# SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

**Forrest N. Iandola**[1], **Song Han**[2], **Matthew W. Moskewicz**[1], **Khalid Ashraf**[1],
**William J. Dally**[2], **Kurt Keutzer**[1]
[1]DeepScale* & UC Berkeley [2]Stanford University
{forresti, moskewcz, kashraf, keutzer}@eecs.berkeley.edu
{songhan, dally}@stanford.edu



## ABSTRACT

Recent research on deep convolutional neural networks (CNNs) has focused primarily on improving accuracy. For a given accuracy level, it is typically possible to identify multiple CNN architectures that achieve that accuracy level. With equivalent accuracy, smaller CNN architectures offer at least three advantages: (1) Smaller CNNs require less communication across servers during distributed training. (2) Smaller CNNs require less bandwidth to export a new model from the cloud to an autonomous car. (3) Smaller CNNs are more feasible to deploy on FPGAs and other hardware with limited memory. To provide all of these advantages, we propose a small CNN architecture called SqueezeNet. SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. Additionally, with model compression techniques, we are able to compress SqueezeNet to less than 0.5MB ($510\times$ smaller than AlexNet).

The SqueezeNet architecture is available for download here: https://github.com/DeepScale/SqueezeNet

## 1 INTRODUCTION AND MOTIVATION

Much of the recent research on deep convolutional neural networks (CNNs) has focused on increasing accuracy on computer vision datasets. For a given accuracy level, there typically exist multiple CNN architectures that achieve that accuracy level. Given equivalent accuracy, a CNN architecture with fewer parameters has several advantages:

- **More efficient distributed training.** Communication among servers is the limiting factor to the scalability of distributed CNN training. For distributed data-parallel training, communication overhead is directly proportional to the number of parameters in the model (Iandola et al., 2016). In short, small models train faster due to requiring less communication.
- **Less overhead when exporting new models to clients.** For autonomous driving, companies such as Tesla periodically copy new models from their servers to customers' cars. This practice is often referred to as an *over-the-air* update. Consumer Reports has found that the safety of Tesla's *Autopilot* semi-autonomous driving functionality has incrementally improved with recent over-the-air updates (Consumer Reports, 2016). However, over-the-air updates of today's typical CNN/DNN models can require large data transfers. With AlexNet, this would require 240MB of communication from the server to the car. Smaller models require less communication, making frequent updates more feasible.
- **Feasible FPGA and embedded deployment.** FPGAs often have less than 10MB[1] of on-chip memory and no off-chip memory or storage. For inference, a sufficiently small model could be stored directly on the FPGA instead of being bottlenecked by memory bandwidth (Qiu et al., 2016), while video frames stream through the FPGA in real time. Further, when deploying CNNs on Application-Specific Integrated Circuits (ASICs), a sufficiently small model could be stored directly on-chip, and smaller models may enable the ASIC to fit on a smaller die.

*http://deepscale.ai

[1]For example, the Xilinx Vertex-7 FPGA has a maximum of 8.5 MBytes (i.e. 68 Mbits) of on-chip memory and does not provide off-chip memory.

As you can see, there are several advantages of smaller CNN architectures. With this in mind, we focus directly on the problem of identifying a CNN architecture with fewer parameters but equivalent accuracy compared to a well-known model. We have discovered such an architecture, which we call *SqueezeNet*. In addition, we present our attempt at a more disciplined approach to searching the design space for novel CNN architectures.

The rest of the paper is organized as follows. In Section 2 we review the related work. Then, in Sections 3 and 4 we describe and evaluate the SqueezeNet architecture. After that, we turn our attention to understanding how CNN architectural design choices impact model size and accuracy. We gain this understanding by exploring the design space of SqueezeNet-like architectures. In Section 5, we do design space exploration on the *CNN microarchitecture*, which we define as the organization and dimensionality of individual layers and modules. In Section 6, we do design space exploration on the *CNN macroarchitecture*, which we define as high-level organization of layers in a CNN. Finally, we conclude in Section 7. In short, Sections 3 and 4 are useful for CNN researchers as well as practitioners who simply want to apply SqueezeNet to a new application. The remaining sections are aimed at advanced researchers who intend to design their own CNN architectures.

## 2 Related Work

### 2.1 Model Compression

The overarching goal of our work is to identify a model that has very few parameters while preserving accuracy. To address this problem, a sensible approach is to take an existing CNN model and compress it in a lossy fashion. In fact, a research community has emerged around the topic of *model compression*, and several approaches have been reported. A fairly straightforward approach by Denton *et al*. is to apply singular value decomposition (SVD) to a pretrained CNN model (Denton et al., 2014). Han *et al*. developed Network Pruning, which begins with a pretrained model, then replaces parameters that are below a certain threshold with zeros to form a sparse matrix, and finally performs a few iterations of training on the sparse CNN (Han et al., 2015b). Recently, Han *et al*. extended their work by combining Network Pruning with quantization (to 8 bits or less) and huffman encoding to create an approach called Deep Compression (Han et al., 2015a), and further designed a hardware accelerator called EIE (Han et al., 2016a) that operates directly on the compressed model, achieving substantial speedups and energy savings.

### 2.2 CNN Microarchitecture

Convolutions have been used in artificial neural networks for at least 25 years; LeCun *et al*. helped to popularize CNNs for digit recognition applications in the late 1980s (LeCun et al., 1989). In neural networks, convolution filters are typically 3D, with height, width, and channels as the key dimensions. When applied to images, CNN filters typically have 3 channels in their first layer (i.e. RGB), and in each subsequent layer $L_i$ the filters have the same number of channels as $L_{i-1}$ has filters. The early work by LeCun *et al*. (LeCun et al., 1989) uses 5x5xChannels[2] filters, and the recent VGG (Simonyan & Zisserman, 2014) architectures extensively use 3x3 filters. Models such as Network-in-Network (Lin et al., 2013) and the GoogLeNet family of architectures (Szegedy et al., 2014; Ioffe & Szegedy, 2015; Szegedy et al., 2015; 2016) use 1x1 filters in some layers.

With the trend of designing very deep CNNs, it becomes cumbersome to manually select filter dimensions for each layer. To address this, various higher level building blocks, or *modules*, comprised of multiple convolution layers with a specific fixed organization have been proposed. For example, the GoogLeNet papers propose *Inception modules*, which are comprised of a number of different dimensionalities of filters, usually including 1x1 and 3x3, plus sometimes 5x5 (Szegedy et al., 2014) and sometimes 1x3 and 3x1 (Szegedy et al., 2015). Many such modules are then combined, perhaps with additional *ad-hoc* layers, to form a complete network. We use the term *CNN microarchitecture* to refer to the particular organization and dimensions of the individual modules.

### 2.3 CNN Macroarchitecture

While the CNN microarchitecture refers to individual layers and modules, we define the *CNN macroarchitecture* as the system-level organization of multiple modules into an end-to-end CNN architecture.

[2]From now on, we will simply abbreviate HxWxChannels to HxW.

Perhaps the mostly widely studied CNN macroarchitecture topic in the recent literature is the impact of *depth* (i.e. number of layers) in networks. Simoyan and Zisserman proposed the VGG (Simonyan & Zisserman, 2014) family of CNNs with 12 to 19 layers and reported that deeper networks produce higher accuracy on the ImageNet-1k dataset (Deng et al., 2009). K. He *et al*. proposed deeper CNNs with up to 30 layers that deliver even higher ImageNet accuracy (He et al., 2015a).

The choice of connections across multiple layers or modules is an emerging area of CNN macroarchitectural research. Residual Networks (ResNet) (He et al., 2015b) and Highway Networks (Srivastava et al., 2015) each propose the use of connections that skip over multiple layers, for example additively connecting the activations from layer 3 to the activations from layer 6. We refer to these connections as *bypass connections*. The authors of ResNet provide an A/B comparison of a 34-layer CNN with and without bypass connections; adding bypass connections delivers a 2 percentage-point improvement on Top-5 ImageNet accuracy.

### 2.4 Neural Network Design Space Exploration

Neural networks (including deep and convolutional NNs) have a large design space, with numerous options for microarchitectures, macroarchitectures, solvers, and other hyperparameters. It seems natural that the community would want to gain intuition about how these factors impact a NN's accuracy (i.e. the *shape* of the design space). Much of the work on design space exploration (DSE) of NNs has focused on developing automated approaches for finding NN architectures that deliver higher accuracy. These automated DSE approaches include bayesian optimization (Snoek et al., 2012), simulated annealing (Ludermir et al., 2006), randomized search (Bergstra & Bengio, 2012), and genetic algorithms (Stanley & Miikkulainen, 2002). To their credit, each of these papers provides a case in which the proposed DSE approach produces a NN architecture that achieves higher accuracy compared to a representative baseline. However, these papers make no attempt to provide intuition about the shape of the NN design space. Later in this paper, we eschew automated approaches – instead, we refactor CNNs in such a way that we can do principled A/B comparisons to investigate how CNN architectural decisions influence model size and accuracy.

In the following sections, we first propose and evaluate the SqueezeNet architecture with and without model compression. Then, we explore the impact of design choices in microarchitecture and macroarchitecture for SqueezeNet-like CNN architectures.

## 3 SqueezeNet: preserving accuracy with few parameters

In this section, we begin by outlining our design strategies for CNN architectures with few parameters. Then, we introduce the *Fire module*, our new building block out of which to build CNN architectures. Finally, we use our design strategies to construct *SqueezeNet*, which is comprised mainly of Fire modules.

### 3.1 Architectural Design Strategies

Our overarching objective in this paper is to identify CNN architectures that have few parameters while maintaining competitive accuracy. To achieve this, we employ three main strategies when designing CNN architectures:

***Strategy 1.*** **Replace 3x3 filters with 1x1 filters.** Given a budget of a certain number of convolution filters, we will choose to make the majority of these filters 1x1, since a 1x1 filter has 9X fewer parameters than a 3x3 filter.

***Strategy 2.*** **Decrease the number of input channels to 3x3 filters.** Consider a convolution layer that is comprised entirely of 3x3 filters. The total quantity of parameters in this layer is (number of input channels) * (number of filters) * (3*3). So, to maintain a small total number of parameters in a CNN, it is important not only to decrease the number of 3x3 filters (see Strategy 1 above), but also to decrease the number of *input channels* to the 3x3 filters. We decrease the number of input channels to 3x3 filters using *squeeze layers*, which we describe in the next section.

***Strategy 3.*** **Downsample late in the network so that convolution layers have large activation maps.** In a convolutional network, each convolution layer produces an output activation map with a spatial resolution that is at least 1x1 and often much larger than 1x1. The height and width of these activation maps are controlled by: (1) the size of the input data (e.g. 256x256 images) and (2)

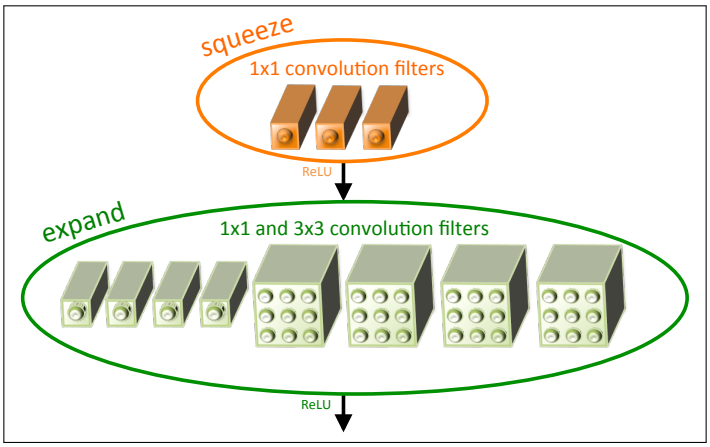



Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

the choice of layers in which to downsample in the CNN architecture. Most commonly, downsampling is engineered into CNN architectures by setting the (stride $> 1$) in some of the convolution or pooling layers (e.g. (Szegedy et al., 2014; Simonyan & Zisserman, 2014; Krizhevsky et al., 2012)). If early[3] layers in the network have large strides, then most layers will have small activation maps. Conversely, if most layers in the network have a stride of 1, and the strides greater than 1 are concentrated toward the end[4] of the network, then many layers in the network will have large activation maps. Our intuition is that large activation maps (due to delayed downsampling) can lead to higher classification accuracy, with all else held equal. Indeed, K. He and H. Sun applied delayed downsampling to four different CNN architectures, and in each case delayed downsampling led to higher classification accuracy (He & Sun, 2015).

Strategies 1 and 2 are about judiciously decreasing the quantity of parameters in a CNN while attempting to preserve accuracy. Strategy 3 is about maximizing accuracy on a limited budget of parameters. Next, we describe the Fire module, which is our building block for CNN architectures that enables us to successfully employ Strategies 1, 2, and 3.

### 3.2 The Fire Module

We define the Fire module as follows. A Fire module is comprised of: a *squeeze* convolution layer (which has only 1x1 filters), feeding into an *expand* layer that has a mix of 1x1 and 3x3 convolution filters; we illustrate this in Figure 1. The liberal use of 1x1 filters in Fire modules is an application of Strategy 1 from Section 3.1. We expose three tunable dimensions (hyperparameters) in a Fire module: $s_{1x1}$, $e_{1x1}$, and $e_{3x3}$. In a Fire module, $s_{1x1}$ is the number of filters in the squeeze layer (all 1x1), $e_{1x1}$ is the number of 1x1 filters in the expand layer, and $e_{3x3}$ is the number of 3x3 filters in the expand layer. When we use Fire modules we set $s_{1x1}$ to be less than ($e_{1x1}$ + $e_{3x3}$), so the squeeze layer helps to limit the number of input channels to the 3x3 filters, as per Strategy 2 from Section 3.1.

### 3.3 The SqueezeNet architecture

We now describe the SqueezeNet CNN architecture. We illustrate in Figure 2 that SqueezeNet begins with a standalone convolution layer (conv1), followed by 8 Fire modules (fire2-9), ending with a final conv layer (conv10). We gradually increase the number of filters per fire module from the beginning to the end of the network. SqueezeNet performs max-pooling with a stride of 2 after layers conv1, fire4, fire8, and conv10; these relatively late placements of pooling are per Strategy 3 from Section 3.1. We present the full SqueezeNet architecture in Table 1.

[3]In our terminology, an "early" layer is close to the input data.

[4]In our terminology, the "end" of the network is the classifier.

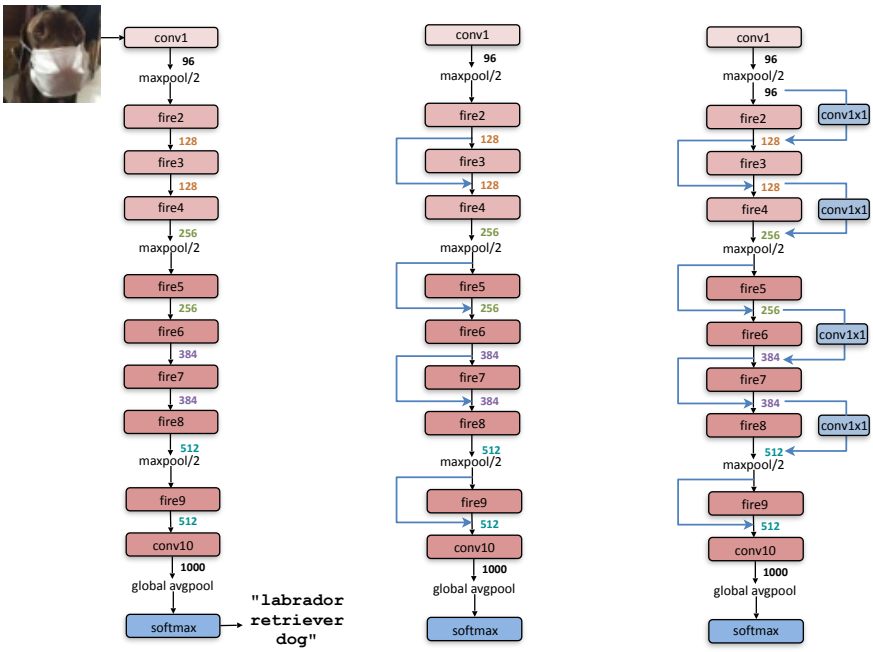



Figure 2: Macroarchitectural view of our SqueezeNet architecture. Left: SqueezeNet (Section 3.3); Middle: SqueezeNet with simple bypass (Section 6); Right: SqueezeNet with complex bypass (Section 6).

#### 3.3.1 Other SqueezeNet details

For brevity, we have omitted number of details and design choices about SqueezeNet from Table 1 and Figure 2. We provide these design choices in the following. The intuition behind these choices may be found in the papers cited below.

- So that the output activations from 1x1 and 3x3 filters have the same height and width, we add a 1-pixel border of zero-padding in the input data to 3x3 filters of expand modules.
- ReLU (Nair & Hinton, 2010) is applied to activations from squeeze and expand layers.
- Dropout (Srivastava et al., 2014) with a ratio of 50% is applied after the fire9 module.
- Note the lack of fully-connected layers in SqueezeNet; this design choice was inspired by the NiN (Lin et al., 2013) architecture.
- When training SqueezeNet, we begin with a learning rate of 0.04, and we linearly decrease the learning rate throughout training, as described in (Mishkin et al., 2016). For details on the training protocol (e.g. batch size, learning rate, parameter initialization), please refer to our Caffe-compatible configuration files located here: https://github.com/DeepScale/SqueezeNet.
- The Caffe framework does not natively support a convolution layer that contains multiple filter resolutions (e.g. 1x1 and 3x3) (Jia et al., 2014). To get around this, we implement our expand layer with two separate convolution layers: a layer with 1x1 filters, and a layer with 3x3 filters. Then, we concatenate the outputs of these layers together in the channel dimension. This is numerically equivalent to implementing one layer that contains both 1x1 and 3x3 filters.

We released the SqueezeNet configuration files in the format defined by the Caffe CNN framework. However, in addition to Caffe, several other CNN frameworks have emerged, including MXNet (Chen et al., 2015a), Chainer (Tokui et al., 2015), Keras (Chollet, 2016), and Torch (Collobert et al., 2011). Each of these has its own native format for representing a CNN architecture. That said, most of these libraries use the same underlying computational back-ends such as cuDNN (Chetlur et al., 2014) and MKL-DNN (Das et al., 2016). The research community has

ported the SqueezeNet CNN architecture for compatibility with a number of other CNN software frameworks:

- MXNet (Chen et al., 2015a) port of SqueezeNet: (Haria, 2016)
- Chainer (Tokui et al., 2015) port of SqueezeNet: (Bell, 2016)
- Keras (Chollet, 2016) port of SqueezeNet: (DT42, 2016)
- Torch (Collobert et al., 2011) port of SqueezeNet's Fire Modules: (Waghmare, 2016)

## 4 EVALUATION OF SQUEEZENET

We now turn our attention to evaluating SqueezeNet. In each of the CNN model compression papers reviewed in Section 2.1, the goal was to compress an AlexNet (Krizhevsky et al., 2012) model that was trained to classify images using the ImageNet (Deng et al., 2009) (ILSVRC 2012) dataset. Therefore, we use AlexNet[5] and the associated model compression results as a basis for comparison when evaluating SqueezeNet.

Table 1: SqueezeNet architectural dimensions. (The formatting of this table was inspired by the Inception2 paper (Ioffe & Szegedy, 2015).)

| layer name/type | output size | filter size / stride (if not a fire layer) | depth | $s_{1x1}$ (#1x1 squeeze) | $e_{1x1}$ (#1x1 expand) | $e_{3x3}$ (#3x3 expand) | $s_{1x1}$ sparsity | $e_{1x1}$ sparsity | $e_{3x3}$ sparsity | # bits | #parameter before pruning | #parameter after pruning |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| input image | 224x224x3 | | | | | | | | | | - | - |
| conv1 | 111x111x96 | 7x7/2 (x96) | 1 | | | | 100% (7x7) | | | 6bit | 14,208 | 14,208 |
| maxpool1 | 55x55x96 | 3x3/2 | 0 | | | | | | | | | |
| fire2 | 55x55x128 | | 2 | 16 | 64 | 64 | 100% | 100% | **33%** | 6bit | 11,920 | 5,746 |
| fire3 | 55x55x128 | | 2 | 16 | 64 | 64 | 100% | 100% | **33%** | 6bit | 12,432 | 6,258 |
| fire4 | 55x55x256 | | 2 | 32 | 128 | 128 | 100% | 100% | **33%** | 6bit | 45,344 | 20,646 |
| maxpool4 | 27x27x256 | 3x3/2 | 0 | | | | | | | | | |
| fire5 | 27x27x256 | | 2 | 32 | 128 | 128 | 100% | 100% | **33%** | 6bit | 49,440 | 24,742 |
| fire6 | 27x27x384 | | 2 | 48 | 192 | 192 | 100% | **50%** | **33%** | 6bit | 104,880 | 44,700 |
| fire7 | 27x27x384 | | 2 | 48 | 192 | 192 | **50%** | 100% | **33%** | 6bit | 111,024 | 46,236 |
| fire8 | 27x27x512 | | 2 | 64 | 256 | 256 | 100% | **50%** | **33%** | 6bit | 188,992 | 77,581 |
| maxpool8 | 13x12x512 | 3x3/2 | 0 | | | | | | | | | |
| fire9 | 13x13x512 | | 2 | 64 | 256 | 256 | **50%** | 100% | **30%** | 6bit | 197,184 | 77,581 |
| conv10 | 13x13x1000 | 1x1/1 (x1000) | 1 | | | | **20%** (3x3) | | | 6bit | 513,000 | 103,400 |
| avgpool10 | 1x1x1000 | 13x13/1 | 0 | | | | | | | | | |
| | activations | | | parameters | | | | compression info | | | 1,248,424 (total) | **421,098** (total) |

In Table 2, we review SqueezeNet in the context of recent model compression results. The SVD-based approach is able to compress a pretrained AlexNet model by a factor of 5x, while diminishing top-1 accuracy to 56.0% (Denton et al., 2014). Network Pruning achieves a 9x reduction in model size while maintaining the baseline of 57.2% top-1 and 80.3% top-5 accuracy on ImageNet (Han et al., 2015b). Deep Compression achieves a 35x reduction in model size while still maintaining the baseline accuracy level (Han et al., 2015a). Now, with SqueezeNet, we achieve a 50X reduction in model size compared to AlexNet, while **meeting or exceeding the top-1 and top-5 accuracy of AlexNet.** We summarize all of the aforementioned results in Table 2.

It appears that we have surpassed the state-of-the-art results from the model compression community: even when using uncompressed 32-bit values to represent the model, SqueezeNet has a $1.4\times$ smaller model size than the best efforts from the model compression community while maintaining or exceeding the baseline accuracy. Until now, an open question has been: *are small models amenable to compression, or do small models "need" all of the representational power afforded by dense floating-point values?* To find out, we applied Deep Compression (Han et al., 2015a)

[5]Our baseline is `bvlc_alexnet` from the Caffe codebase (Jia et al., 2014).

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

| CNN architecture | Compression Approach | Data Type | Original → Compressed Model Size | Reduction in Model Size vs. AlexNet | Top-1 ImageNet Accuracy | Top-5 ImageNet Accuracy |
|---|---|---|---|---|---|---|
| AlexNet | None (baseline) | 32 bit | 240MB | 1x | 57.2% | 80.3% |
| AlexNet | SVD (Denton et al., 2014) | 32 bit | 240MB → 48MB | 5x | 56.0% | 79.4% |
| AlexNet | Network Pruning (Han et al., 2015b) | 32 bit | 240MB → 27MB | 9x | 57.2% | 80.3% |
| AlexNet | Deep Compression (Han et al., 2015a) | 5-8 bit | 240MB → 6.9MB | 35x | 57.2% | 80.3% |
| SqueezeNet (ours) | None | 32 bit | 4.8MB | **50x** | 57.5% | 80.3% |
| SqueezeNet (ours) | Deep Compression | 8 bit | 4.8MB → 0.66MB | **363x** | 57.5% | 80.3% |
| SqueezeNet (ours) | Deep Compression | 6 bit | 4.8MB → 0.47MB | **510x** | 57.5% | 80.3% |

to SqueezeNet, using 33% sparsity[6] and 8-bit quantization. This yields a 0.66 MB model (363× smaller than 32-bit AlexNet) with equivalent accuracy to AlexNet. Further, applying Deep Compression with 6-bit quantization and 33% sparsity on SqueezeNet, we produce a 0.47MB model (510× smaller than 32-bit AlexNet) with equivalent accuracy. **Our small model is indeed amenable to compression.**

In addition, these results demonstrate that Deep Compression (Han et al., 2015a) not only works well on CNN architectures with many parameters (e.g. AlexNet and VGG), but it is also able to compress the already compact, fully convolutional SqueezeNet architecture. Deep Compression compressed SqueezeNet by $10\times$ while preserving the baseline accuracy. In summary: by combining CNN architectural innovation (SqueezeNet) with state-of-the-art compression techniques (Deep Compression), we achieved a $510\times$ reduction in model size with no decrease in accuracy compared to the baseline.

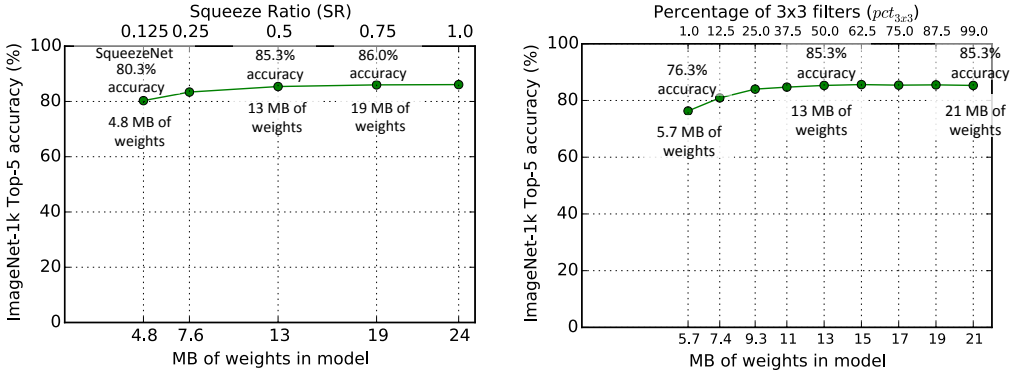
Finally, note that Deep Compression (Han et al., 2015b) uses a *codebook* as part of its scheme for quantizing CNN parameters to 6- or 8-bits of precision. Therefore, on most commodity processors, it is *not* trivial to achieve a speedup of $\frac{32}{8} = 4x$ with 8-bit quantization or $\frac{32}{6} = 5.3x$ with 6-bit quantization using the scheme developed in Deep Compression. However, Han *et al*. developed custom hardware – *Efficient Inference Engine (EIE)* – that can compute codebook-quantized CNNs more efficiently (Han et al., 2016a). In addition, in the months since we released SqueezeNet, P. Gysel developed a strategy called *Ristretto* for linearly quantizing SqueezeNet to 8 bits (Gysel, 2016). Specifically, Ristretto does computation in 8 bits, and it stores parameters and activations in 8-bit data types. Using the Ristretto strategy for 8-bit computation in SqueezeNet inference, Gysel observed less than 1 percentage-point of drop in accuracy when using 8-bit instead of 32-bit data types.

## 5 CNN Microarchitecture Design Space Exploration

So far, we have proposed architectural design strategies for small models, followed these principles to create SqueezeNet, and discovered that SqueezeNet is 50x smaller than AlexNet with equivalent accuracy. However, SqueezeNet and other models reside in a broad and largely unexplored design space of CNN architectures. Now, in Sections 5 and 6, we explore several aspects of the design space. We divide this architectural exploration into two main topics: *microarchitectural exploration* (per-module layer dimensions and configurations) and *macroarchitectural exploration* (high-level end-to-end organization of modules and other layers).

In this section, we design and execute experiments with the goal of providing intuition about the shape of the microarchitectural design space with respect to the design strategies that we proposed in Section 3.1. Note that our goal here is *not* to maximize accuracy in every experiment, but rather to understand the impact of CNN architectural choices on model size and accuracy.

[6]Note that, due to the storage overhead of storing sparse matrix indices, 33% sparsity leads to somewhat less than a $3\times$ decrease in model size.




(a) Exploring the impact of the squeeze ratio ($SR$) on model size and accuracy.

(b) Exploring the impact of the ratio of 3x3 filters in expand layers ($pct_{3x3}$) on model size and accuracy.

Figure 3: Microarchitectural design space exploration.

### 5.1 CNN MICROARCHITECTURE METAPARAMETERS

In SqueezeNet, each Fire module has three dimensional hyperparameters that we defined in Section 3.2: $s_{1x1}$, $e_{1x1}$, and $e_{3x3}$. SqueezeNet has 8 Fire modules with a total of 24 dimensional hyperparameters. To do broad sweeps of the design space of SqueezeNet-like architectures, we define the following set of higher level *metaparameters* which control the dimensions of all Fire modules in a CNN. We define $base_e$ as the number of *expand* filters in the first Fire module in a CNN. After every $freq$ Fire modules, we increase the number of expand filters by $incr_e$. In other words, for Fire module $i$, the number of expand filters is $e_i = base_e + (incr_e * \left\lfloor \frac{i}{freq} \right\rfloor)$. In the expand layer of a Fire module, some filters are 1x1 and some are 3x3; we define $e_i = e_{i,1x1} + e_{i,3x3}$ with $pct_{3x3}$ (in the range $[0, 1]$, shared over all Fire modules) as the percentage of expand filters that are 3x3. In other words, $e_{i,3x3} = e_i * pct_{3x3}$, and $e_{i,1x1} = e_i * (1 - pct_{3x3})$. Finally, we define the number of filters in the squeeze layer of a Fire module using a metaparameter called the *squeeze ratio (SR)* (again, in the range $[0, 1]$, shared by all Fire modules): $s_{i,1x1} = SR * e_i$ (or equivalently $s_{i,1x1} = SR * (e_{i,1x1} + e_{i,3x3})$). SqueezeNet (Table 1) is an example architecture that we generated with the aforementioned set of metaparameters. Specifically, SqueezeNet has the following metaparameters: $base_e = 128$, $incr_e = 128$, $pct_{3x3} = 0.5$, $freq = 2$, and $SR = 0.125$.

### 5.2 SQUEEZE RATIO

In Section 3.1, we proposed decreasing the number of parameters by using *squeeze layers* to decrease the number of input channels seen by 3x3 filters. We defined the *squeeze ratio (SR)* as the ratio between the number of filters in *squeeze* layers and the number of filters in *expand* layers. We now design an experiment to investigate the effect of the squeeze ratio on model size and accuracy.

In these experiments, we use SqueezeNet (Figure 2) as a starting point. As in SqueezeNet, these experiments use the following metaparameters: $base_e = 128$, $incr_e = 128$, $pct_{3x3} = 0.5$, and $freq = 2$. We train multiple models, where each model has a different squeeze ratio (SR)[7] in the range [0.125, 1.0]. In Figure 3(a), we show the results of this experiment, where each point on the graph is an independent model that was trained from scratch. SqueezeNet is the SR=0.125 point in this figure.[8] From this figure, we learn that increasing SR beyond 0.125 can further increase ImageNet top-5 accuracy from 80.3% (i.e. AlexNet-level) with a 4.8MB model to 86.0% with a 19MB model. Accuracy plateaus at 86.0% with SR=0.75 (a 19MB model), and setting SR=1.0 further increases model size without improving accuracy.

### 5.3 TRADING OFF 1X1 AND 3X3 FILTERS

In Section 3.1, we proposed decreasing the number of parameters in a CNN by replacing some 3x3 filters with 1x1 filters. An open question is, *how important is spatial resolution in CNN filters?*

[7]Note that, for a given model, all Fire layers share the same squeeze ratio.

[8]Note that we named it *Squeeze*Net because it has a low squeeze ratio (SR). That is, the squeeze layers in SqueezeNet have 0.125x the number of filters as the expand layers.

The VGG (Simonyan & Zisserman, 2014) architectures have 3x3 spatial resolution in most layers' filters; GoogLeNet (Szegedy et al., 2014) and Network-in-Network (NiN) (Lin et al., 2013) have 1x1 filters in some layers. In GoogLeNet and NiN, the authors simply propose a specific quantity of 1x1 and 3x3 filters without further analysis.[9] Here, we attempt to shed light on how the proportion of 1x1 and 3x3 filters affects model size and accuracy.

We use the following metaparameters in this experiment: $base_e = incr_e = 128$, $freq = 2$, $SR = 0.500$, and we vary $pct_{3x3}$ from 1% to 99%. In other words, each Fire module's expand layer has a predefined number of filters partitioned between 1x1 and 3x3, and here we turn the knob on these filters from "mostly 1x1" to "mostly 3x3". As in the previous experiment, these models have 8 Fire modules, following the same organization of layers as in Figure 2. We show the results of this experiment in Figure 3(b). Note that the 13MB models in Figure 3(a) and Figure 3(b) are the same architecture: $SR = 0.500$ and $pct_{3x3} = 50\%$. We see in Figure 3(b) that the top-5 accuracy plateaus at 85.6% using 50% 3x3 filters, and further increasing the percentage of 3x3 filters leads to a larger model size but provides no improvement in accuracy on ImageNet.

## 6 CNN MACROARCHITECTURE DESIGN SPACE EXPLORATION

So far we have explored the design space at the microarchitecture level, i.e. the contents of individual modules of the CNN. Now, we explore design decisions at the macroarchitecture level concerning the high-level connections among Fire modules. Inspired by ResNet (He et al., 2015b), we explored three different architectures:

- Vanilla SqueezeNet (as per the prior sections).
- SqueezeNet with simple bypass connections between some Fire modules. (Inspired by (Srivastava et al., 2015; He et al., 2015b).)
- SqueezeNet with complex bypass connections between the remaining Fire modules.

We illustrate these three variants of SqueezeNet in Figure 2.

Our *simple bypass* architecture adds bypass connections around Fire modules 3, 5, 7, and 9, requiring these modules to learn a residual function between input and output. As in ResNet, to implement a bypass connection around Fire3, we set the input to Fire4 equal to (output of Fire2 + output of Fire3), where the + operator is elementwise addition. This changes the regularization applied to the parameters of these Fire modules, and, as per ResNet, can improve the final accuracy and/or ability to train the full model.

One limitation is that, in the straightforward case, the number of input channels and number of output channels has to be the same; as a result, only half of the Fire modules can have simple bypass connections, as shown in the middle diagram of Fig 2. When the "same number of channels" requirement can't be met, we use a *complex bypass* connection, as illustrated on the right of Figure 2. While a simple bypass is "just a wire," we define a complex bypass as a bypass that includes a 1x1 convolution layer with the number of filters set equal to the number of output channels that are needed. Note that complex bypass connections add extra parameters to the model, while simple bypass connections do not.

In addition to changing the regularization, it is intuitive to us that adding bypass connections would help to alleviate the representational bottleneck introduced by squeeze layers. In SqueezeNet, the squeeze ratio (SR) is 0.125, meaning that every squeeze layer has 8x fewer output channels than the accompanying expand layer. Due to this severe dimensionality reduction, a limited amount of information can pass through squeeze layers. However, by adding bypass connections to SqueezeNet, we open up avenues for information to flow *around* the squeeze layers.

We trained SqueezeNet with the three macroarchitectures in Figure 2 and compared the accuracy and model size in Table 3. We fixed the microarchitecture to match SqueezeNet as described in Table 1 throughout the macroarchitecture exploration. Complex and simple bypass connections both yielded an accuracy improvement over the vanilla SqueezeNet architecture. Interestingly, the simple bypass enabled a higher accuracy accuracy improvement than complex bypass. Adding the

[9] To be clear, each filter is 1x1xChannels or 3x3xChannels, which we abbreviate to 1x1 and 3x3.

Table 3: SqueezeNet accuracy and model size using different macroarchitecture configurations

| Architecture | Top-1 Accuracy | Top-5 Accuracy | Model Size |
|---|---|---|---|
| Vanilla SqueezeNet | 57.5% | 80.3% | 4.8MB |
| SqueezeNet + Simple Bypass | **60.4%** | **82.5%** | 4.8MB |
| SqueezeNet + Complex Bypass | 58.8% | 82.0% | 7.7MB |

simple bypass connections yielded an increase of 2.9 percentage-points in top-1 accuracy and 2.2 percentage-points in top-5 accuracy without increasing model size.

## 7 CONCLUSIONS

In this paper, we have proposed steps toward a more disciplined approach to the design-space exploration of convolutional neural networks. Toward this goal we have presented SqueezeNet, a CNN architecture that has $50\times$ fewer parameters than AlexNet and maintains AlexNet-level accuracy on ImageNet. We also compressed SqueezeNet to less than 0.5MB, or $510\times$ smaller than AlexNet without compression. Since we released this paper as a technical report in 2016, Song Han and his collaborators have experimented further with SqueezeNet and model compression. Using a new approach called *Dense-Sparse-Dense (DSD)* (Han et al., 2016b), Han *et al*. use model compression during training as a regularizer to further improve accuracy, producing a compressed set of SqueezeNet parameters that is 1.2 percentage-points more accurate on ImageNet-1k, and also producing an uncompressed set of SqueezeNet parameters that is 4.3 percentage-points more accurate, compared to our results in Table 2.

We mentioned near the beginning of this paper that small models are more amenable to on-chip implementations on FPGAs. Since we released the SqueezeNet model, Gschwend has developed a variant of SqueezeNet and implemented it on an FPGA (Gschwend, 2016). As we anticipated, Gschwend was able to able to store the parameters of a SqueezeNet-like model entirely within the FPGA and eliminate the need for off-chip memory accesses to load model parameters.

In the context of this paper, we focused on ImageNet as a target dataset. However, it has become common practice to apply ImageNet-trained CNN representations to a variety of applications such as fine-grained object recognition (Zhang et al., 2013; Donahue et al., 2013), logo identification in images (Iandola et al., 2015), and generating sentences about images (Fang et al., 2015). ImageNet-trained CNNs have also been applied to a number of applications pertaining to autonomous driving, including pedestrian and vehicle detection in images (Iandola et al., 2014; Girshick et al., 2015; Ashraf et al., 2016) and videos (Chen et al., 2015b), as well as segmenting the shape of the road (Badrinarayanan et al., 2015). We think SqueezeNet will be a good candidate CNN architecture for a variety of applications, especially those in which small model size is of importance.

SqueezeNet is one of several new CNNs that we have discovered while broadly exploring the design space of CNN architectures. We hope that SqueezeNet will inspire the reader to consider and explore the broad range of possibilities in the design space of CNN architectures and to perform that exploration in a more systematic manner.

## REFERENCES

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

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *arxiv:1511.00561*, 2015.

Eddie Bell. A implementation of squeezenet in chainer. `https://github.com/ejlb/squeezenet-chainer`, 2016.

J. Bergstra and Y. Bengio. An optimization methodology for neural network weights and architectures. *JMLR*, 2012.

Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv:1512.01274*, 2015a.

Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, 2015b.

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cuDNN: efficient primitives for deep learning. *arXiv:1410.0759*, 2014.

Francois Chollet. Keras: Deep learning library for theano and tensorflow. `https://keras.io`, 2016.

Ronan Collobert, Koray Kavukcuoglu, and Clement Farabet. Torch7: A matlab-like environment for machine learning. In *NIPS BigLearn Workshop*, 2011.

Consumer Reports. Teslas new autopilot: Better but still needs improvement. `http://www.consumerreports.org/tesla/tesla-new-autopilot-better-but-needs-improvement`, 2016.

Dipankar Das, Sasikanth Avancha, Dheevatsa Mudigere, Karthikeyan Vaidyanathan, Srinivas Sridharan, Dhiraj D. Kalamkar, Bharat Kaul, and Pradeep Dubey. Distributed deep learning using synchronous stochastic gradient descent. *arXiv:1602.06709*, 2016.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

E.L Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv:1310.1531*, 2013.

DT42. Squeezenet keras implementation. `https://github.com/DT42/squeezenet_demo`, 2016.

Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh Srivastava, Li Deng, Piotr Dollar, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C. Platt, C. Lawrence Zitnick, and Geoffrey Zweig. From captions to visual concepts and back. In *CVPR*, 2015.

Ross B. Girshick, Forrest N. Iandola, Trevor Darrell, and Jitendra Malik. Deformable part models are convolutional neural networks. In *CVPR*, 2015.

David Gschwend. Zynqnet: An fpga-accelerated embedded convolutional neural network. Master's thesis, Swiss Federal Institute of Technology Zurich (ETH-Zurich), 2016.

Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *arXiv:1605.06402*, 2016.

S. Han, H. Mao, and W. Dally. Deep compression: Compressing DNNs with pruning, trained quantization and huffman coding. *arxiv:1510.00149v3*, 2015a.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural networks. In *NIPS*, 2015b.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: Efficient inference engine on compressed deep neural network. *International Symposium on Computer Architecture (ISCA)*, 2016a.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Shijian Tang, Erich Elsen, Bryan Catanzaro, John Tran, and William J. Dally. Dsd: Regularizing deep neural networks with dense-sparse-dense training flow. *arXiv:1607.04381*, 2016b.

Guo Haria. convert squeezenet to mxnet. `https://github.com/haria/SqueezeNet/commit/0cf57539375fd5429275af36fc94c774503427c3`, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015a.

Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015b.

Forrest N. Iandola, Matthew W. Moskewicz, Sergey Karayev, Ross B. Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv:1404.1869*, 2014.

Forrest N. Iandola, Anting Shen, Peter Gao, and Kurt Keutzer. DeepLogo: Hitting logo recognition with the deep neural network hammer. *arXiv:1510.02131*, 2015.

Forrest N. Iandola, Khalid Ashraf, Matthew W. Moskewicz, and Kurt Keutzer. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. In *CVPR*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *JMLR*, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

Y. LeCun, B.Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1989.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv:1312.4400*, 2013.

T.B. Ludermir, A. Yamazaki, and C. Zanchettin. An optimization methodology for neural network weights and architectures. *IEEE Trans. Neural Networks*, 2006.

Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of cnn advances on the imagenet. *arXiv:1606.02228*, 2016.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going deeper with embedded fpga platform for convolutional neural network. In *ACM International Symposium on FPGA*, 2016.

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

J. Snoek, H. Larochelle, and R.P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 2014.

R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. In *ICML Deep Learning Workshop*, 2015.

K.O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Neurocomputing*, 2002.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv:1409.4842*, 2014.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv:1512.00567*, 2015.

Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv:1602.07261*, 2016.

S. Tokui, K. Oono, S. Hido, and J. Clayton. Chainer: a next-generation open source framework for deep learning. In *NIPS Workshop on Machine Learning Systems (LearningSys)*, 2015.

Sagar M Waghmare. FireModule.lua. `https://github.com/Element-Research/dpnn/blob/master/FireModule.lua`, 2016.

Ning Zhang, Ryan Farrell, Forrest Iandola, and Trevor Darrell. Deformable part descriptors for fine-grained recognition and attribute prediction. In *ICCV*, 2013.