COMP 551 Final Project
"Size isn't everything, it's what you (can) do with it that counts": a comparison of SqueezeNet and AlexNet

Xavier Morin Duchesne 260760216
Bahare Samadi 260556423
Gordon Krieger 119005244

April 17, 2019

Abstract

We examine the deep learning neural network model SqueezeNet. Improvement in deep learning networks sometimes comes at the expense of size; increasing the depth or width of a network can often give an increase in performance. SqueezeNet is an attempt to move in the other direction: produce a small network that optimizes for size, while keeping an acceptable baseline accuracy. We confirm the main claim of the paper of having AlexNet-level accuracy. We also perform multiple ablation studies of SqueezeNet's architecture: our main finding is that adding a batch normalization layer to SqueezeNet's Fire modules gives a measurable increase in performance.

1 Introduction

The aim of this study is to investigate the neural network model SqueezeNet, as presented in Iandola et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size" [1].

The main motivation behind SqueezeNet is to create a neural network design with a small model size. The advantages of small models are made clear in the paper itself: among other things, smaller models require less memory to store, and less bandwidth to transmit. A model with total size of less than a megabyte, as the authors claim for their model, would lend itself to storage on a chip for embedded use. (Two of SqueezeNet's authors, Forrest Iandola and Kurt Keutzer, are founders of a company applying deep learning to the auto industry.) SqueezeNet has since been applied to various learning domains, such as pedestrian detection [2], video classification [3], medical imaging [4, 5] and internet of things (IoT) [6].

We verify the main claim of the paper, that SqueezeNet can meet the same level of accuracy as AlexNet [7] for some visual classification tasks. We trained both models from scratch, although time and computational constraints meant using a smaller dataset than the ImageNet Large Scale Visual Recognition Challenge [8] dataset used in the SqueezeNet paper. We opted instead to use the CIFAR-10 dataset [9]; details are given in section 4.1.

We performed multiple ablation studies to gain insight into SqueezeNet both at the broader achitectural level in terms of the number of network layers, as well as the micro-architectural details of SqueezeNet's Fire modules, which form the basic building blocks of the network. Most interestingly, we find that adding a batch normalization [10] layer to the modules helps increase performance with little additional cost.

2 SqueezeNet

2.1 Background

A common approach to improving the accuracy of a convolutional neural network (CNN) is increase the size of the network, for example by adding more layers [11]. But this approach has costs for the size needed to store the network, and the number of operations required for training and prediction. SqueezeNet is an attempt to go in

the other direction: shrink the network, while maintaining a baseline level of accuracy. The baseline in this case is AlexNet [7]. AlexNet is best known for its breakthrough performance in the 2012 edition of the ImageNet Large Scale Visual Recognition Challenge [8]. It was the the first CNN do to so, and helped foster the current interest in deep learning models. All subsequent ImageNet competitions have been won by some CNN model. AlexNet's accuracy is no longer state of the art: models exist with better accuracy. Some, such as VGG [12], have many more parameters, although a few, such as Google's Inception network, have fewer.

2.2 SqueezeNet model design

In this section, we go through the Iandola et al. [1] study and explain their proposed model. The main objective of SqueezeNet's architecture is to optimize for parameter count, building a small model that meets an acceptable level of accuracy. They mention three main strategies to meet this goal:

- 1. Replace (some) 3x3 filters with 1x1 filters. This leads to fewer parameters and works as a means of dimensionalty reduction. In the filter space, 1x1 convolution acts like coordinate-dependent transformation [13].
- 2. Decrease the number of inputs to 3x3 filters. This again decreases the total parameter count.
- 3. Downsample late in the network (closer to the output layer). This is an attempt to maximize accuracy on a limited parameter budget. In a convolutional network, each layer produces an output feature map, the size of which is determined by the the size of the input data and the choice of layers. Downsampling in a CNN is typically achieved by setting stride greater than 1 in a convolutional filter. Doing this early in a network leads to smaller feature maps; in SqueezeNet downsampling is performed later in the network, with the intuition that larger feature maps can lead to greater classification accuracy, something strongly suggested in a study by He & Sun [14].

SqueezeNet's Fire module implements all three of these strategies. It is a repeatable building block of the network, consisting of a squeeze convolutional layer (of 1x1 filters) to reduce the parameters i.e. implementing strategy 1, an expand layer that has a mix of 1x1 and 3x3 convolutional filters. The microarchitecture and macroarchitecture of SqueezeNet are shown in Figures 1 and 2, respectively. It is composed of 8 Fire modules (numbered 2-9) and 2 convolutional layers, maxpooling layers, and a final global average pool before a softmax activation function. SqueezeNet performs maxpooling with a stride of 2 after layers conv1, fire4, fire8 and conv10 as part of the third strategy above.

Dropout [15] with a ratio of 0.5 is applied after the final Fire module. To evaluate the accuracy of SqueezeNet, Iandola et al. used AlexNet as a comparison baseline, settling on a initial learning rate [16] of 0.04, with subsequent linear decay. They reported that with a model 50x smaller than AlexNet [7], they achieved the same top-1 and top-5 accuracy on ImageNet. They also report that SqueezeNet can undergo significant size compression without accuracy loss; we address this more in section 4.7.

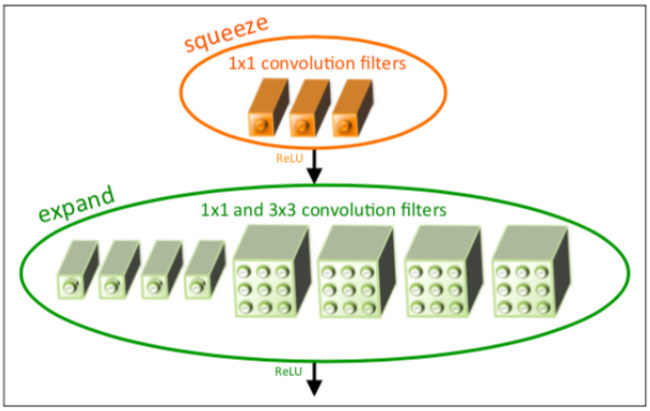

Figure 1: Microarchitectural view: Organization of convolution filters in the Fire module.[1]

2.3 Architectural considerations

The "micro-architectural" details of SqueezeNet concerns the construction of the Fire module. Each module has three hyperparameters: s_{1x1}, the number of 1x1 filters in the initial "squeeze" layer, e_{1x1}, the number of 1x1 filters in the second, "expand" layer, and e_{3x3}, the number of 3x3 filters, all in the expand layer. The eight Fire modules have 24 such parameters between them. Figure 2 shows an example module with a squeeze layer of three 1x1 filters, followed by an expand layer of four 1x1 and 3x3 layers each.

The authors define a *squeeze ratio* (SR) as the ratio between the number of filters in squeeze layers and the number of filters in expand layers. By varying the architecture, the authors found that increasing SR beyond 0.125 can increase the level of the accuracy on ImageNet around from 80.3 % to 86%, but increases the model size from

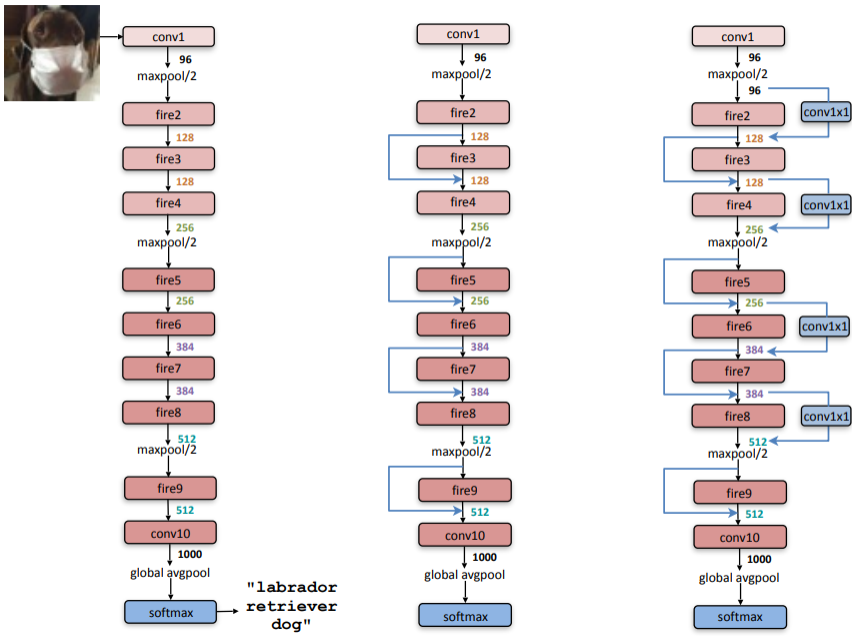

Figure 2: Macroarchitectural view of the SqueezeNet architecture. Left: SqueezeNet; Middle: SqueezeNet with simple bypass; Right: SqueezeNet with complex bypass.[1]

4.8MB to 19MB. Accuracy peaks at 86% at SR = 0.75. Adding more 3x3 filters leads to a larger model without any improvement in accuracy on ImageNet. Fire modules in SqueezeNet vary in size, but all have SR = 0.125 (note that the image in Fig 1, drawn from the SqueezeNet paper, actually shows SR = 0.375).

Macro-architectural experiments in the SqueezeNet paper involve adding bypass connections between layers in the network. The motivation here is to allow a means for information to pass through the network that might otherwise be squeezed out by the Fire modules. The authors explore both "simple" and "complex" connections (see Fig. 2). Simple connections just repeat the outputs of previous layers later in the network, this requires matching input and output sizes between non-consecutive modules. Complex connections add a convolution to the connection between layers: input and output sizes need not match, but this adds further parameters to the network. In experiments the authors found that simple bypass lead to greater improvement than complex ones.

3 Related work

Here we consider only work related to the elements of SqueezeNet that we examine in this paper. In particular we do not consider literature on model compression or on the use of bypass connections in neural networks.

Work on neural networks has a history going back to work in the 1940s and 50s with with work in neurophysiology, cybernetics and early conceptual models such as the perceptron [17, 11]. Neural networks have grown in and out of fashion since, and the beginning of the current deep learning era is typically dated to around 2006, with the appearance of several seminal papers [18, 19, 20]. Convolutional neural networks have been the state of the art for image classification since the 2012 appearance of AlexNet [7], although with noteworthy earlier successes, such as the work of LeCun et al. with recognizing handwritten digits [21].

3.1 Resource-optimized networks

SqueezeNet is part of a growing field of resource-optimized learning networks. The main optimization in SqueezeNet is for size, as measured by parameter count. Other models also optimize for power usage, often by limiting the number of operations, typically measured as the number of multiply-accumulate operations; there are

also networks that optimize for communication latency. Many of these small models are intended for embedded or mobile applications, particularly for smartphones; examples include MobileNet [22, 23], ShuffleNet [24], and Condensenet [25]. Google's mobile neural network architecture MnasNet ("Mobile Neural Architecture Search") is the result of more recent work into advancing the development of mobile neural networks by automating the network design process itself [26].

3.2 Network microarchitecture

Although the title of the original SqueezeNet paper refers to AlexNet (aka "Supervision" [7]), the architecture of SqueezeNet has only a passing resemblance to classic CNNs such as AlexNet or VGG [12]. Instead it uses a network of repeated "modules" that are themselves small CNNs; this puts it in the same category as more recent networks such as Google's "Inception" network (also called "GoogLeNet"[1]).

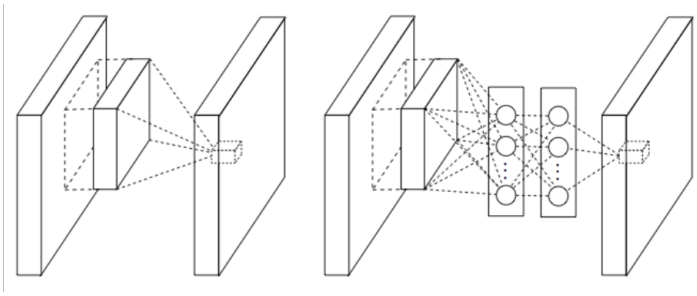

Figure 3: The "Network in Network" model from Lin et al. [27]. The left image shows the operation of a standard CNN convolutional filter, on the right is their approach which replaces the convolutional filter with a miniature neural network.

The repeatable building-block design of both Inception and SqueezeNet draw on ideas from the 2013 paper "Network in Network" from Lin et al [27]. The authors argue that the convolution filter of a standard CNN provides only a low level of feature abstraction: although it later passes through a non-linear activation function, the filter itself is linear, so represents only what the authors call a "generalized linear model" of the local input passing through the filter. Better, non-linear function approximators are available, among them are neural networks themselves. The network-in-network approach proposes replacing the linear convolutional filters of a CNN with a "micro-network", essentially itself a small CNN, see Fig. 3. The micro-network slides over the input in the same manner as a standard convolutional filter, but is better able to represent non-linear features of the input.

Lin et al's micro-networks [27] make extensive use of 1x1 convolutional filters. These ignore the spatial features captured by larger filters such as 3x3, instead operating only over the channels for a particular point in the input: 1x1 filters essentially perform a per-pixel projection into a lower-dimensional feature space.

Both SqueezeNet and Google's Inception network employ the network-in-network strategy, making heavy use of 1x1 filters for dimensionality reduction and following a building-block design. The building blocks for SqueezeNet are the Fire modules, Google calls their micro-networks "Inception modules".

4 Proposed Approach

Our project had five objectives:

Objective 1 To perform a grid search of learning parameters on both AlexNet and SqueezeNet to guide the remaining objectives.

Objective 2 To verify the claim that SqueezeNet performs at similar levels of accuracy as AlexNet does using the best parameters obtained in Objective 1 for each model.

Objective 3 To measure the impact of macroarchitecture modifications to SqueezeNet, namely adding and removing Fire modules.

Objective 4 To determine the impact of microarchitecture modifications, specifically adding batch normalization layers to SqueezeNet's Fire modules.

Objective 5 To determine how transfer learning affects learning on CIFAR.

[1]The name "GoogLeNet" is used ambiguously. In PyTorch and elsewhere, it refers to a particular 22-layer version of the Inception network architecture (also known as "Inception v1") [28]. But it is also used to refer to Google's winning entry in the 2014 ImageNet competition, which was an *ensemble* of 7 networks, only 6 of which used the Inception v1 design. Details are in the papers describing the multiple incarnations of Inception: [28, 10, 29, 30].

The following sections discuss our choice of dataset, the five objectives, and, finally, elements of the model which we did not address.

4.1 Tools and dataset

The code for this project, including the models and analyses, were written in Python 3 [31]. The code for both SqueezeNet and AlexNet was provided by the Torchvision library [32]. Our code also relied heavily on the PyTorch [33] and NumPy libraries [34], and used the SciPy library for statistical testing [35] and the Matplotlib library for the figures in the report [36]. The models were trained, validated, and tested on Google's Colaboratory [37].

We used the CIFAR-10 dataset (Canadian Institude For Advanced Research; 10 classes) [9]. This dataset contains $60,000$ images over 10 classes, which we split into a training set ($45,000$ images), a validation set ($5,000$ images), and a test set ($10,000$ images). All images were randomly cropped, resized to 64×64, and randomly flipped horizontally before each use.

Many datasets were available for the project: the original ImageNet [8], ImagetNet variants (i.e., ImageNet-16, ImageNet-32, and ImageNet-64) [39], CIFAR-10, CIFAR-100 (both from [9]), and CINIC-10 [40]. Table 1 displays the size of the compressed datasets, as well as the time it took to train each model for a single epoch using Google's Colaboratory GPU processors. Using this table, it is easy to understand why we rejected ImageNet, its variants, and CINIC-10; ImageNet was simply too big, whereas the ImageNet variants and the CINIC-10 training times made them impractical for grid searches. Finally, we also rejected CIFAR-100, since the models' performance over the first few epochs gave us reason to believe that it would take at several dozen epochs to get decent results which also makes this dataset impractical for grid searches. In the end, using CIFAR-10 was simply a pragmatic choice.

		Training time	
Dataset	**Size**	**AlexNet**	**SqueezeNet**
ImageNet	161,075 MB[†]	—	—
ImageNet-16	923 MB	1897 s/epoch	1697 s/epoch
CINIC-10	656 MB	158 s/epoch	166 s/epoch
CIFAR-100	161 MB	33 s/epoch	34 s/epoch
CIFAR-10	163 MB	37 s/epoch	39 s/epoch

[†] This is actually the size of the dataset for the ILSVRC2014 challenge [38]. It was not readily possible to obtain the size of the ILSVRC2011 dataset which was used for SqueezeNet.

Table 1: Size of the compressed datasets and time elapsed to train each models for one epoch on each dataset using Google Colaboratory.

4.2 Objective 1: AlexNet and SqueezeNet Grid Search

Our first objective was to find the best parameters for training AlexNet and SqueezeNet. Based on our initial attempts at training SqueezeNet, we figured it out that it would be difficult to train the SqueezeNet model. For instance, one of the first combinations of learning parameters we tried were the ones proposed by Iandola et al. [1] (i.e. learning rate = 0.1, momentum = 0.9, and weight decay = 0.0002), which led to a very poor performance by SqueezeNet (10.0%, chance). We thus decided to obtain the learning parameters more systematically by running a grid search. The parameters and their values are displayed in Table 2. Both models were trained using all 54 combinations of these parameters and tested using CIFAR-10's test set for each.

4.3 Objective 2: AlexNet vs SqueezeNet

Having thoroughly analysed the impact of learning parameters on both models, we then trained them over 50 epochs using the parameters which had produced the best results in the previous step.

4.4 Objective 3: Macroablations and macroextensions of SqueezeNet

We were also interested in the impact of removing and adding Fire modules on SqueezeNet. We believed that some of the issues we faced training SqueezeNet were perhaps due to its size; that, having fewer parameters, there was less room for error in the parameters' values. We expected to find that fewer modules would mean that the model would be that much harder to train, and, conversely, that more modules would make it correspondingly easier

to train. In order to evaluate this hypothesis, we ran four experiments. In the first two, we created two modified versions of SqueezeNet from which one and two Fire modules were removed from the end. The last remaining Fire module's parameters were also adapted to fit with the "classifier" portion of the model. For the third and fourth experiments, we created two versions of the model with one and two extra Fire modules, respectively. Each experiment then consisted of running a grid search identical to the one in Objective 1 with the exception that the batch size was fixed at 64 and weight decay, 0. The parameters are displayed in Table 2.

Parameter	**Objective 1**	**Objective 3**	**Objective 4**	**Objective 5**
Batch size	64, 512, 1024	64	64	64
Learning rate	0.05, 0.01, 0.001	0.05, 0.01, 0.001	0.2, 0.1, 0.05, 0.01, 0.001	0.05, 0.01, 0.001
Momentum	0.9, 0.5, 0.1	0.9, 0.5, 0.1	0.9 0.5, 0.1	0.9 0.5, 0.1
Weight decay	0, 0.002	0	0	0

Table 2: Grid search parameters for Objectives 1, 3, and 4.

4.5 Objective 4: Batch Normalization layers

One obvious solution to the difficulties we were facing with training SqueezeNet was to add batch normalization layers. Following Ioffe and Szegedy's suggestion to place batch normalization layers before non-linearies such as ReLUs [10] and taking inspiration from the VGG-19 architecture [12], we modified the microstructure of the Fire modules by inserting a batch normalization layer between first convolution layer and the first layer of ReLUs. The parameters used for this experiment are displayed in Table 2. We expected that adding batch normalization layers would greatly improve the ability of the network to learn and that it would be able to handle larger learning rates.

4.6 Objective 5: Transfer learning

One other way to improve a model's ability to learn is to use transfer learning. That is to say that we can use the parameter values found by Iandola et al. on ImageNet as starting points, then finetune the network parameters on CIFAR-10 data. Transfer learning is an effective and commonly used technique, particularly in domains such as medical imaging where labelled data is scarce [41, 42].

4.7 Unaddressed issues

Our report does not consider model compression or the addition of bypass connections. Bypass is mentioned only in a single section of the paper and is not part of the central results, and is not present in the PyTorch implementation of SqueezeNet. This was left out in the interests of time.

Concerning model compression, standard SqueezeNet has a size of 4.8MB. The "<0.5 MB model size" claim in the paper refers to the size of SqueezeNet after undergoing a compression technique called "deep compression," involving network pruning, quantizing weights to lower bit depths, Huffman coding, and other techniques, detailed in Han el al. [43]. We have not considered the compressed model size of SqueezeNet for a number of reasons:

- Neither SqueezeNet nor AlexNet suffer any accuracy loss when using deep compression. This is not a technique particular to SqueezeNet or that favours either model.
- Deep compression moves the sizes of the models closer together: compressed SqueezeNet is ∼14.5 times smaller than compressed AlexNet, substantially less than the 50x difference between uncompressed models given in the title of the SqueezeNet paper.
- The compression details are not part of the SqueezeNet paper.

The authors' sole point about model size seems to be that, although SqueezeNet is small, it can still be compressed further without loss of accuracy, to a size small enough to consider printing on a chip. This is not without interest, but arguably speaks more to the power of the deep compression technique itself, which has been applied across different models with similar results.

5 Results and Discussion

5.1 Objective 1: AlexNet and SqueezeNet Grid Search

The results of our grid search on the impact of learning parameters on SqueezeNet and AlexNet are presented in Table A1. There are three things to note. First, as noted previously, the learning parameters recommended to train SqueezeNet on ImageNet [1], namely $learningrate = 0.01$, $momentum = 0.9$, and $weightdecay = 0.0002$, lead to chance-level accuracy (10%) on CIFAR-10 using a batch size of 64 and mild accuracy using larger batch sizes (29.76% for a batch size of 512 and 19.28% for a batch size of 1024). Second, the best parameters for both AlexNet and SqueezeNet within our search space and their corresponding performance are displayed in Table 3. These parameters were used in our second objective. The third and last thing to note is that both models performed best with a batch size of 64 and that weight decay, for a batch size of 64, had little-to-no effect. We therefore chose to fix those parameters to 64 and 0, respectively, in order to reduce the search space for our other grid searches.

Model	Batch size	Learning rate	Momentum	Weight decay	Accuracy
AlexNet	64	0.01	0.9	0	40.74
SqueezeNet	64	0.001	0.9	0.0002	40.04

Table 3: The parameters within our search space which led to the highest performances for both models and the corresponding accuracy on the test set. The full results table, Table A1, can be found in the Appendix.

5.2 Objective 2: AlexNet vs SqueezeNet

Figure 4 compares the performances of AlexNet and SqueezeNet when using optimal parameters. The plots on the left represent runs which used the parameters described in Table 3. These runs also used a step scheduler which reduced the learning rate by a factor of 10 every 10 epochs. The plot on the right displays a run using a different set of parameters: AlexNet was trained using a batch size of 64, a learning rate of 0.0125, a momentum of 0.5, and a weight decay of 0, and SqueezeNet was trained using a batch size of 64, a learning rate of 0.01, a momentum of 0.1, and a weight decay of 0. This run also used an adaptive scheduler [44] rather than a step scheduler; the adaptive scheduler reduced the learning rate by half if the model's validation loss had not decreased over the last three epochs. We present this run here because it led to the highest performances by both models while showing that SqueezeNet is able to match AlexNet's high performance.

5.3 Objective 3: Macroablations and macroextensions of SqueezeNet

We had hypothesized that some of our issues with training SqueezeNet came, perhaps, from its size: with fewer parameters, there would be less room for error. This was motivated, namely, by prior experience with the VGG-19 model on a previous assignment. The results of our grid searches are presented in Table A2. In order to determine whether model size did have an impact, for each condition in our grid searches, we calculated a relative test accuracy for each experimental model. More specifically, we subtracted the experimental model's performance by that of the original, unmodified SqueezeNet for that condition. These relative test accuracies are displayed in Figure 5. Though there does appear to be effect, we found a correlation [45] of $r = -.1972$ which was not statistically significant, $p = 0.2489$.

5.4 Objective 4: Batch normalization layers

Our fourth objective was to evaluate the impact that adding batch normalization layers to the Fire modules would have on model performance. The results of the grid search are presented in Table A3. Using a repeated measures t-test [45], we confirmed that the new model with added batch normalization layers performed significantly better than the original model; $t = 3.14$ (repeated measure), $p < .05$.

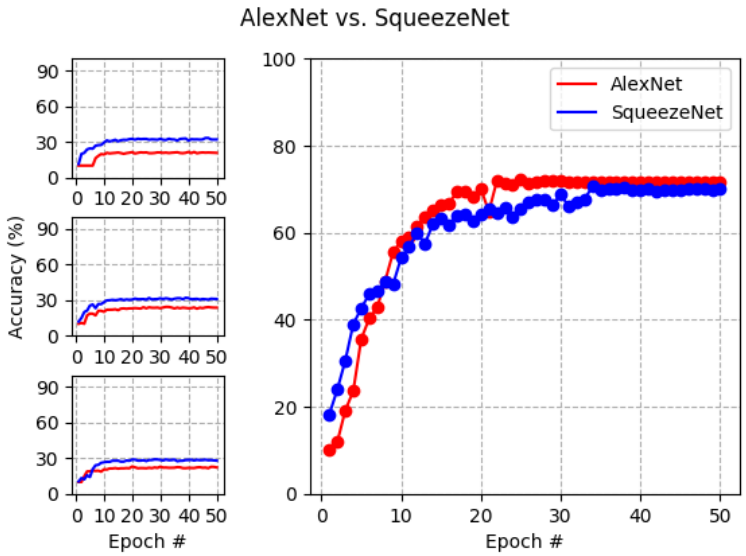

Figure 4: AlexNet vs. SqueezeNet. This figure highlights both the fact that SqueezeNet can match AlexNet's performance and even outperform it, but also that initial weights appear to matter enormously when training these models. For the three plots on the left, the parameters used were those reported in Table 3. For the large plot on the right, both models were trained using an alternative set of parameters and using an adaptive scheduler, rather than a step scheduler.

5.5 Objective 5: Transfer learning

The results for transfer learning on CIFAR-10 using the model parameters from ImageNet are in Table A4. As expected, the model performed significantly better using the ImageNet parameters than random parameter values; $t = 2.33$ (repeated measures), $p < .05$.

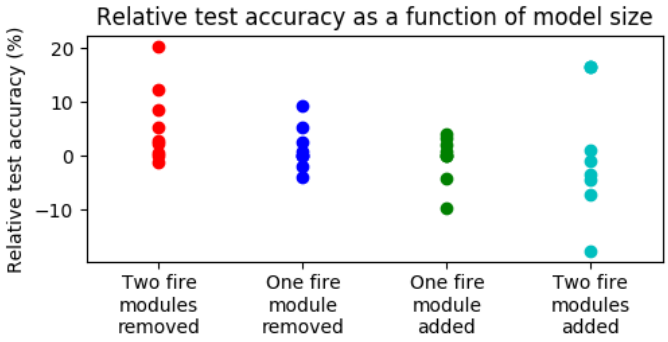

Figure 5: The impact of macroablations and macroexpansions on SqueezeNet's performance after 10 epochs. All relative test accuracies were calculated by subtracting the modified model accuracy with the unmodified SqueezeNet's accuracy under the same conditions.

5.6 General discussion

Using a grid search, we found the combinations of parameters under which AlexNet and SqueezeNet performed best after ten epochs. Using those to compare the models' performances over 50 epochs, we came to two conclusions.

The first is that, given a good set of parameters, SqueezeNet is indeed able to match or even outperform AlexNet, even when the latter is trained using its best set of parameters (see Figure 4).

The second is that parameter initialization matters. That is to say that we were surprised to find that the models performed significantly more poorly when trained on 50 epochs than they had during grid search. After investigating, we discovered that the random number generator seed was only set at the beginning of the experiment, meaning that the models in the searches were initialized with different parameter values. That parameter initialization has such a strong impact on performance suggests that SqueezeNet may have an irregular optimization landscape in which the starting position is very important. One can imagine, for example, that there may be many local minima, most of which are shallow, with a few deep minima which greatly increase the model's performance.

We hypothesized that the irregular shape of the optimization space might be due to the fact that SqueezeNet is smaller: it has fewer parameters and therefore less room for error. The results from Objective 3, however, suggest otherwise. Instead, these results suggest that the issue may be one of propagation: loss is more easily back propagated through the smaller models which are therefore able to improve more quickly. This is consistent with

results from Bjorck et al. [46] who reported that large gradient updates were more likely to result in diverging losses in deeper models. This also fits with the common adage in Machine Learning: "Deeper neural networks are more difficult to train" [47].

Given the difficulties we faced with training SqueezeNet, one obvious solution was to add batch normalization layers. Adding batch normalization layers significantly increased SqueezeNet's ability to learn, as well as the speed at which it learned. These results are in line with previous studies on the impact of implementing batch normalization layers [46].

This begs the question: Why, then, did the authors not include batch normalization layers in the original model? We believe that it is possible that the authors did not know about them or learned about them too late. The Ioffe et al. [10] paper came out only a year before the Iandola et al. [1] paper was first submitted. The other possibility is that the authors underestimated the importance of batch normalization layers. We found that a second version of SqueezeNet, SqueezeNet 1.1 [48], does not implement batch normalization either and although SqueezeNext [49], the spiritual successor to SqueezeNet, does implement them, the paper only mentions batch normalization once, in passing. Whatever the case may be, it is worth mentioning that adding batch normalization layers in the way that we have only adds 1673 parameters to the model, an increase of 0.13% in the number of parameters. The issue is therefore not one of size.

Finally, we used transfer learning to train SqueezeNet on CIFAR-10. As expected, we found that the model performed significantly better over a wide range of learning parameters. This again highlights the importance of initial parameter values.

6 Conclusion

In conclusion, initial parameter values matter and starting with a pre-trained model makes training significantly easier. If using a pre-trained model is not an option, we recommend using batch normalization layers. Though they do increase the number of parameters in the model, we believe that they are a worthy addition, since they make the model much easier to use. After all, "size isn't everything, it's what you (can) do with it that counts."

Statement of Contributions

All members contributed equally to the report and to the direction of the project. Most of the programming was handled by Xavier. Everyone contributed to running the experiments (Colab requires frequent interaction).

References

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

[2] Rytis Verbickas, Robert Laganiere, Daniel Laroche, Changyun Zhu, Xiaoyin Xu, and Ali Ors. Squeezemap: fast pedestrian detection on a low-power automotive processor using efficient convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 146–154, 2017.

[3] Geraldin Nanfack, Azeddine Elhassouny, and Rachid Oulad Haj Thami. Squeeze-segnet: a new fast deep convolutional neural network for semantic segmentation. In *Tenth International Conference on Machine Vision (ICMV 2017)*, volume 10696, page 106962O. International Society for Optics and Photonics, 2018.

[4] V Mohanraj, Ramachandra Guda, and JV Kameshwar Rao. Hybrid approach for pixel-wise semantic segmentation using Segnet and Squeezenet for embedded platforms. In *First International Conference on Artificial Intelligence and Cognitive Computing*, pages 155–163. Springer, 2019.

[5] M Betancourt-Hernández, G Viera-López, and A Serrano-Muñoz. Automatic diagnosis of rheumatoid arthritis from hand radiographs using convolutional neural networks. *Revista Cubana de Física*, 35(1):39–43, 2018.

[6] Mohammad Motamedi, Daniel Fong, and Soheil Ghiasi. Fast and energy-efficient CNN inference on IoT devices. *arXiv preprint arXiv:1611.07151*, 2016.

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

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR09*, 2009.

[9] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. Dataset downloaded from `https://www.cs.toronto.edu/~kriz/cifar.html`.

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

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

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

[13] Kye-Hyeon Kim, Sanghoon Hong, Byungseok Roh, Yeongjae Cheon, and Minje Park. Pvanet: Deep but lightweight neural networks for real-time object detection. *arXiv preprint arXiv:1608.08021*, 2016.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] Dmytro Mishkin, Nikolay Sergievskiy, and Jiri Matas. Systematic evaluation of convolution neural network advances on the imagenet. *Computer Vision and Image Understanding*, 161:11–19, 2017.

[17] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[18] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[19] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[20] Christopher Poultney, Sumit Chopra, Yann L Cun, et al. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems*, pages 1137–1144, 2007.

[21] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[22] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[24] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

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

[26] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

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

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[29] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[30] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[31] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Paramount, CA, 2009.

[32] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pages 1485–1488, New York, NY, USA, 2010. ACM.

[33] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[34] Travis E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, USA, 2nd edition, 2015.

[35] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2019-04-17].

[36] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science Engineering*, 9(3):90–95, May 2007.

[37] Google Colaboratory. `https://colab.research.google.com`.

[38] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[39] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017.

[40] Luke Nicholas Darlow, Elliot J. Crowley, Antreas Antoniou, and Amos J. Storkey. CINIC-10 is not imagenet or CIFAR-10. *CoRR*, abs/1810.03505, 2018.

[41] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[42] Minyoung Huh, Pulkit Agrawal, and Alexei A Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[43] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[44] torch.optim. `https://pytorch.org/docs/stable/optim.html`.

[45] MartinJ Crowder. *Analysis of repeated measures*. Routledge, 2017.

[46] Johan Bjorck, Carla P. Gomes, and Bart Selman. Understanding batch normalization. *CoRR*, abs/1806.02375, 2018.

[47] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[48] Wolfram Neural Net Repository. SqueezeNet V1.1 trained on imagenet competition data. https://resources.wolframcloud.com/NeuralNetRepository/resources/SqueezeNet-V1.1-Trained-on-ImageNet-Competition-Data, 2017. Accessed: 2019-04-17.

[49] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter H. Jin, Sicheng Zhao, and Kurt Keutzer. SqueezeNext: Hardware-aware neural network design. *CoRR*, abs/1803.10615, 2018.

Appendix

Batch size	Learning rate	Momentum	Weight decay	AlexNet accuracy (%)	SqueezeNet accuracy (%)
64	0.05	0.9	0	10.0	nan
			0.0002	10.0	nan
		0.5	0	10.0	nan
			0.0002	19.92	nan
		0.1	0	28.85	nan
			0.0002	29.53	nan
	0.01	0.9	0	40.74	27.69
			0.0002	22.57	10.0
		0.5	0	31.96	32.49
			0.0002	27.77	33.35
		0.1	0	20.46	35.21
			0.0002	24.78	35.23
	0.001	0.9	0	22.67	37.49
			0.0002	23.25	40.04
		0.5	0	10.0	33.04
			0.0002	11.78	33.47
		0.1	0	10.03	30.73
			0.0002	10.02	29.06
512	0.05	0.9	0	nan	nan
			0.0002	10.0	nan
		0.5	0	10.0	nan
			0.0002	20.76	nan
		0.1	0	10.0	10.0
			0.0002	18.47	10.0
	0.01	0.9	0	10.0	24.24
			0.0002	19.26	29.76
		0.5	0	10.0	24.85
			0.0002	10.04	24.25
		0.1	0	10.0	24.31
			0.0002	9.96	22.39
	0.001	0.9	0	10.2	28.8
			0.0002	10.78	20.11
		0.5	0	10.0	20.77
			0.0002	10.0	14.53
		0.1	0	10.0	18.37
			0.0002	10.0	19.67
1024	0.05	0.9	0	19.77	nan
			0.0002	nan	nan
		0.5	0	10.0	10.0
			0.0002	10.0	nan
		0.1	0	10.58	10.0
			0.0002	10.0	10.0
	0.01	0.9	0	10.0	15.84
			0.0002	10.0	19.28
		0.5	0	10.0	17.72
			0.0002	10.0	17.47
		0.1	0	11.59	12.38
			0.0002	10.0	19.79

0.001	0.9	0	10.0	22.13
		0.0002	10.09	18.89
	0.5	0	9.99	14.13
		0.0002	10.0	13.11
	0.1	0	10.28	13.64
		0.0002	10.71	13.51

Table A1: Results of learning parameter grid search for AlexNet and SqueezeNet.

Version	**Learning rate**	**Momentum**	**Modified SqueezeNet accuracy (%)**
1 Fire module removed	0.05	0.9	10.0
		0.5	10.0
		0.1	10.0
	0.01	0.9	37.03
		0.5	37.8
		0.1	36.0
	0.001	0.9	40.06
		0.5	29.14
		0.1	28.86
2 Fire modules removed	0.05	0.9	nan
		0.5	30.35
		0.1	18.44
	0.01	0.9	40.1
		0.5	37.71
		0.1	37.62
	0.001	0.9	40.38
		0.5	33.62
		0.1	29.56
1 Fire module added	0.05	0.9	nan
		0.5	nan
		0.1	nan
	0.01	0.9	31.84
		0.5	35.73
		0.1	25.65
	0.001	0.9	39.49
		0.5	33.76
		0.1	26.52
2 Fire modules added	0.05	0.9	26.52
		0.5	26.52
		0.1	26.52
	0.01	0.9	10.0
		0.5	29.04
		0.1	28.09
	0.001	0.9	38.44
		0.5	32.19
		0.1	26.4

Table A2: Results of grid search experiments on ablated and expanded versions of SqueezeNet.

Learning rate	Momentum	Modified SqueezeNet accuracy (%)
0.2	0.9	41.99
	0.5	55.12
	0.1	56.92
0.1	0.9	49.31
	0.5	51.35
	0.1	49.65
0.05	0.9	52.44
	0.5	54.37
	0.1	50.21
0.01	0.9	53.83
	0.5	41.79
	0.1	41.47
0.001	0.9	43.79
	0.5	34.08
	0.1	30.73

Table A3: Results of a grid search on SqueezeNet with added batch normalization layers.

Learning rate	Momentum	SqueezeNet accuracy (%)
0.05	0.9	nan
	0.5	nan
	0.1	nan
0.01	0.9	10.0
	0.5	64.7
	0.1	67.66
0.001	0.9	64.2
	0.5	64.79
	0.1	59.88

Table A4: Results of a grid search using a pre-trained SqueezeNet to evaluate transfer learning.