. The image width and height are powers of 2 typically and thus a progressive down-sampling by a factor of 2 is efficient. A typical FCN network is illustrated in Figure 3 and it is considered as a baseline network. Other key design ideas which can be built on top of this are abstracted out based on the survey of various networks and discussed below.

Cross channel filters: Cross channel filters can model functional dependencies across channels. This is typically a faster approximation of a more generic 3D convolution filter. Inception network successfully demonstrated this idea first and XceptionNet (Figure 5 (a)) optimized it for efficiency even further [5].

Cross layer connections: ResNet skip connections pass the previous layer output to the next layer via a summation junction. DenseNet took this further where all the layers have connections to their previous ones as illustrated in Figure 5 (b). HighwayNet employs a similar idea but allows data-driven learning of relevant connections [35]. The cross layer connections provide exponential number of connections from input to output as it passes through several layers.

Wider bank of filters: The general trend to improve the CNN capacity is to add deeper layers. In contrast, [39] shows that 16 wider layers can attain similar capacity of 100s of layers. This design can have advantages of parallelism for training and inference. It can also enable feature dependent gating. Figure 5 (c) illustrates progressively wider bank of filters.

Split branching and summation joining: The simplest form was introduced by ResNet for skip connections. It is more efficient than concatenation joining which increases feature dimensionality. ResNeXt extended the idea for more parallel paths [40] and PolyNet (shown in Figure 5 (d)) generalized it further to achieve higher efficiency [43].

Handling scale of objects: Detecting objects at different scale is very important for many applications like autonomous driving. CNNs are not scale invariant and there is plenty of empirical evidence which illustrates explicit handling of scale [2]. Figure 4 illustrates different strategies to handle scale. For segmentation, dilated convolutions are commonly used for handling multiple scales. However, they are not efficient for embedded implementation as they access non-contiguous data. Outputs of different layers of CNN naturally express scale and the efficient approach would be to share multi-scale feature maps from different layers to the decoder [24].

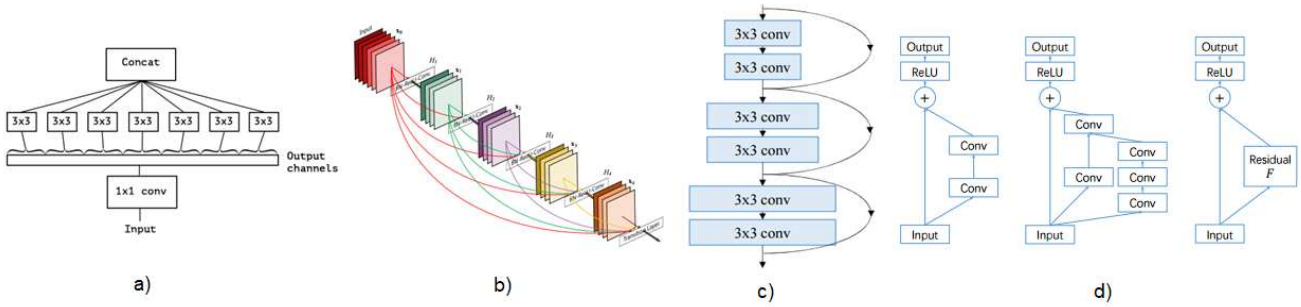

Figure 5. Canonical examples of design extensions - (a) cross channel filters [5], (b) cross layer connections [13], (c) wider bank of filters and (d) Split branching and summation joining [43]

3. Optimization Techniques

There have been various approaches to reduce the complexity of CNNs. In this section, we discuss some of the important techniques that are relevant for embedded platforms.

3.1. Quantization

The standard data format for CNN models is 32-bit floating point but full precision (fp32) operations are computationally intensive, both for training and inference. The need to make inference predictions scalable with real time low resource embedded systems has boosted research on reducing the precision of data representation. This quantization process can cover both the weights and the activations, reducing both computational cost and memory needs. There is plenty of empirical evidence showing that it is possible to reduce pre-trained networks with fp-32 weights and activations to a lower resolution such as INT8 with minimal loss in accuracy. This has led to customized 8-bit CNN hardware such as Google TPU. Model conversion for 8-bit inference is now largely supported by DL frameworks such as Google Tensorflow and Nvidia Tensor RT [46].

A better alternative to quantize trained network is to directly train network with low resolution. Low resolution can be either applied just to weights/activations or even gradients. Compared to offline low resolution conversion, training neural networks with low precision results in larger accuracy loss. Different levels of quantization can be considered, binary networks are the extreme case. XNOR-Nets [30] show peaks improvements of 58x faster operations and 32x memory savings with only 2.9 percents accuracy loss on AlexNet. Accuracy is more sensitive to low resolution activations than low resolution weights and even more sensitive to quantization of gradients. More generic Quantized Neural Networks [15] offers better quantization scheme and 1-2-6 bit quantization (weight-activation-gradient) seems to be a optimum balance when exploring full quantization.

3.2. Efficient architecture design principles

Compared to fully connected network architecture, FCN networks drastically save a huge number of parameters and computational cost by introducing local connections and parameters sharing. However, CNNs still have huge redundancy which can be reduced by exploring the rich space of more advanced architecture design techniques than original vanilla structure. Decreasing model size by limiting depth and/or feature maps number is trivial but not optimal strategy since it directly impacts the model accuracy.

3.2.1 Scaling the model size through architecture hyperparameters

From the first CNN success namely AlexNet to ResNet architecture, ImageNet challenge accuracy has mostly been powered by the constant increase of the number of layers and feature maps per layer resulting in over parametrized models. The first basic idea towards efficiency is to better balance model size and accuracy by limiting these high level architecture hyperparameters :

- Network depth : As it is now commonly known, increasing network depth helps to learn hierarchically high level features to better detect semantic patterns and thus enable better predictions.
- Channels per layer : Iandola [16] illustrates a controlled way of limiting the number of input channels for a convolutional layer referred to as channel reduction. Channel reduction reduces memory footprint and computational cost.
- Input image size : Input image resolution and early downsampling stage can also help to reduce computation and at some lower level to decrease memory dedicated to temporaries.

As an early attempt, Reddie [47] built an efficient design of a vanilla CNN architecture with both few computations and few parameters. His tiny-darknet model achieves

same accuracy as AlexNet with 60x and 2x less parameters and operations respectively. Comparing tiny-darknet and SqueezeNet efficient designs, it was pointed out that SqueezeNet is efficient only for parameter count but requires as much computation as AlexNet. This illustrates that efficiency has to be considered in both aspects of memory and computation. Power consumption is a third important aspect of efficiency but is more related to hardware architecture than network design. Most recent architectures targeting real time embedded application such as MobileNet [11] and ShuffleNet [44] provide a family of parametrized architecture designs with different input resolution (to limit computations) and channel depth (to limit memory and computational costs).

3.2.2 Filter redundancy and kernel reduction

Filter kernels spatial size: The design of the very first deep convnets such as AlexNet[22] and ZF-Net[42] relied on using larger receptive fields kernels especially in the first few layers. Simonyan et al [34] democratized the use of smaller 3x3 kernels with VGG-16. They argued that stacking three 3x3 convolutional layers corresponds to an effective 7x7 receptive layer but can achieve this with less parameters and computation. This has been a way to introduce regularization by forcing kxk layers to have a decomposition through 3x3 layers (with some additional non linearity in between).

Filter kernel spatial factorization: There exist more efficient ways to decompose kxk convolutions by the sequence of separable 1D convolutions (1xk and kx1 kernel) as illustrated in [29]. Spatial factorization into separable asymmetric convolutions has also been considered in the upgraded version of inception module [37]. For 3x3 convolution, Szegedy et al. have evaluated that a factorized two-layer solution (using a 3x1 convolution followed by a 1x3 convolution) is 33% cheaper for the same number of output filters, if the number of input and output filters are equal. In practice, they showed that this approximation provided good results only in the early layers. Ioannou and al in [18] referred to this factorization as sequential separable filters and gave an insight of a more generic way of defining low-rank filters decomposition as learning a set of small basis of filters.

Regularizing filter kernel through activation functions: More recently, some strategies to compute multiple features maps with the same filter kernel have been presented as a way to reduce filter redundancy. Based on this idea, Shang et al. built new advanced activation function called Concatenated Rectified Linear Unit (CReLU) [33]. They observed that early conv layers of well-know convnets such AlexNet capture both negative and positive phase information through learning pairs of negatively correlated filters. CReLU exploits this redundancy by duplicating linear response of convolution and negating the copy before concatenating it with the original response. This process which makes more efficient use of trainable parameters has been adopted by Kim et al. in the initial layers of the PVANet architecture [21] to build lightweight network for real-time object detection.

Regularizing filter kernel through symmetry: Similar filter weights reuse to reduce redundancy has been taken advantage of by Cohen and Welling with Group equivariant Convolutional Neural Networks (G-CNNs) [6] by exploiting symmetry. CNN equivariance to translation is then extended to G-CNN equivariance to rotation and even reflection.

3.2.3 Convolution design

Different strategies have been elaborated to alleviate efficiency constraints using advanced convolutions design such as feature maps dimensionality reduction by 1×1 convolution, group convolution or more recently exploited separable convolutions.

1x1 convolutions as low dimensional feature maps embeddings: Number of parameters involved in the computation of a single convolution is directly linked to the number of channels of the previous layer. Different strategies have been elaborated to compress input feature maps in a lower dimensional space and almost every recent convnet architecture relies on 1×1 convolution, most often integrated in processing blocks called bottlenecks.

Standard convolution operates layer by layer to filter existing features in order to produce new representations. 1×1 convolutions are in some way different from larger kernel convolutions since they will build new features through computing linear combinations of existing ones.

Iandola et al. in [17] explicitly formalized the non linear dimensionality reduction within their micro-architectural fire modules composed of two sub-layers called squeeze layers and expand layers. Their reasoning is very elementary but yet ultra efficient : The total quantity of parameters in this layer is $(\text{number of input channels}) \times (\text{number of filters}) \times (k \times k)$. So, to maintain a small total number of parameters in a CNN, it is important not only to decrease the number of $k \times k$ filters, but also to decrease the number of input channels to the $k \times k$ filters. Using squeeze layers (1×1 conv) they can decrease the number of input channels for $k \times k$ filters the way they want. Now most architectures widely rely on this strategy, often referred as bottlenecks.

Very recent Mobilenet V2 [32] paper further discuss the

impact of ReLU non linearities coupled with 1×1 convolutions. They explain that the manifold of interest (the encoded information) should lie in a low-dimensional subspace of the higher dimensional activation space, if not the ReLu non linearities could hurt the accuracy. As a result, their model rely on thin bottleneck layers (which can be seen as the capacity of the network at each layer) coupled with intermediate expansion layer (seen as the expressiveness) on which they build non-linear convolutional blocks.

Filter Groups: Another similar way to minimize involved parameters in a given layer is filter groups. This very old concept dates back to 2012 AlexNet [22] implementation whose architecture was specifically designed with two separate convolutional filter groups across most of the layers with the main motivation of enabling training across two GPUs to overcome memory constraints. What has been seen during many years as an engineering trick has more recently been presented as an efficient technique to remove parameters and learn better representations.

Filter groups design a model where filters operate on a fraction of this input volume by creating g independent groups of filters with associated groups of feature map input channels. (Each filter sees only a fraction of the feature map channels within the input volume reducing filter dimensions in the channel extent from $k*k*C_i$ to $k*k*C_i/g$). This change does not affect the dimensions of the input and output feature maps but significantly reduces computational complexity and the number of model parameters.

ResNeXt architecture [40] proposed a model which integrates group convolution within ResNet building blocks. Huang et al. in [12] combined it with filter pruning to introduced learned group convolution. They both achieved state-of-the-art architecture regarding model efficiency.

Depth-wise Separable Convolutions: Some similar approach to efficiently reduce the channel extent of filter kernel size is the use of depth-wise separable convolution which pushes it to the extreme case where every single filter sees a single input channel. These convolutions were first used in the Xception model [5] and consists in a depth-wise spatial convolution performed independently over each channel of an input, followed by a point-wise 1x1 convolution, i.e. a 1x1 convolution, projecting the channels output onto a new channel space. In other words, they split a standard 3D convolution into two successive layers: first one filters and second on combines outputs from first layer.They can be interpreted as an extreme version of the GoogleNet Inception module but they differ in two ways: the channel-wise spatial convolution is performed first and there is no ReLU. Mobilenets [11] propose hyper-parametric model to propose light model for resource restricted application that can trade-off latency and accuracy.

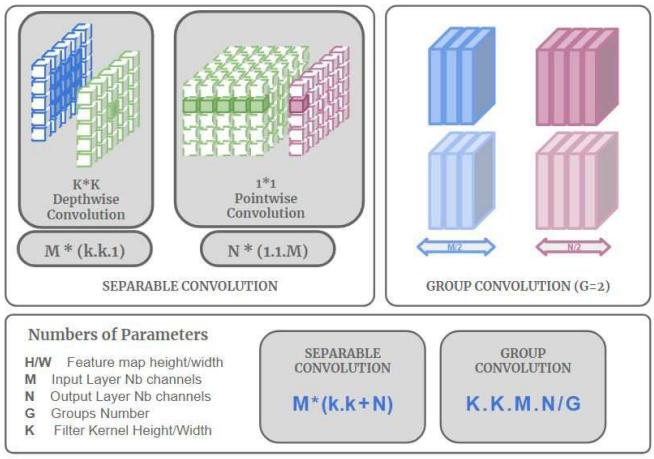

Figure 6. Efficiency: Depth-wise Separable Convolutions vs Group Convolutions

Efficiency - Depth-wise Separable Convolutions vs Group Convolutions: Compared to standard convolution, separable convolution and group convolution have respectively a $\frac{1}{N} + \frac{1}{k^2}$ and a g parameters reduction. Same reduction ratio is valid for computation. In separable convolution all the complexity is then related to 1x1 convolutions. Combining it with group convolution within 1x1 conv layer provides even more efficient architecture. This combined strategy inside a ResNet bottleneck with additional shuffle channel to maintain information flow has been illustrated by Zhang et al. with ShuffleNet in [44]. In practice, separable convolutions theoretical computation savings are difficult to achieve on HW implementation due to low arithmetic intensity (ratio of operations to memory accesses).

3.3. Model Compression

3.3.1 Weight Pruning

Most of the key concepts of deep compression have been introduced by Song Han et al. building compression framework around the very basic idea of weight pruning [9], trying to learn both weights and connections in a three steps process. The fact that regularization loss pushes parameters towards zero value lead the author to the conclusion that parameters with value close to zero characterizes connections which are not important for the learned model. Their 3-steps method first learns a model in a classical manner; then prunes unimportant connections based on their weight value and finally performs a last step which fine-tunes the remaining connections. This technique lead to impressive compression results by reducing the number of parameters of Alexnet by a factor x9 with no loss in accuracy. The main motivation of deep compression is to reduce model size mainly for network transfers or for small internal memory architectures like FGPA. The major drawback of this

technique is that the resulting compressed model is sparse and that the gain in computation speed on classical HW is far below the parameter reduction factor. Indeed most common computation platform do not take benefit of sparsity with their highly parallel dense matrix computation scheme. To alleviate sparsity constraint, Song Han presented in [8] an efficient inference engine relying on sparse matrix-vector multiplication with weight sharing. The resulting computation speed achieves x189 and x13 gain when compared to CPU and GPU implementations of the same CNN without compression. Model compression also enables networks to fit in the on-chip SRAM which reduces energy consumption per memory read by a factor x120 compared from fetching weights from DRAM.

3.3.2 Filter Pruning

A very similar approach to weight pruning is filter pruning. Whereas weight pruning results in sparse connectivity pattern, removing whole filters and their associated feature maps preserve dense connectivity. Consequently computation cost reduction does not rely on sparse convolution libraries or dedicated hardware and existing efficient BLAS libraries for dense matrix multiplication can be further used. Wen et al. [38] proposed filter pruning using model structure learning using group lasso which is an efficient regularization to learn sparse structures. In fact their method is even more general than filter regularization since the Structured Sparsity Learning (SSL) method can regularize any structure (i.e., filters, channels, filter shapes, and layer depth) of CNNs. This learning technique acts like a compression method to learn compressed model from bigger CNN reducing computation cost without hurting the computation pipeline by generating hardware friendly structured sparsity. Experiments proved an average speedup of x5.1 and x3.1 against CPU and GPU on reference AlexNet. More recent Condensenet architecture presented by [12] rely on learned grouped convolutions, which are based on filter groups coupled with Group Lasso learning strategy which enabled to automatically select important connections and subsequently prune away a fraction of unimportant filters for which weights have the lowest magnitude. Huang et al. presented in [23] another pruning approach not based on filter magnitude. The method rely on Reinforcement Learning to train a pruning agent which makes a set of binary actions to decide to remove each filter or not. It maximizes a reward function which combines two terms, the accuracy term (ensuring the performance drop is bounded) and the efficiency term (encouraging to prune more filters away).

3.3.3 Network Distillation

Another method to produce compressed networks is distillation introduced in [10] by Hinton et al. Basic concept is to transfer knowledge from a cumbersome trained model (teacher network) to a smaller model (student network). Training is supervised with a loss function not computed from training labels but instead from teacher network output distribution. In a classification task, student will be trained to fit teacher output softmax layer class probability values. In other works, student network will try to imitate the outputs of the teacher without using the data of the original training. Using teacher output distribution as soft targets rather than true labels as hard targets, distillation improves generalization. When true labels are available, this target distribution could be fine tuned.

4. Hardware Accelerators for CNN

Majority of the CNN inference run-time and optimizations are reported on Nvidia platforms. The optimization techniques can be highly dependent on the type of processor. Thus we provide an overview of various architectures from the perspective of CNN deployment.

4.1. GPUs

GPU (Graphics Processing Unit) was traditionally designed for graphics acceleration. GPUs excel at performing matrix operations (primarily matrix multiplications) that underlie graphics, AI, and many scientific algorithms. Basically, GPUs are very fast and relatively flexible. They are slowly being re-targeted to be used as additional resource for vision algorithms through OpenCL. With respect to Vision algorithms, CUDA (Compute Unified Device Architecture) of Nvidia is significantly more powerful than other GPUs provided by ARM (Mali), PowerVR (SGX) and Vivante. CUDA uses SIMT (single instruction multiple threads) with threading done in hardware. It has a limitation of dealing with load/store intensive operations which is improved by high data-bandwidth provided to it. GPU architecture is most widely used for training CNNs. However, lot of optimization techniques discussed in the previous section may not be efficient/supported on the GPU. Sparse matrix multiplications would not be efficient on the GPU. Quantization could also not be supported initially as most GPUs only supported floating point (fp32) operations. However, the recent GPUs provide flexibility of using integer operations and thereby obtain speedup by the use of quantization.

4.2. ASICs

ASIC (Application Specific Integrated Circuit) implements the entire algorithm in hardware with minimal flexibility like modification of parameters coded via register settings. Google's Tensor Processing Unit (TPU) is an accel-

erator specifically designed by Google for its TensorFlow framework, which is extensively used for CNNs. It focuses on a high volume of 8-bit precision arithmetic. The first generation of TPU was focused on the inference, while the second generation increased capability to support neural network training also. Recently introduced by Intel Nervana, the NNP (Neural Network Processor) is a processor built specifically for neural networks. The NNP architecture optimizes memory bandwidth utilization, uses Flexpoint numerics and includes high speed serializer/deserializer which enable more than a terabit-per-second of bidirectional offchip bandwidth. The high bandwidth enables model parallelism in addition to data parallelism. However, ASIC architectures for deep learning is not highly preferred as the field is still evolving and hence, flexibility and programmability is preferred.

4.3. FPGA

FPGAs (Field-programmable gate arrays) are a natural choice for implementing neural networks because they can combine computing, logic and memory resources in a single device. FPGA's combine two processing regions, DSP and ALU logic. The unique flexibility of the FPGA allows the logic precision to be adjusted to the minimum that a particular network design requires. With the deep-learning field evolving and the network architectures changing frequently, having the flexibility to reprogram the hardware is very essential. FPGA's have been impractical for wide spread use in complex algorithmic-based systems due to the traditional low-level hardware programming environments. However, recently Intel FPGA SDK for OpenCL solves this problem, making FPGA useful for accelerating complex algorithms including CNNs. Intel has shown that using their OpenCL framework, Arria 10 FPGA can run Alexnet twice as fast compared to the Xeon CPU at one third power and 2X better performance to power ratio compared to Titan X GPU. For detailed benchmarks refer to whitepaper [45]. Optimization techniques such as quantization and pruning can be efficiently supported on the FPGAs.

4.4. SIMD type architectures

SIMD engines are quite popular in the application areas of image processing. This is because the input data is 8-bit fixed point and the initial stages of image processing are typically embarrassingly parallel. These processors are typically designed to be power-efficient by avoiding floating-point and having a simplified processing pipeline. For instance, performance/watt of TI's Embedded Vision Engine (EVE) [26] is $\sim$8X than that of A15. TI has efficiently mapped CNNs for performing real time semantic segmentation on EVE. For more details about the implementation, you can refer to [27]. Also, for performance boosts, some of the strategies like quantization and sparsity which we discussed earlier are extensively used to get a big boost in real time performance.

5. Results and discussion

Evaluating exhaustively all the previously described efficient designs and optimizations would be a tremendous work as most of the approaches are orthogonal and could be implemented jointly. To illustrate the concept of efficiency we propose to discuss using semantic segmentation for autonomous driving as a case study, focusing on a single HW and exploring several combinations of meta-architecture and feature extraction designs with different level of efficient optimizations.

5.1. Case Study: Semantic Segmentation Benchmarking

In this section, a comparative evaluation of different semantic segmentation architectures is presented. We present an evaluation of diverse kinds of networks, both in terms of accuracy and speed to highlight efficiency. Some efficient networks that has not been evaluated before on the semantic segmentation for automated driving are also presented. Other architectures such as DeepLab[4] achieving state of the art accuracy in segmentation are not included in the comparison since they are computationally inefficient. This can guide further decisions on what would best fit in the automated driving system. Evaluation metrics used are mean intersection over union(IoU) and per class IoU. The running time for inference is computed in seconds. The different architectures are evaluated on a GTX TITAN GPU with images of resolution 480x360. The comparison is shown in Table 1.

The architectures that are primarily evaluated are : (1) Unet [31] (2) Xception [5] which are classification networks that were not used in the segmentation problem before. (3) Dilated FCN16s, an architecture that was designed to be computationally and memory efficient with reasonable accuracy. (4) FCN8s [25]. (5) Segnet Basic [1]. (6) Dilation8 [41]. (7)Enet [29], which is the most efficient architecture for semantic segmentation. A unified framework with the first five architectures is going to be publicly available to help further research. While the results of the last two architectures are reported from their work. Note that the mean class IoU is computed over all classes, even the ones not included in the Table 1. But, only the classes of interest were the ones mentioned in Table 1.

Although Dilation8 outperforms all previous architectures in mean IoU it has the largest running time. This renders it as an inefficient solution to semantic segmentation for automated driving. However, the dilated convolution idea can be adapted in a shallower network. It uses dilated convolution to increase receptive field while maintaining the resolution of the segmentation. Dilated FCN16s

Table 1. Semantic Segmentation Results on CamVid. Running time in seconds, mean IoU, and perclass IoU is shown. Some of the 11 classes are shown due to limited space.

	Run-time (s)	Mean Class IoU	Per-Class IoU						
			Sky	Building	Road	Sidewalk	Vegetation	Car	Pedestrian
FCN16s Dilated	0.07	46.7	86.3	69.1	87.8	63.7	60.8	63.6	21.4
Xception[5]	**0.02**	42.8	81.9	68.9	86.6	62.9	61.6	60.8	19.8
Segnet-Basic[1]	0.03	46.4	87.0	68.7	86.2	60.5	52.0	58.5	25.3
FCN8s[25]	0.33	49.7	87.6	75.5	87.2	67.2	70.6	76.4	27.7
Unet[31]+BN[19]	0.56	53.9	90.2	72.6	89.1	67.2	67.7	74.7	34.1
Dilation8[41]	0.6474	**65.3**	89.9	**82.6**	92.2	75.3	76.2	**84.0**	56.3
Enet[29]	0.047	51.3	**95.1**	74.7	**95.1**	**86.7**	**77.8**	82.4	**67.2**

is an adapted version of FCN-16s as originally introduced in [25]. Two pooling layers are removed along with the convolutional layers in between them and conv4/conv5 layers are reduced to two dilated convolution layers with dilation factors of 2 and 4 respectively. This leads to a decrease in the size of the network and its running time for real-time applications. Another architecture used for medical image segmentation was experimented on CamVid which is called Unet. It turned to work second best on CamVid, but the running time is still not practical for real deployment. Xception [5] is an architecture that is mainly relying on depthwise separable convolution, that separates the spatial convolution from depthwise convolution. Although the network is designed for classification, it has been transformed to a fully convolutional network for the purpose of segmentation. The network mean IoU was much lower than other architectures, with a very small improvement in the running time against Segnet. Although Segnet is not considered as the state of the art in segmentation, but it turned out to provide a good balance between mean IoU and speed. In our experiments using batch normalization [19] turned to be effective in training both Segnet and Unet. It turned to converge faster, and it got better mean IoU of 47.3% in case of Segnet. It is worth noting that in case of FCN8s, we were able to reproduce similar results to the work in[7]. But this is less by 2% than what was reported in [20].

5.2. Design Exploration & Guidelines

Some good design choices that are accepted with-in the community are presented:
(1) The use of 3x3 convolutions similar to VGG architectures [34] turned to be useful experimentally, especially in scenarios where you care about the resolution of your input such as segmentation.
(2) The dilated convolution is considered to be the best practice in segmentation as it increases the receptive field without downgrading resolution.
(3) Batch normalization [19] turned to be a very useful trick for better convergence during training in our experiments due to the reduction of change in distribution of network activations.
(4) The resolution of the input image largely affects the segmentation, although it seems as a tiny detail. We found that higher input image resolution can help with segmenting small objects like pedestrian. Also, using random crops to help reduce the class imbalance can further help the segmentation.

6. Conclusion

We provided an overview of various optimizations for CNN implementation, de-constructing a wide array of CNN architectures into its constituent building blocks and discussed efficient design techniques in a taxonomic way. Then we detailed various hardware architectures focusing on how to construct an efficient CNN for the particular hardware. Finally, we performed a case study by benchmarking semantic segmentation for autonomous driving and shared our results. The case study showed that the efficient design techniques used were limited and there is a need to systematically explore and evaluate a combination of various efficient design techniques shared in the survey.

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

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

[3] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.

[5] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

[6] T. Cohen and M. Welling. Group equivariant convolutional networks. In *International Conference on Machine Learning*, pages 2990–2999, 2016.

[7] M. Fayyaz, M. H. Saffar, M. Sabokrou, M. Fathy, and R. Klette. STFCN: spatio-temporal FCN for semantic video segmentation. *CoRR*, abs/1608.05971, 2016.

[8] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, pages 243–254. IEEE Press, 2016.

[9] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[10] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[11] 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.

[12] G. Huang, S. Liu, L. van der Maaten, and K. Q. Weinberger. Condensenet: An efficient densenet using learned group convolutions. *arXiv preprint arXiv:1711.09224*, 2017.

[13] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017.

[14] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016.

[15] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

[16] F. Iandola and K. Keutzer. Keynote: Small neural nets are beautiful: Enabling embedded systems with small deep-neural-network architectures. *arXiv preprint arXiv:1710.02759*, 2017.

[17] 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.

[18] Y. Ioannou, D. Robertson, J. Shotton, R. Cipolla, and A. Criminisi. Training cnns with low-rank filters for efficient image classification. *arXiv preprint arXiv:1511.06744*, 2015.

[19] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[20] A. Kendall, V. Badrinarayanan, and R. Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *CoRR*, abs/1511.02680, 2015.

[21] K.-H. Kim, S. Hong, B. Roh, Y. Cheon, and M. Park. Pvanet: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021*, 2016.

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

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[24] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

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

[26] D. K. Mandal, J. Sankaran, A. Gupta, K. Castille, S. Gondkar, S. Kamath, P. Sundar, and A. Phipps. An embedded vision engine (eve) for automotive vision processing. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 49–52. IEEE, 2014.

[27] M. Mathew, K. Desappan, P. Kumar Swami, and S. Nagori. Sparse, quantized, full frame cnn for low power embedded devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[28] D. Mishkin, N. Sergievskiy, and J. Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 2017.

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

[30] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[31] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[32] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

[33] W. Shang, K. Sohn, D. Almeida, and H. Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *International Conference on Machine Learning*, pages 2217–2225, 2016.

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

[35] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[36] V. Sze, Y.-H. Chen, T.-J. Yang, and J. Emer. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint arXiv:1703.09039*, 2017.

[37] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[38] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[39] Z. Wu, C. Shen, and A. v. d. Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv preprint arXiv:1611.10080*, 2016.

[40] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995. IEEE, 2017.

[41] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[42] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[43] X. Zhang, Z. Li, C. C. Loy, and D. Lin. Polynet: A pursuit of structural diversity in very deep networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3900–3908. IEEE, 2017.

[44] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

[45] Intel altera's efficient neural networks. `https://www.altera.com/en_US/pdfs/literature/solution-sheets/efficient_neural_networks.pdf`, 2017 (accessed March 16, 2018).

[46] Tensorrt 8-bit inference. `http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit-inference-with-tensorrt.pdf`, 2017 (accessed March 16, 2018).

[47] Tiny darknet. `https://pjreddie.com/darknet/tiny-darknet/`, 2017 (accessed March 16, 2018).