MiniProject 4: Evaluating SqueezeNet and Exploring Variations

Daniel Chernis	260707258
Archit Gupta	260720757
Chenthuran Sivanandan	260749843

Abstract

For the last mini-project, we chose to re-implement SqueezeNet, a Deep CNN. As the ImageNet dataset, that the original paper uses, is too big for us to reasonably complete a number of experiments, we use the CIFAR-10 dataset of small images and modify the SqueezeNet model accordingly. We first evaluate the original model on the CIFAR-10 dataset. After that, we run different experiments by modifying the model architecture. The experiments we performed were (1) - Modifying max pooling kernel size from 3 to 2; (2) - Changing convolutional kernel size from 7 to 2; (3) - Removing a fire module and decreasing dropout; (4) - Delaying downsampling. Finally we choose the highest accuracy model from the experiments above and test it on the CIFAR-100 dataset. For small images of the CIFAR-10 dataset, we were able to increase the accuracy of the original model by making these modifications.

1. Introduction

Deep convolutional neural networks perform well on image recognition tasks (in terms of accuracy), Unfortunately their large size severely limits them due to their large number of parameters. From [11] we learn that large models are not good at distributed data-parallel training. AlexNet is a perfect example of this phenomenon. AlexNet is a large model, and frequent updates are not realistic with AlexNet, because it usually requires more than 240MB for updates. Many researchers have developed methods to make large DNNs smaller, while retaining their accuracy, because a smaller DNN can be much more efficient than a large network.

For this mini-project we evaluated the SqueezeNet model, which is much smaller than the AlexNet model (in terms of the number of parameters), but maintains the same accuracy as AlexNet. For our experiments, we used the CIFAR dataset as the ImageNet dataset requires a lot of computational power for training. We started out by running the original model using the CIFAR dataset and obtaining a baseline. After that we ran a few experiments where we changed the stride, convolutional kernel size, decreased the amount of dropout, and the maximum pooling kernel size.

2. Related Work

Neural Networks are computationally and memory intensive, making them difficult to deploy on embedded systems. To address this limitation the authors in [2] introduce “deep compression” which is a three stage pipeline: pruning, trained quantization, and Huffman coding, in order to reduce the storage requirement of the neural network. Their method first prunes the redundant connections from the network. Next they quantize the weights to enforce weight sharing and then finally they apply Huffman coding.

In [3] the authors use “deep compression”, which was introduced in [2]. Then the authors propose an energy efficient inference engine (EIE) that performs inference on this compressed network model and

accelerates the resulting sparse matrix-vector multiplication with weight sharing.

Pruning is a popular method for compressing a neural network. In [5] the authors examined ResNets and DenseNets obtained through pruning and tuning, and observed that reduced networks — smaller versions of the original network trained from scratch — consistently outperform pruned networks. They also noticed that if you take a pruned network and train it from scratch it will be significantly more competitive. Finally they compared the inference speed of reduced and pruned networks on hardware, and noticed that reduced networks are significantly faster.

In [4] the authors were motivated by putting neural networks onto mobile devices, however the computational limits of mobile devices simply do not allow them to carry neural networks. In this paper the authors show that competitive compression rates can be achieved by using a version of soft weight sharing. Their method achieves both quantization and pruning in one retraining procedure.

In [1] the authors exploit the linear structure present within the convolutional filters to derive approximations which significantly reduce the required computations. Using large state-of-the-art models, they demonstrate speedups of the convolutional layers on both CPU and GPU by a factor of 2x, while keeping the accuracy within 1% of the original model.

3. Dataset

The original SqueezeNet model was trained using the ImageNet dataset. Unfortunately the hardware we were using did not support the ImageNet dataset. As a result we used the CIFAR dataset for training as well as validation.

3.1 ImageNet

ImageNet is a large dataset of annotated photographs intended for computer vision research. The dataset contains over 10,000,000 images which belong to 10,000 categories. Most notably the ImageNet dataset is used in the ImageNet Large Scale Visual Recognition Challenges (ILSVRC).

3.2 CIFAR

The CIFAR-10 dataset is a collection of images that are commonly used to train computer vision algorithms. The dataset contains 60,000 colour images, each of which are size 32x32, and they belong to ten different classes. Thus every class has 6000 images, 5000 for training and 1000 for testing. The CIFAR-100 dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class.

4. Proposed Approach

We modified the architecture of the SqueezeNet model so that it can work with the CIFAR dataset. We started out by running the original model using the CIFAR dataset and obtaining a baseline. Afterwards we conducted experiments on the architecture of SqueezeNet to see if it could lead to any improvements on the accuracy of the baseline.

4.1 SqueezeNet Model

SqueezeNet is a DNN that has the same accuracy as AlexNet, but with far fewer parameters. The authors achieved fewer parameters by changing the size of filters, as well as modifying the network depth.

4.1.1 Architecture

By reducing the size of a filter, there will be far less parameters. Additionally, the number of input channels to a filter is reduced by Squeeze layers. A squeeze convolution layer only consists of 1 x 1 convolution filters which feeds into an expand layer that is a mix of 1 x 1 and 3 x 3 filters. This structure is referred to as a Fire module. There are 3 hyperparameters in a Fire module: s1x1 - The number of 1x1 filters in the squeeze layer; e1x1 - The number of 1x1 filters in the expand layer and e3x3 - The number of 3x3 filters in the expand layer.

4.2 Experiments

We establish a baseline using the original SqueezeNet model. Then we run the following experiments by making design choice changes to the architecture of SqueezeNet. We conduct 4 different experiments on the CIFAR-10 dataset and 1 experiment on the CIFAR-100 dataset.

Baseline: The original SqueezeNet 1.0: The baseline is constructed using the original architecture of the SqueezeNet model (with all parameters intact). However, as we are using a smaller sized dataset (CIFAR instead of ImageNet), we expect the accuracy to suffer as the original settings (7 x 7 convolutional kernel and 3 x 3 max pooling layers) seem to be overkill for smaller images.

Experiment 1 (E1): Image Preprocessing + Max Pooling Kernel Size: We changed the image preprocessing technique to one better adopted for the CIFAR dataset and also decreased the max pooling kernel size from 3 to 2 to better fit the smaller images.

Experiment 2 (E2): Convolutional Kernel Size: We decreased the convolutional kernel size from 7 to 2, again in an attempt to use parameters better suited for smaller images.

Experiment 3 (E3): Fire_3 Removal + Dropout: In this experiment, we removed the Fire_3 module to change the architecture. Because there are still 6 Fire modules left, we decreased the dropout rate from 0.5 to 0.3 in an attempt to maximize the effects of changes we observe.

Experiment 4 (E4): Delaying Downsampling: We remove Maxpool_1 and then delay the downsampling to reduce the number of parameters.

Experiment 5 (E5): Best Model Using CIFAR-100: We select the best model from the previous experiments and run that model with the CIFAR-100 dataset instead of the CIFAR-10 dataset.

5. Results

Table 1 shows the Top-1 accuracy and number of parameters for all of our 5 experiments along with the baseline. **Table 2** shows the architecture of the best performing model. Here, $s_{1\times1}$ is the number of 1 × 1 filters in the squeeze layer of the Fire module, $e_{1\times1}$ is the number of 1 × 1 filters in the expand layer, and $e_{3\times3}$ number of 3 × 3 filters in the expand layer. Training/Validation accuracy graphs for each experiment are shown in the **Appendix**

Table 1: Top-1 accuracy (%) and number of parameters for different model architectures

	Dataset	Classes	Top-1	Parameters
Baseline	CIFAR-10	10	46.54	740 554
Experiment 1	CIFAR-10	10	58.43	740 554
Experiment 2	CIFAR-10	10	66.17	727 594
Experiment 3	CIFAR-10	10	69.24	715 169
Experiment 4	CIFAR-10	10	72.05	727 594
Experiment 5	CIFAR-100	100	38.39	773 764

Table 2: Architecture of best-performing model.

Layer	Output	filter/Stride/Pad	Depth	s_{1x1}	e_{1x1}	e_{3x3}	Parameters
Input Image	33 x 33 x 3						
$Conv_1$	33 x 33 x 96	2 x 2 x 2/1/1(x96)	1				
$Fire_2$	33 x 33 x 128		2	16	64	64	11 920
$Fire_3$	33 x 33 x 128		2	16	64	64	12 432
$Fire_4$	33 x 33 x 256		2	16	128	128	43 344
$MaxPool_4$	17 x 17 x 256	2 x 2/2/0	0				
$Fire_5$	17 x 17 x 256		2	32	128	128	49 440
$Fire_6$	17 x 17 x 384		2	48	192	192	104 880
$Fire_7$	17 x 17 x 384		2	48	192	192	111 024
$Fire_8$	17 x 17 x 512		2	64	256	256	188 992
$MaxPool_8$	9 x 9 x 512	2 x 2/2/0	0				
$Fire_9$	9 x 9 x 512		2	64	256	256	197 184
$Conv_{10}$	9 x 9 x 10	1 x 1/1/0(x10)	1				5 120
$AdapAvgPool_{10}$	1 x 1 x 10	1 x 1/1/0	0				

6. Discussion and Comparison of Models

The accuracy that we get for the baseline is 46.54 %. This is expected since we used the CIFAR-10 dataset in a model that was built using the ImageNet dataset. We trained this model for 20 epochs using the GPU's provided by Google colab. Most of these models took around 2 hours to train.

We observe higher accuracies (compared to Baseline) for Experiments 1-4 because our architectural modifications were more suitable for training with the CIFAR-10 dataset instead of the ImageNet dataset. By decreasing the max pooling and convolutional kernel sizes and by delaying the downsampling, we were able to more easily extract features from the low resolution images of CIFAR-10.

We observe a high accuracy in Experiment 3 when we removed a $Fire_3$ module and decreased the dropout. This decreases the number of parameters and also the size of the network.

However, we had the best accuracy in Experiment 4 when we delayed the down-sampling until later layers in the network. Because more feature information is retained deeper into the network, our model performs better.

Finally, in Experiment 5, we trained the best model (out of the 4 models described above) on the CIFAR-100 dataset. Our best model was obtained in Experiment 4, where we delayed the downsampling until later layers. However, on training this model with the CIFAR-100 images, we see a sharp decline in our accuracy, from 72% to 38%. This can mostly be attributed to the different nature of CIFAR-100 compared to CIFAR-10 as the number of examples for each class in CIFAR-100 is 600, against 6000 examples for each class in CIFAR-10.

In keeping with the main aim of SqueezeNet, we reduced the number of parameters of our different models mainly by reducing the different filter's kernel sizes and changing the architecture directly by removing a $Fire_3$ module and delaying downsampling.

7. Conclusion

We modified the SqueezeNet architecture to work with the CIFAR dataset. In the end we found that some experiments have a bigger effect on accuracy than others. In the future we would like to explore more combinations of parameters, for example the number of squeeze filters, the kernel sizes in each layer, as well as the number of fire layers. Additionally we would like to experiment with more computation power, since that will allow us to work with ImageNet instead of CIFAR-10.

SqueezeNet is an important step towards containing the size of neural networks while preserving accuracy. A compact DNN has several advantages, like more efficient distributed training, less overhead when transporting models and feasible FPGA and embedded deployment.

8. Contributions

All members contributed equally.

References

[1] Emily Denton et al. Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation. 2014. arXiv: 1404.0736 [cs.CV].

[2] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. 2016. arXiv: 1510.00149v5. URI: https://arxiv.org/pdf/1510.00149v5.pdf.

[3] Song Han et al. Learning both weights and connections for efficient neural networks. In Advances

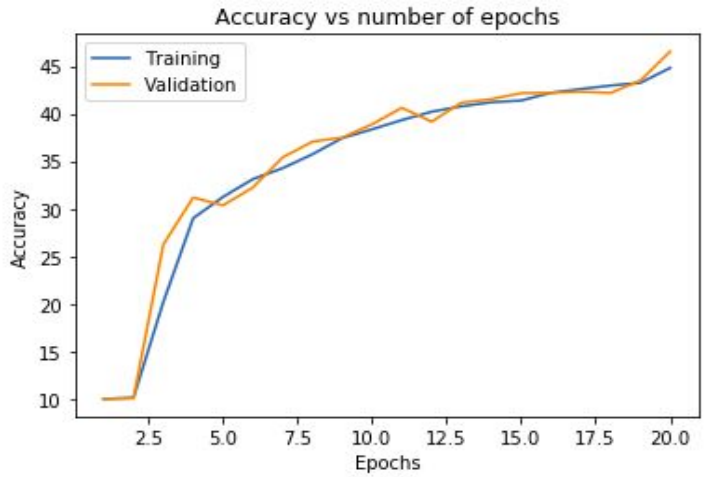
in Neural Information Processing Systems. 2016. arXiv: 1602.01528. URl: https://arxiv.org/pdf/1602.01528.pdf.

[4] Karen Ullrich, Max Welling, and Edward Meeds. Soft Weight-Sharing for Neural Net- work Compression. 2017. arXiv: 1702.04008. URl: https://arxiv.org/pdf/1702. 04008.pdf.

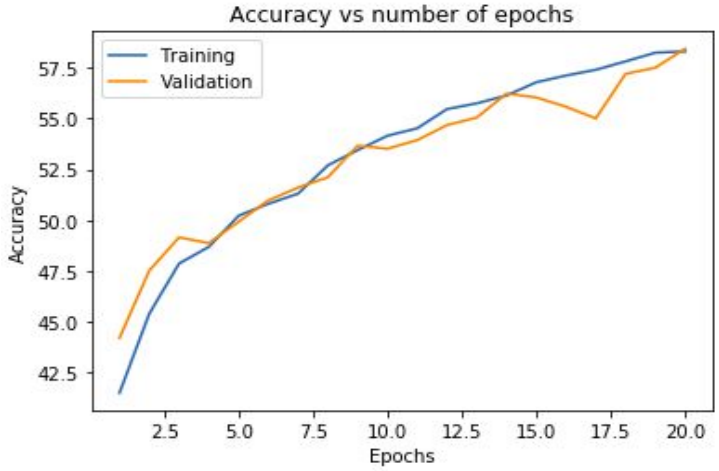
[5] Elliot J. Crowley et al. Pruning neural networks: is it time to nip it in the bud? 2019. arXiv: 1810.04622v2. URl: https://arxiv.org/pdf/1810.04622v2.pdf.

Appendix

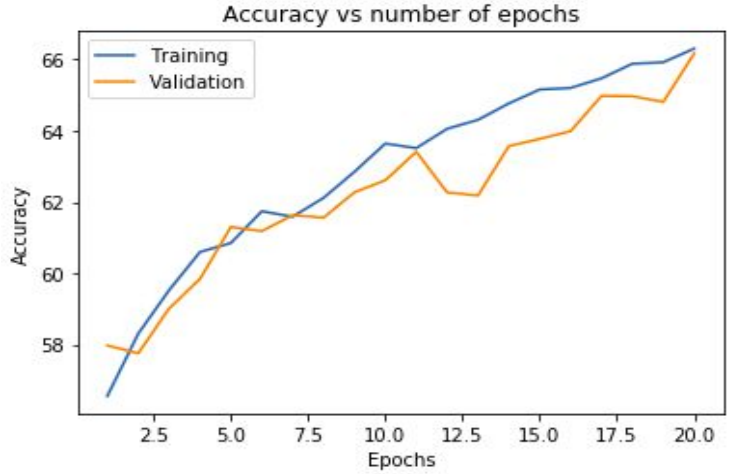
Baseline



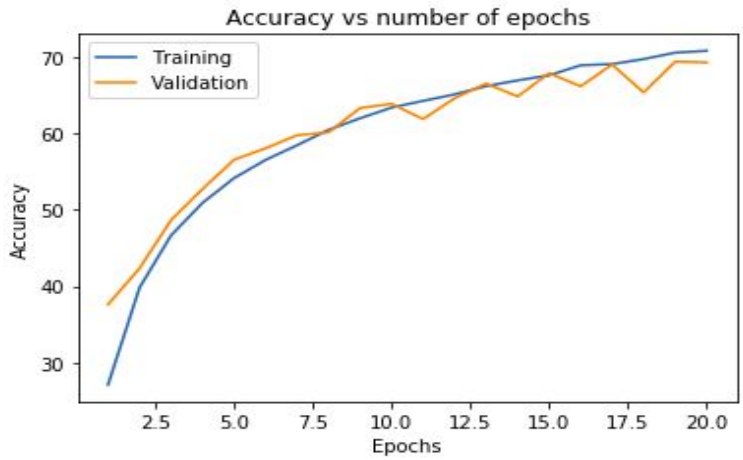
Experiment 1



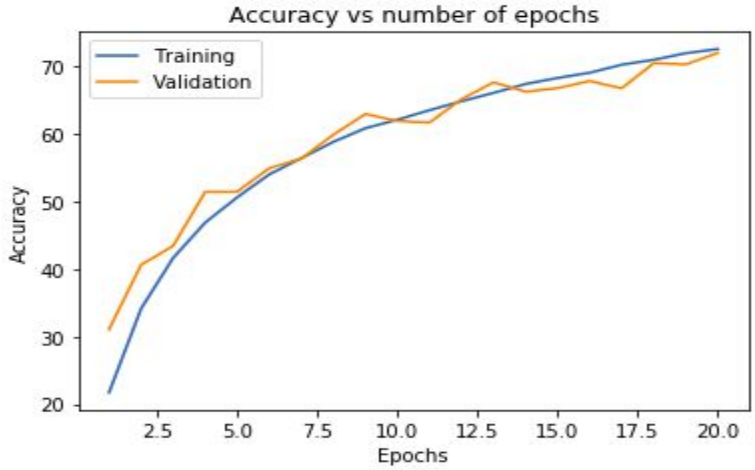
Experiment 2



Experiment 3



Experiment 4



Experiment 5