Project 3 Report : Pruning
Artificial Intelligence on Chip, Winter 2020

Hao Jen Chien UID: 405219534
Ananya Ravikumar UID: 205431526

***Abstract*—In this project, we perform magnitude-based, non-recoverable pruning on SqueezeNet v1.1, and analyse the effects of pruning on the accuracy and size of the model. In this project, SqueezeNet is trained on the CIFAR-10 dataset.**

I. SqueezeNet Architecture

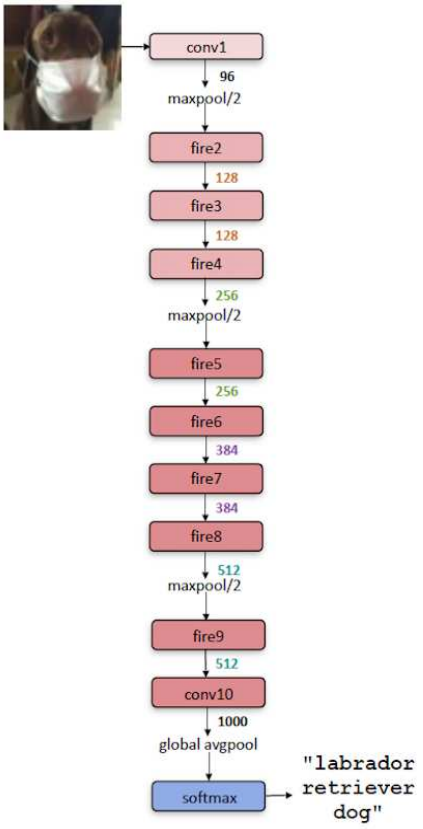

Fig. 1. SqueezeNet Structure[1]

SqueezeNet [1] is a highly compressed convolutional neural network which is able to provide Alexnet-level [2] accuracy with 50x fewer parameters. SqueezeNet achieves this size reduction by:

1) Replacing a majority of 3x3 filters (9 parameters) with 1x1 filters (1 parameter).

2) Reducing the number of input channels to 3x3 filters. This reduces parameters because the number of parameters in a convolutional layer with 3x3 filters is given by (3x3)*(no. of input channels)*(no. of filters).

3) Down-sampling at later stages in the network to increase accuracy.

4) The design of a special unit called a *fire module*, which consists of a *squeeze layer* containing of only 1x1 filters, and an *expand layer* containing both 1x1 and 3x3 filters. The number of 1x1 and 3x3 filters in these layers of the module are hyper-parameters. The architecture of SqueezeNet is shown in figure 1.

The model used in this project is SqueezeNet V1.1. It has 2.4x less computation and slightly fewer parameters than the original SqueezeNet (SqueezeNet V1.0) without any change in accuracy [3].

II. Pruning

In this project, only weights and biases of the network are considered to be prunable parameters. Magnitude-based, non-recoverable, iterative pruning is used. A description of the terminology is given below.

- *Pruning* refers to removing parameters in a neural network in order to decrease its size. A good pruning algorithm will prune as many parameters as possible while minimizing loss of accuracy.

- In the case of *magnitude-based pruning*, parameters whose absolute value lies below a threshold are removed.

- Pruning can be recoverable or non-recoverable. If the type of pruning is *non-recoverable*, a parameter that is removed cannot be added back to the network at a later stage.

- Iterative pruning refers to performing multiple rounds of pruning and re-training to minimize size while keeping the accuracy drop as low as possible.

III. Pruning Algorithm

Pruning and training is done in 2-phases, which are repeated until the model reaches a point at which pruning any more parameters will cause a significant drop of accuracy. The algorithm is depicted in Figure 2, and is described below.

In this pruning algorithm, **Inputs**: Trained model $\mathbf{M}$, set of prunable parameters $\mathbf{P} \in \mathbf{M}$, initial preserved percentage

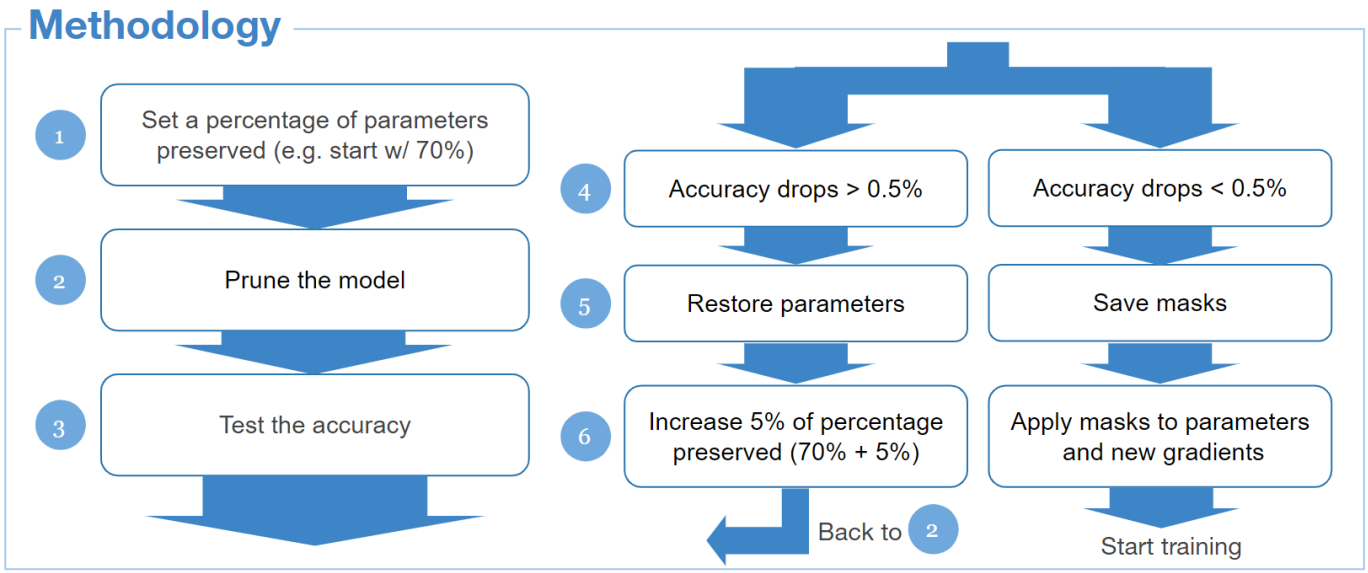

Fig. 2. Pruning Algorithm

(p_{pres}), and x - training and testing datasets. **Output**: Final pruned and re-trained model $\mathbf{M}_P$.

1) **PHASE 1 : PRUNING**

a) Initialise p_{pres}, the percentage of the network preserved, to 70% (or whatever percentage desired).

b) Find a threshold value for each layer such that p_{pres}% of the parameters are preserved ($100 - p_{pres}$% are pruned).

c) Prune the network by setting values of prunable parameters $\mathbf{P}$ which fall below their respective thresholds to 0 and keeping the value of the rest of the parameters. Also set the gradients of pruned parameters to 0.

d) Calculate accuracy of pruned model:

- If the drop in testing accuracy is greater than 0.5%, then increase p_{pres} by 5%. If p_{pres} increases to 100%, which means that there are no parameters being pruned, then go to PHASE 2. Otherwise, return to step 1b.
- If the drop in accuracy is less than 0.5%, go to Phase 2.

2) **PHASE 2: TRAINING**

a) If p_{pres} is less than 100%, re-train the network and compute the testing accuracy. By default, the model begins to re-train where it left off in the previous iteration.

b) If p_{pres} is 100%, the current values held by the parameters are deemed unsatisfactory and the parameters are re-initialised before re-training. This helps to redistribute the weights of parameters, and therefore increases the number of parameters that can be pruned.

c) If the accuracy after retraining drops by more than 5% compared to the accuracy before pruning in this round, abandon the result.

In Phase 2, initialization right before retraining is an important key to continue the pruning method. Decisions of when to implement initialization will have a strong effect on the pruning result. Figure 3 depicts an example of initialization.

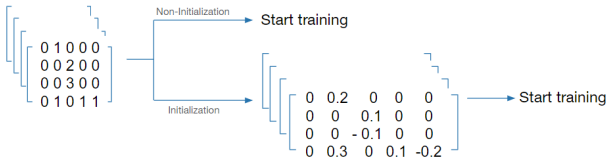

Fig. 3. example of initialization flow

IV. Implementation

A. *Environment*

Below is the setup for the project:

- This project was implemented in Python 3.6.9 using the *tensorflow* library [4].
- A model of SqueezeNet v1.1 was trained on the CIFAR-10 dataset, which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [5]. The code was implemented and run using Google Colaboratory.

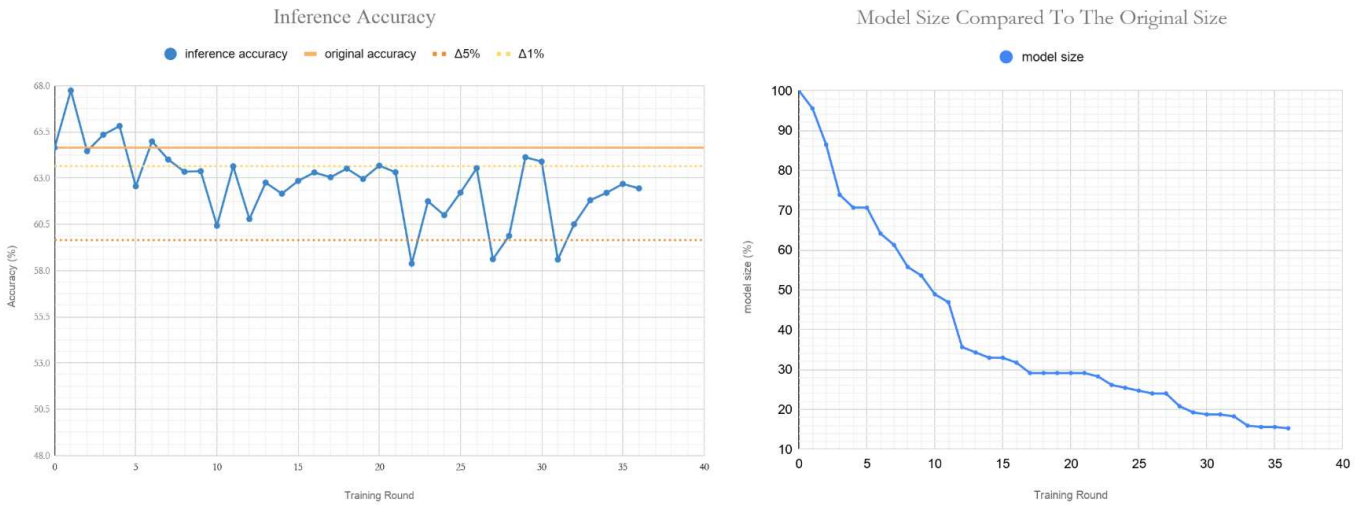

Fig. 4. (left) Accuracy vs. % Training Rounds; (right) Size vs. % Traning Rounds

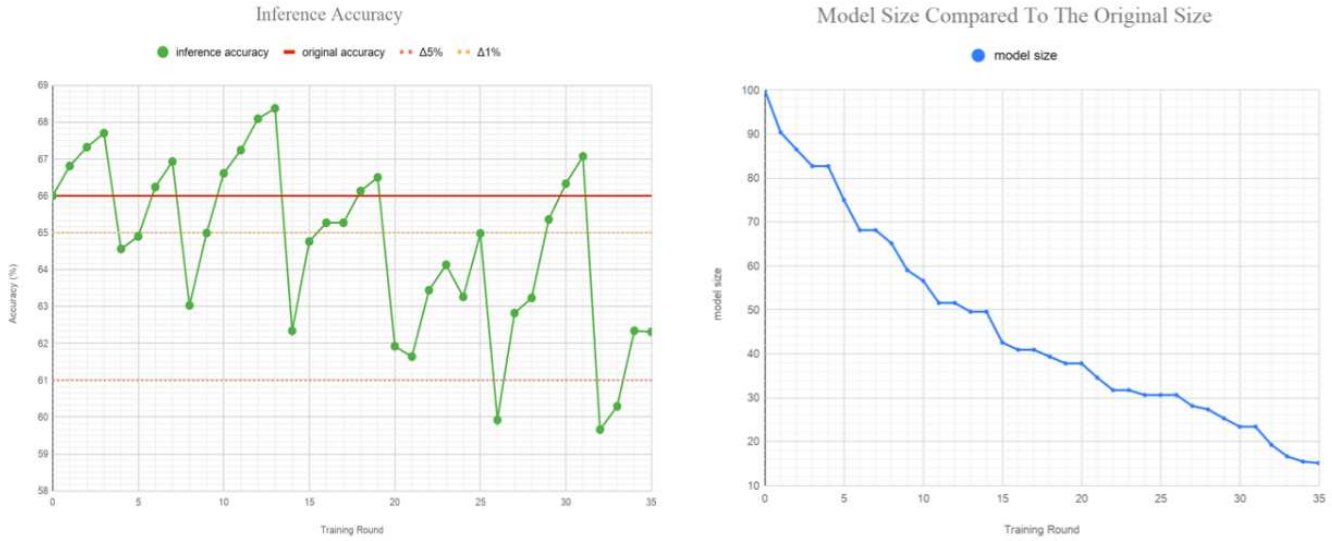

Fig. 5. (left) Accuracy vs. % Training Rounds; (right) Size vs. % Traning Rounds

B. *Fine-Tuning*

There are four hyper-paremeters that can be tuned during the implementation:

- Learning rate
- Training epoch
- Accuracy drop when pruning ($\leq 0.5\%$)
- Accuracy drop after retraining ($\leq 5\%$)

There are no rules to decide values of hyperparameters; the numbers are empirical. We re-ran each training round with different sets of hyperparameters and picked the one that gives the best post-training accuracy. During the whole pruning process, the ”accuracy drop when pruning” is fixed to be 0.5% and ”accuracy drop after retraining” to be 5%.

V. Results

Figure 4 (left) shows how the accuracy of the model varies with multiple rounds of pruning and re-training. It can be seen that Figure 4 (right) shows how the size of the model varies with with pruning. We can see that the accuracy increases in the first few training rounds. This indicates that the model contains redundancy.Figure 5 shows that the pruning yielded similiar results on repeating the experiment.

Trade-off in results The original testing accuracy is 64.54%. The best results are (1) training round 29, with a testing accuracy of 64.19% and a size of 19.26% of the original size, and (2) training round 35, with a testing accuracy of 62.69% and a size of 15.6% of the original size. The first best result only has a drop of 0.35% of accuracy. The second result has a drop of 1.85% of accuracy, but the size of the model is

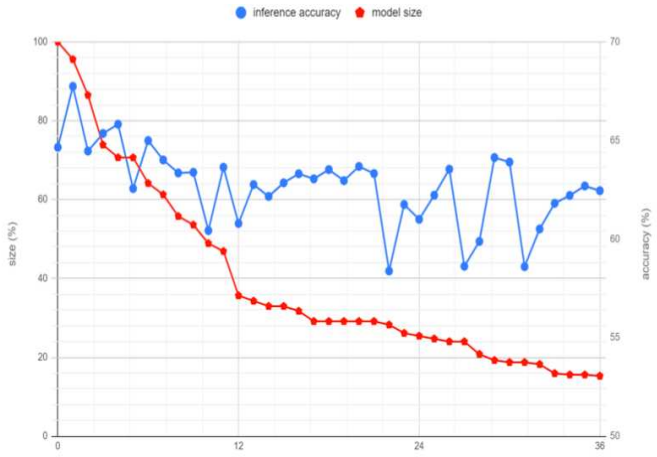

Fig. 6. Trade-off : Accuracy vs. Size

3.66% smaller than the first best result.

The trade-off between size and accuracy is determined by the application. In some cases, accuracy decreasing 5% is acceptable if the size can be significantly small. On the other hand, the size cannot be compressed as small but it is able to keep the accuracy within 1%. It is up to users to decide the goal and therefore the trade-off.

Trade-off in pruning The decision of implementing initialization is a trade-off between continuing pruning and accuracy loss. In Figure 6, there are accuracy drops at the training round 12, 22, 27, and 30. This is due to the initialization. For training round 12, and 22, the model is able to recover a similar accuracy after few training rounds. However, we can see that the model becomes unstable after training round 20. The initialization in training round 27 and 30 causes a much higher accuracy drop, and the model is not able to recover to a similar accuracy afterwards. The accuracy eventually starts dropping significantly after 30 training rounds. The model size is compressed to 15% of the original size.

VI. Observations and Conclusions

There are several trends observed after implementing the pruning algorithm on the model. The tendencies are:

1) After several rounds of pruning, the remaining parameters become more important (increase in magnitude).
2) The model parameters saturate after several iterations of pruning, i.e., the parameter values change very little because a better local minimum of the loss function cannot be found by the learning algorithm.
3) The accuracy drops more after Initialization, but it can be re-obtained after several rounds.

1st observation: In each training round, the default value of percentage of preserved parameters (p_{pres}) is set to 70%. However, for the same percentage, the accuracy drops significantly in later training rounds. This may imply that after pruning several times, the significance of remaining parameters increases.

2nd observation: Based on the first observation and the fact that we encountered a zero pruning decision PHASE 1 of the whole implementation, we conclude that after multiple rounds of pruning, the model is saturated because all the remaining parameters contribute meaning to the result. In order to continue pruning, initialization is introduced to force the weights of the parameters redistributed.

3rd observation: After introducing initialization between pruning and retraining, the testing accuracy generally drops 3% compared to the original testing accuracy. A possible explanation is that the remaining architecture after pruning is specific to a certain distribution of weights. When the parameters are initialized, it is no longer the same distribution, therefore causing the testing accuracy to drop.

VII. Future Work

To further shrink the model, we can look into each layer and prune them individually with different value of percentages instead of using one percentage to prune the layers throughout the model. Our functions are already able to print out values of each layers and the corresponding threshold values.

We can also relax the boundary for "accuracy drop when pruning" from 0.5% to different value of percentage as well as change the increment of percentage from 5% to smaller a number in order to understand the effect of these restrictions.

Link to code (please check this for code with instructions): shorturl.at/hzJMT

References

[1] F. Iandola, M. Moskewicz, K. Ashraf, S. Han, W. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and textless1mb model size," 02 2016.
[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (Red Hook, NY, USA), p. 1097–1105, Curran Associates Inc., 2012.
[3] P. Team, "Squeezenet— pytorch." https://pytorch.org/hub/pytorch_vision_squeezenet/, 2018.
[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.
[5] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.