Final Project: Write-up

Ruohan Wang, Katayoun Sayar, Philippe Sayegh (Group 75) — COMP551 Final Project

Apr 15th, 2019

Abstract *Recent studies have shown that Deep Neural network (DNN) models should be designed to be tightly correlated with hardware resources in order to not only maximize accuracy and throughput, but also minimize energy and cost in an efficient manner. In this project, we will highlight various efforts that have been made towards the correlated design of DNN models and hardware. SqueezeNet [8] is a small CNN architecture with AlexNet[11]-level accuracy that is accomplished by 50x fewer parameters on ImageNet[3]. Our main goal is to analyze how the creators of SqueezeNet worked towards such a goal, by ablating and/or tweaking components of their model and recording the impact of these changes. As a result, we found that, by adjusting the microarchitecture of SqueezeNet [9], it is possible to improve the validation accuracy on a subset of ImageNet [3] from 32.05% to 32.85% with 1.4 times fewer parameters. We also explored the design strategies proposed by the creators of SqueezeNet [9] and confirmed the importance of late downsampling for maintaining the accuracy of SqueezeNet [9] model.*

1 Introduction

Generally, by dealing with smaller-sized CNN architecture with fewer parameters with no significant accuracy degradation, we offer the following advantages: (1) Communication bandwidth across servers will be preserved for distributed training with smaller CNNs; (2) Offloading new models from the Cloud to the Edge (e.g., to be deployed to IoT devices) requires significantly less bandwidth when it comes to smaller-sized CNNs; (3) Smaller-sized CNNs are more likey adopted to resource-constrained devices such as FPGAs with very constrained memory.

SqueezeNet [8] with its proposed efficient network architecture has modules called fire modules. Fire modules consist of a 'squeeze' layer with 1x1 filters feeding an 'expand' layer with 1x1 and 3x3 filters. With this kind of architecture, squeezeNet able to get AlexNet-level accuracy on ImageNet with 50x fewer parameters, and then can further do network compression on this to get up to 510x smaller than AlexNet (0.5Mb). SqueezeNet begins with a standalone convolution layer, followed by 8 Fire modules, ending with a final convolution layer. The number of filters per fire module from the beginning to the end of the network were gradually increased. Downsampling is also done using Max Pooling layers with a stride of 2 after the first convolution layer, third fire module, seventh fire module, and the final convolution layer.

In this project, we explore different architecture settings of CNN and investigate the influence of microarchitecture and position of downsampling on the model size and accuracy. We found that although improvement of accuracy often appears at the same time with a growth of model size, the right adjustment on the microarchitecture of SqueezeNet [9] can decrease the number of parameters that the model has and achieve a even better accuracy than SqueezeNet [9] on our subset of the Imagenet [3] dataset. Experiments also confirm that late downsampling is an important strategy to improve the accuracy of the model without changing its number of parameters.

2 Related Work

Deep/Convolutional neural network (DNN/CNN) is a promising technique which has recently been very successful on a variety of recognition and classification tasks [15]. However, recent studies revealed that DNN models should be designed tightly correlated with hardware resources in order to maximize accuracy and throughput as well as minimizing energy and cost in an efficient manner. Broadly speaking, techniques that consider correlation/adaption between DNN/CNN models and hardware resources can be classified

into two different categories: (i) techniques that reduce the precision of operations and operands which covers approaches like linear/non-linear quantization [4] or weight sharing [1]; (ii) techniques that reduce number of operations and model size which includes approaches such as network pruning [5] and compact network architectures [8, 2, 7]. Here, in this project, we have investigated approaches that are proposed for the second category.

2.1 Network Pruning

Pruning removes a large number of redundant weights from the over-parameterized networks. This process is called network pruning. Aggressive network pruning often requires some fine-tuning of the weights to maintain the original accuracy. The idea was to compute the impact of each weight on the training loss referred to as the weight saliency. The low-saliency weights were removed and the remaining weights were fine-tuned; this process was repeated until the desired weight reduction and accuracy were reached. Likewise, the authors in [5] proposed pruning based on the magnitude of the weights as a metric rather than using the saliency, because of the heavy computational burden that saliency produced for the large-scaled DNNs. Small weights were pruned and the model was fine-tuned to restore the accuracy. Also by applying fine-tuning, over 80% of the weights were pruned. This approach overall could reduce the number of weights in AlexNet by 9 times and the number of multiplication and accumulation by 3 times. It is worth noting that the weight reduction had greater influence on the fully-connected layers (by 9.9x) rather than convolutional layers (by 2.7x).

2.2 Compact Network Architectures

This technique argues that the number of weights and operations can also be reduced by improving the network architecture itself. The idea behind this is to replace a large filter with a series of smaller filters, which have fewer weights in total. Indeed, by applying filters sequentially, they can obtain the same overall effective result. This technique can be either applied before training (when designing the network architecture) or after training (when deploying the inference model) by decomposing the filters of a trained model. The latter one avoids the hassle of training networks from scratch. However, it is less flexible than the former one.

Compact before training

Filters with a smaller width and height are applied more frequently because concatenating several small filters can represent a larger filter. For example, one 5×5 convolution can be replaced with two 3×3 convolutions. Also, one N×N convolution can be decomposed into two 1-D convolutions, one N×1 and one 1×N convolution [17]. Furthermore, a 3-D convolution can also be replaced by a set of 2-D convolutions followed by 1×1 3-D convolutions as demonstrated in Xception [2] and MobileNets [7]. 1×1 convolutional layers can also be applied to reduce the number of channels in the output feature map for a particular layer. It reduces the number of filter channels and, in turn, computation cost for the filters in the subsequent layer as described in [6, 16, 13]. In order to do this, the number of 1×1 filters has to be less than the number of channels in the 1×1 filter. SqueezeNet [8] uses these 1×1 filters to significantly reduce the number of weights. The authors in [8] proposed a fire module that first 'squeezed' the model with 1×1 convolution filters and then expanded it with multiple 1×1 and 3×3 convolution filters. This approach resulted in approximately 50× reduction in number of weights compared to AlexNet, while maintaining the same accuracy.

Compact after training

Regarding the approaches applied after training on inference models, in order to decompose filters in a trained model without impacting the accuracy, tensor decomposition is proposed[14, 10]. It considers weights in a layer as a 4-D tensor and splits it into a combination of smaller tensors (i.e., several layers).

Low-rank approximation can then be applied to further increase the compression rate at the cost of accuracy degradation, which can be restored by fine-tuning the weights. This methodology takes advantage of the combination of CP-decomposition with low-rank approximation in order to achieve a speed-up on CPUs [12]. However, CPdecomposition cannot be computed in a numerically stable way when the dimension of the tensor, which represents the weights, is larger than a predefined constant (i.e., 2).

In this project, our experiments focus on before training approaches (more specifically Squeezenet) on a subset of ImageNet, investigating the impact of microarchitecture of the model on the model size and accuracy. We also explore the importance of late downsampling for maintaining the accuracy of the model after decrease the model size.

3 Dataset and Evaluation

In the SqueezeNet [8] paper, the model is trained to classify images using the ImageNet [3] dataset. The results are then evaluated using comparison with AlexNet [11] and model compression results associated with AlexNet. However, in our project, due to our limited resources, we only used a small subset of the ImageNet [3] dataset to reproduce the experiments done in the SqueezeNet paper, to train and to validate our modified SqueezeNet model.

The dataset used in this project consists of 200 classes. Each class has 40 training images and 10 validation images. The images are all 64×64 RGB images, but since the SqueezeNet model requires the size of the input images to be at least 224×224, we re-sized the images in our dataset to 224×224. After re-sizing, the only pre-processing we did was to normalize the images.

Because of the difference between the dataset used in the SqueezeNet paper and the dataset used in this project, the results of the experiments done in this project cannot be evaluated by comparing with SqueezeNet or AlexNet [11] trained on the entire ImageNet [3] dataset. The main focus of our work is also not to improve the accuracy, but to explore the influence of the CNN architecture on model size and accuracy. Therefore, we trained AlexNet [11] and SqueezeNet [9] on the same subset of ImageNet [3] that we used to train and test our models for comparison. As shown in figure 2, AlexNet achieves a top-1 accuracy of 15.40% and a top-5 accuracy of 35.65%, while SqueezeNet achieves a top-1 accuracy of 12.75% and a top-5 accuracy of 32.05%.

4 Experiments and Results

4.1 Design Strategies for constructing SqueezeNet-like architectures

As demonstrated in the SqueezeNet paper, there are three design strategies employed to construct the SqueezeNet architecture. The three strategies are:

- Replace 3x3 filters with 1x1 filters.
- Decrease the number of input channels to 3x3 filters.
- Downsample late in the network so that convolution layers have large activation maps.

The purpose of the first two strategies is to decrease the quantity of parameters of a CNN without sacrificing the accuracy. The quantity of parameters of a convolution layer comprised of kxk filters is:

$$(\text{number of input channels}) \times (\text{number of filters}) \times (k \times k)$$

So reducing the number of input channels and reducing the size of filters can both decrease the quantity of parameters.

The Fire modules of the SqueezeNet [9] model consists of a 'squeeze' layer and an 'expand' layer. The 'squeeze' layer, which has less filters than the 'expand' layer, is designed to limit the number of input channels to 3x3 filters. The output of 'squeeze' layer is then fed to the 'expand' layer where some of the 3x3 filters are replaced by 1x1 filters. The number of filters in a 'squeeze' layers (denoted as s_{1x1}), the number of 1x1 filters in an 'expand' layers (denoted as e_{1x1}), the number of 3x3 filters in an expand layers (denoted as e_{3x3}) are therefore important hyperparameters to consider for constructing SqueezeNet-like microarchitecture.

4.2 Microarchitecture

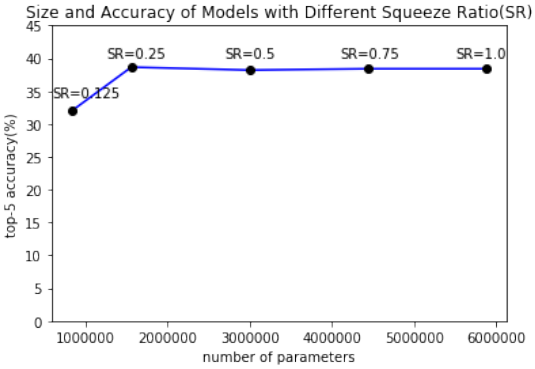
As defined in the Squeeze [9] paper, the microarchitecture of SqueezeNet-like models can be controlled by 5 high level metaparameters:

1. $base_e$, the number of expand filters in the first Fire module.
2. $freq$, the frequency of which the number of expand filters is increased. In other words, the number of expand filters is increased after every $freq$ Fire modules.
3. $incr_e$, the amount by which the number of expand filters is increased after every $freq$ Fire modules. Therefore for the ith Fire module, the number of expand filters is $e_i = base_e + (incr_e \cdot \lfloor \frac{i}{freq} \rfloor)$.
4. pct_{3x3}, the percentage of expand filters that are 3x3 inside the Fire modules.
5. squeeze ratio (SR), the ratio between squeeze filters and expand filters within the Fire modules. In other words, $s_{i,1x1} = SR \cdot (e_{i,1x1} + e_{i,3x3})$

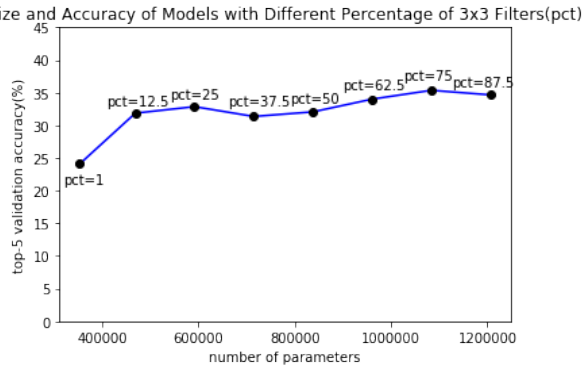
Among the five metaparameters, squeeze ratio, which controls the number of input channels to expand layers, and percentage of 3x3 expand filters are investigated in the SqueezeNet [9] paper. We reproduced their experiments to show how squeeze ratio and the percentage of 3x3 expand filters can affect the model size and accuracy. In the following experiments, we use the number of parameters to evaluate the model size. The models are all trained on the training set of the mini-Imagenet which consists of 200 classes and 40 images per class. The accuracy of the models are then evaluated using the top-1 accuracy and top-5 accuracy that the models achieve on the validation set which contains 10 images per class. We also trained the AlexNet [11] model on the same training set for comparison. AlexNet [11] achieves 15.40% top-1 accuracy and 35.65% top-5 accuracy on the validation set, while SqueezeNet [9] achieves 12.75% top-1 accuracy and 32.05% top-5 accuracy on the same validation set but has 72.9 times less parameters than AlexNet [11].

First, to investigate the effect of the squeeze ratio on the model size and accuracy, we use SqueezeNet as a starting point and modify the architecture by adjusting squeeze ratio. We trained 5 models each with a different squeeze ratios in the range $[0.125, 1.0]$. The other metaparameters used in these experiments are set to: $base_e = 128$, $incr_e = 128$, $freq = 2$, $pct_3x3 = 0.5$. The results are shown in figure 1a and figure 2. From the figures, we can see that increasing the squeeze ratio to 0.25 can further improve the top-5 accuracy of the model to 38.65%, while the number of parameters is also increased to 1.85 times of that of SqueezeNet [9] but the model size is still much smaller that AlexNet [11]. However, increasing the squeeze ratio beyond 0.25 increases the size of the model but does not improve the accuracy. This discovery also agrees with the results of experiments done in the SqueezeNet [9] paper.

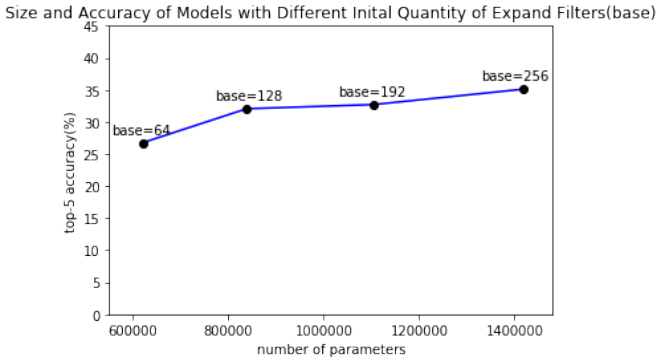
To explore how the percentage of 3x3 expand filters affects the model size and performance, we trained multiple models with metaparameters $base_e = 128$, $incr_e = 128$, $freq = 2$, $SR = 0.125$ and pct_{3x3} in the range from 1% to 87.5%. The total number of expand filters is the same for all the models trained in these experiments, but the ratio between 1x1 filters and 3x3 filters changes from "mostly 1x1" to "mostly 3x3".



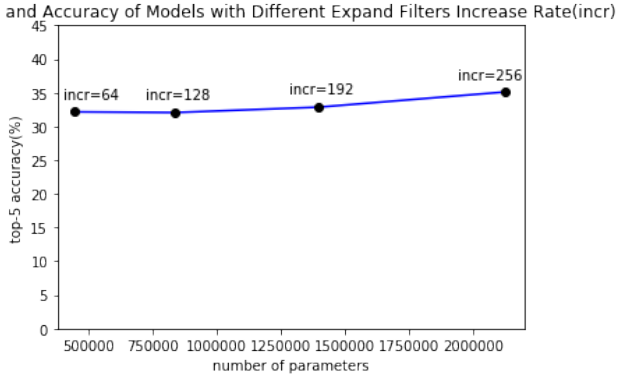
(a) Impact of squeeze ratio (SR) on the model size and accuracy



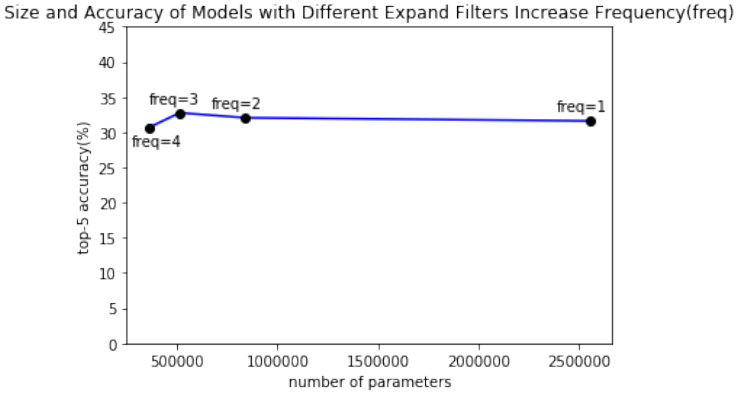
(b) Impact of the percentage of 3x3 expand filters (pct_{3x3} on the model size and accuracy



(c) Impact of the quantity of expand filters in the first Fire module ($base_e$) on the modle size and accuracy



(d) Impact of the increase rate of expand filters ($incr_e$) on the modle size and accuracy



(e) Impact of the increase frequency of expand filters ($freq$) on the modle size and accuracy

Figure 1: Explore microarchitectural design space

	Metaparameters					Accuracy		Model size
	base_e	incr_e	pct_3x3	freq	SR	top-1 accuracy(%)	top-5 accuracy(%)	number of parameters
AlexNet	\	\	\	\	\	15.40	35.65	61100840
SqueezeNet	128	128	50.0	2	0.125	12.75	32.05	838024
Experiments on base_e	**64**	128	50.0	2	0.125	9.05	26.75	620104
	192	128	50.0	2	0.125	12.85	32.70	1104072
	256	128	50.0	2	0.125	14.35	35.10	1418248
Experiments on incr_e	128	**64**	50.0	2	0.125	13.25	32.15	443944
	128	**192**	50.0	2	0.125	13.05	32.85	1397992
	128	**256**	50.0	2	0.125	14.70	35.10	2123848
Experiments on pct_3x3	128	128	**1.0**	2	0.125	9.20	24.10	354184
	128	128	**12.5**	2	0.125	11.90	31.85	469384
	128	128	**25.0**	2	0.125	12.45	32.85	592264
	128	128	**37.5**	2	0.125	12.50	31.35	715144
	128	128	**62.5**	2	0.125	12.90	34.00	960904
	128	128	**75.0**	2	0.125	14.30	35.35	1083784
	128	128	**87.5**	2	0.125	13.80	34.65	1206664
Experiments on freq	128	128	50.0	**1**	0.125	13.00	31.60	2556552
	128	128	50.0	**3**	0.125	13.10	32.75	513720
	128	128	50.0	**4**	0.125	11.10	30.65	359688
Experiments on SR	128	128	50.0	2	**0.250**	16.15	38.65	1556680
	128	128	50.0	2	**0.500**	15.60	38.20	2993992
	128	128	50.0	2	**0.750**	17.30	38.40	4431304
	128	128	50.0	2	**1.000**	16.75	38.40	5868616

Figure 2: Results of experiments on the five metaparameters in comparison with AlexNet [11] and SqueezeNet [9]

The results are shown in figure 1b and figure 2. The results show that increasing the percentage of expand filters that are 3x3 can generally improve the accuracy although it also increases the size of the model. But when $pct_{3x3} = 25$, the model achieves a top-5 accuracy of 32.85% with 1.415 times less parameters than SqueezeNet [9]. Also note that increasing the percentage of 3x3 expand filters to 87.5 increases the size of the model but leads to a drop on the accuracy.

In addition to reproducing the above experiments, we also explored effect of the other three metaparameters, $base_e$, $incr_e$, and $freq$, on the model size and accuracy. These three metaparameters together decide the total number of expand filters in each Fire module. In the following experiments, we use SqueezeNet [9], which uses the metaparameters: $base_e = 128$, $incr_e = 128$, $freq = 2$, $pct_{3x3} = 0.5$, $SR = 0.125$, as a starting point and modify only one metaparameter for each model. First, we trained 4 models with $base_e$ in the range $[64, 256]$, with the other metaparameters left to their default SqueezeNet values. Then, we trained 4 models with $incr_e$ in the range $[64, 256]$ and, similarly, the other four metaparameters were set to their SqueezeNet values. Lastly, we trained 4 models with $freq$ in the range $[1, 4]$, with the other metaparameters set to their default SqueezeNet values. The results are shown in figure 1c, figure 1d, figure 1e, and figure 2.

From the figures, we learn that, in general, increasing the total number of expand filters in Fire modules results in both an growth of the model size and an improvement on accuracy. However, we also notice that changing $incr_e$ from 128 to 64 leads to a better accuracy than SqueezeNet [9] with the number of parameters being only half as much as that of SqueezeNet. At the same time, adjusting $base_e$ to 256 and adjusting $incr_e$ to 256 results in nearly the same accuracy but the model with $incr_e$ set to 256 has 1.5 times more parameters than the model with $base_e$ set to 256. Distinguishing it from the other two metaparameters, $freq$ does not seem to have a significant impact on the model size and accuracy. As shown in figure 2, changing the frequency from 3 to 2 or from 2 to 1 only increases the model size but does not improve the accuracy. we observe that setting $freq$ to be 3 achieves a better accuracy (top-5 accuracy of 32.75% and top-1 accuracy

of 13.1%) than SqueezeNet [9] and decreases the number of parameters by 1.63 times.

4.3 Downsampling

As suggested in Strategy 3 in section 4.1, the SqueezeNet's team's main strategy to maintain higher accuracy while shrinking the model is to postpone downsampling steps to a further position in the neural network's flow in order to preserve information as much as possible. In the SqueezeNet [9] model, downsampling is done using Max Pooling layers with stride set to more than 1. We explored the usefulness and implementation of this strategy by tweaking the positions of two of three of the model's max pooling layers (the first always stays right after the first convolution layer). The top-1 accuracy and top-5 accuracy results of all the experiments are displayed in Fig. 3 and Fig. 4.

The original model places the two later Max Pooling layers (kernel size of 3, stride of 2) after the third and seventh Fire modules.

For the first experiment, of those two max pooling layers, we left one between the third and the fourth Fire modules, and moved the other to right after the first Fire module. This resulted in a 2% drop of accuracy measured on validation set overall when compared to SqueezeNet [9] at the end of the training, and through most of the training epochs.

For the second experiment, we moved one pooling layer to between the second and the third Fire modules while the other pooling layers are still located as in the first experiment. This time, the accuracy dropped much more significantly: by about 6% at the end of the training, but with a drop of about 10% through a great part of the training (from around epoch 20 to 45).

For the third experiment, we instead pushed the two later max-pooling layers to a much later stage in the model's architecture, pushing the author's strategy to as close to the extreme as possible while still being able to train the model within a reasonable time. The architecture contains one pooling layer between

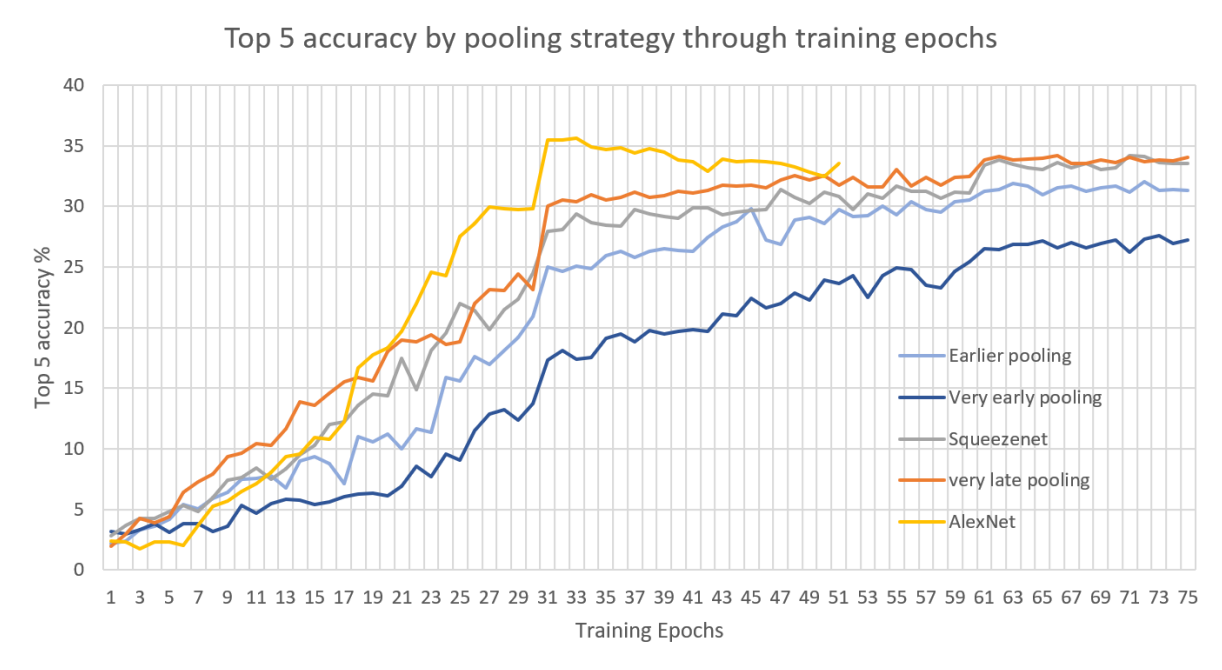


Figure 3: Top 5 accuracy by pooling strategy through the training process

	Position of pooling layers	Top-1 accuracy(%)	Top-5 accuracy(%)
AlexNet	\	15.40	35.65
SqueezeNet	After 1st Convolution layer Between 3rd and 4th Fire Modules Between 7th and 8th Fire Modules	12.75	32.05
Very early pooling	After 1st Convolution layer Between 1st and 2nd Fire Modules Between 2nd and 3rd Fire Modules	10.00	27.55
Early pooling	After 1st Convolution layer Between 1st and 2nd Fire Modules Between 3rd and 4th Fire Modules	12.00	31.85
Very late pooling	After 1st Convolution layer Between 6th and 8th Fire Modules Between 7th and 8th Fire Modules	13.45	34.15

Figure 4: Accuracy of models with downsampling at different positions

the sixth and the seventh Fire modules, and one between the seventh and the eighth Fire modules. This resulted in a very slight increase (about 0.5%) in accuracy comparing to the SqeezeNet results at the end of a training, but made the training process much longer.

These experiments seemed to concur with the intuition of the authors of the SqueezeNet [9] paper, as placing the max pooling layers later seems to correlate with an increase in resulting validation accuracy, albeit at a potentially high cost in time and memory requirements if pushed too far.

5 Discussion and Conclusion

In this project, we investigated the 3 design strategies presented by the authors of SqueezeNet [9] for decreasing the model size of CNN without sacrificing the accuracy. From our experiments, we found that increasing the squeeze ratio of SqueezeNet from 0.125 to 0.250 can greatly improve the accuracy of the model although it also increases the size of the model. Increasing the percentage of expand filters that are 3x3 in the Fire modules can also improve the accuracy with a growth of model size, but adjusting pct_{3x3} from 25.0 to 37.5 leads to a drop on accuracy and still increases the number of parameters. Adjusting the three metaparameters $base_e$, $incr_e$, and $freq$ to increase the total number of expand filters in the Fire module also achieves a gain in accuracy, however it also results in a larger model size. Therefore, we conclude that adjustments on the architecture of the models that improve the accuracy normally also lead to a growth on the model size, but the changes on model size and on accuracy are not always correlated. Making the right adjustments on the architecture, for example, setting $incr_e$ to 64 instead of 128, leads to both a better accuracy and fewer parameters.

By comparing models which apply downsampling at different positions of the architecture, we reaffirm that late downsampling does indeed greatly improve the accuracy of the model without changing the number of parameters in the model. However, applying downsampling too late in the model also results in a longer training process. After trying models with downsampling done at different stages, we found that the downsampling strategy used by SqueezeNet [9] indeed achieves the best result/cost balance, when using two max pooling layers, plus one right after the initial convolution.

6 Statement of Contribution

All group members contributed to the design and implementation of the project to write Python script and to the analysis of the results and working on the write ups. To be more specific, Ruohan and Philippe accomplished ”Dataset and Evaluation”, ”Experiments and Results” also ”Discussion and Conclusion”. Katayoun worked on ”Abstract”, ”Introduction” and ”Related Work”.

References

[1] Wenlin Chen et al. “Compressing neural networks with the hashing trick.” In: International Conference on Machine Learning. 2015, pp. 2285–2294.

[2] François Chollet. “Xception: Deep learning with depthwise separable convolutions.” In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2017, pp. 1251–1258.

[3] J. Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database.” In: CVPR09. 2009.

[4] Song Han, Huizi Mao, and William J Dally. “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.” In: arXiv preprint arXiv:1510.00149 (2015).

[5] Song Han et al. “Learning both weights and connections for efficient neural network.” In: Advances in neural information processing systems. 2015, pp. 1135–1143.

[6] Kaiming He et al. “Deep residual learning for image recognition.” In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, pp. 770–778.

[7] Andrew G Howard et al. “Mobilenets: Efficient convolutional neural networks for mobile vision applications.” In: arXiv preprint arXiv:1704.04861 (2017).

[8] Forrest N Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size.” In: arXiv preprint arXiv:1602.07360 (2016).

[9] Forrest N. Iandola et al. “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size.” In: CoRR abs/1602.07360 (2017).

[10] Yong-Deok Kim et al. “Compression of deep convolutional neural networks for fast and low power mobile applications.” In: arXiv preprint arXiv:1511.06530 (2015).

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “Imagenet classification with deep convolutional neural networks.” In: Advances in Neural Information Processing Systems. p. 2012.

[12] Vadim Lebedev et al. “Speeding-up convolutional neural networks using fine-tuned cp-decomposition.” In: arXiv preprint arXiv:1412.6553 (2014).

[13] Min Lin, Qiang Chen, and Shuicheng Yan. “Network in network.” In: arXiv preprint arXiv:1312.4400 (2013).

[14] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. “Introduction to tensor decompositions and their applications in machine learning.” In: arXiv preprint arXiv:1711.10781 (2017).

[15] Vivienne Sze et al. “Efficient processing of deep neural networks: A tutorial and survey.” In: Proceedings of the IEEE 105.12 (2017), pp. 2295–2329.

[16] Christian Szegedy et al. “Going deeper with convolutions.” In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015, pp. 1–9.

[17] Christian Szegedy et al. “Rethinking the inception architecture for computer vision.” In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, pp. 2818–2826.