Quantisation and Pruning for Neural Network Compression and Regularisation

Yael & Yuval Dahari

1 Introduction

In this report we will present the paper "Quantisation and Pruning for Neural Network Compression and Regularisation" [1] by Kimessha Paupamah, Steven James, and Richard Klein, and will try to reproduce the results from their work using more updated libraries. The authors, Kimessha Paupamah, Steven James, and Richard Klein, investigate network pruning and quantisation as techniques to compress neural networks, making them more efficient for consumer-grade and low-powered devices. They compare the efficiency of these compression methods on large networks like `AlexNet` against more compact architectures such as `MobileNet` and `ShuffleNet`, demonstrating significant size reduction and improved speed. Furthermore, their work highlights pruning's utility in correcting overfitting within neural networks. The research concludes that compact architectures, when combined with compression techniques, offer a more promising approach than merely compressing larger networks.

2 Methods Overview

The paper uses two main methods - pruning and quantisation.

2.1 Pruning

The pruning stage is deleting some of the weights of the model, reducing its expressivity and calculation complexity. They noticed that they need to choose carefully which weights they can remove without harming the model performance. To determine which weights are insignificant to the model for pruning, the researchers identified parameters with the smallest weights. The number of parameters to be removed was determined by the network's sensitivity to pruning. This sensitivity was established through "sensitivity scans," which were conducted to find how many of the smallest weights could be removed without negatively impacting the network's performance. They used different approach for every network.

2.2 Quantisation

They used a Q-function to change the weights and reduce the precision of weights and activations by lowering the number of bits used to represent them. In general the function maps floating-point values, the previous weights, to new weight representing by integers using the formula:

$$Q(x, \Delta, z) = round(x/\Delta + z) \tag{1}$$

Where:

- x is the original floating-point value.
- Δ is the scale, which indicates the step size of the quantiser.
- z is the zero-point, an integer to which a floating-point zero maps without error.
- round() is a function that rounds the result to the nearest integer.

simple example of using the formula: Assuming we want to quantise a floating-point weight x to an 8-bit integer. Suppose we have:

- A floating-point weight x = 0.75
- A chosen scale (Δ) = 0.01
- A chosen zero-point (z) = 128 (This z value is common when mapping to an unsigned 8-bit integer range, like orthodox char representation).

Now, we substitute in 1:

$$Q(0.75, 0.01, 128) = round(0.75/0.01 + 128) = round(75 + 128) = round(203) = 203$$

So, the floating-point value 0.75 would be quantised to the integer 203. This integer 203 can then be stored using fewer bits (e.g., 8 bits) compared to the original floating-point representation, thereby reducing the network's size and computational requirements.

3 The Algorithm

The methods were combined sequentially one after the other. First, the network was iteratively pruned, and then the quantisation was applied.

3.1 Iterative Pruning

The process began with iterative pruning, using the following steps:

- Fully training the neural network as usual.
- Iteratively pruning connections with weights below a manually determined threshold (the threshold was found through "sensitivity scans" process, which identify how many of the smallest weights could be removed without negatively impacting performance). Neurons without input or output connections were also removed, creating a sparse network.
- Finally, the new network was retrained. During retraining, weights were generally fine-tuned, but not re-initialised (except for compact architectures like `MobileNet` and `ShuffleNet`, where uniform re-initialisation performed better) to save the understanding of the original network.

3.2 Per-Channel Quantisation

After the pruning process was complete and the networks were sparse and fine-tuned, "per-channel quantisation" was applied. This technique reduces the precision of weights and activations by lowering the number of bits used to represent parameters along the depth (or channels) of a layer. This step is described in the paper as a "quick and efficient way to reduce a network's size without the need for retraining". In this step, the algorithm pass on the network and apply the Q-function (as detailed above) on it.

4 Results

The results of combining pruning and quantisation, achieved several benefits, primarily in reducing the computational and memory requirements of neural networks.

4.1 Sensitivity Scans

The sensitivity scans helped determine how many weights could be removed from a network without making its performance worse.

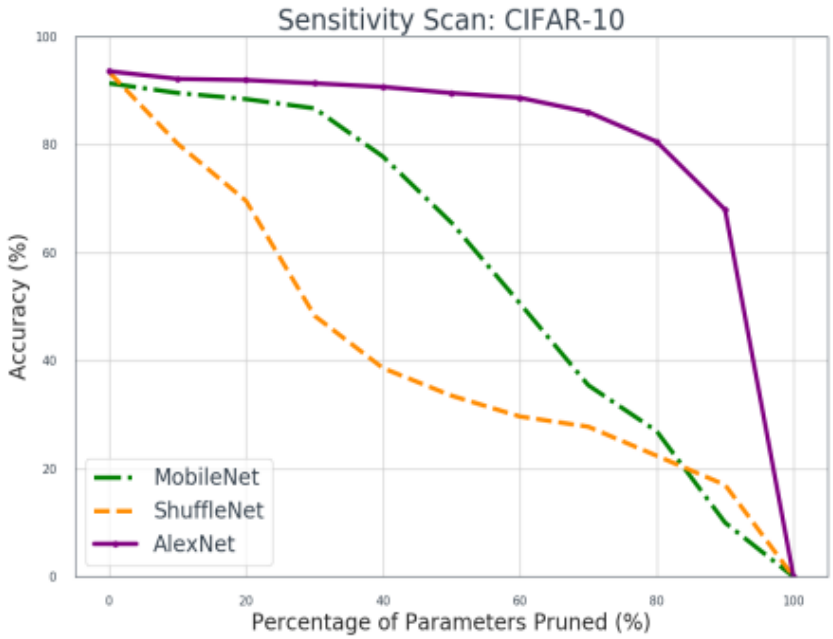

Figure 1: **Their** Sensitivity of pruning networks trained on CIFAR-10

4.1.1 Their Results

- `CIFAR-10` (Figure 1): On this dataset, `MobileNet` performed best with a pruning sensitivity of 0.4, `ShuffleNet` with 0.3, and `AlexNet` with 0.5.
- `FashionMNIST` (Figure 2): On `FashionMNIST`, `MobileNet` responded well to a sensitivity of 0.7, `ShuffleNet` to 0.35, and `AlexNet` performed best with a sensitivity of 0.9.

The results make sense - the small networks (like `ShuffleNet`) were more sensitive to pruning, in contrast big networks like `AlexNet` were less sensitive.

4.1.2 Our Results

In our reproduction, in Figures 3 and 4, we got that all the models were insensitive to up to 85% of parameters pruned, which doesn't make sense, since like we said before, smaller networks are expected to have higher sensitivity to pruning. We will discuss the possible reasons behind these results at the end in length.

4.2 Network Pruning

Tables 1 and 2 investigate the effectiveness of **network pruning** in reducing model complexity while maintaining accuracy across `MobileNet`, `ShuffleNet`, `AlexNet` architectures architectures and `CIFAR-10`, `FashionMNIST` datasets.

4.2.1 Their Results

This is a breakdown of the main trends seen in Table 1:

- **Significant Parameter Reduction:** Pruning successfully reduced the total parameters for all networks. Larger networks like `AlexNet` saw the most substantial reductions (up to 90% on `FashionMNIST`), while `MobileNet` and `ShuffleNet` also achieved parameter decreases.

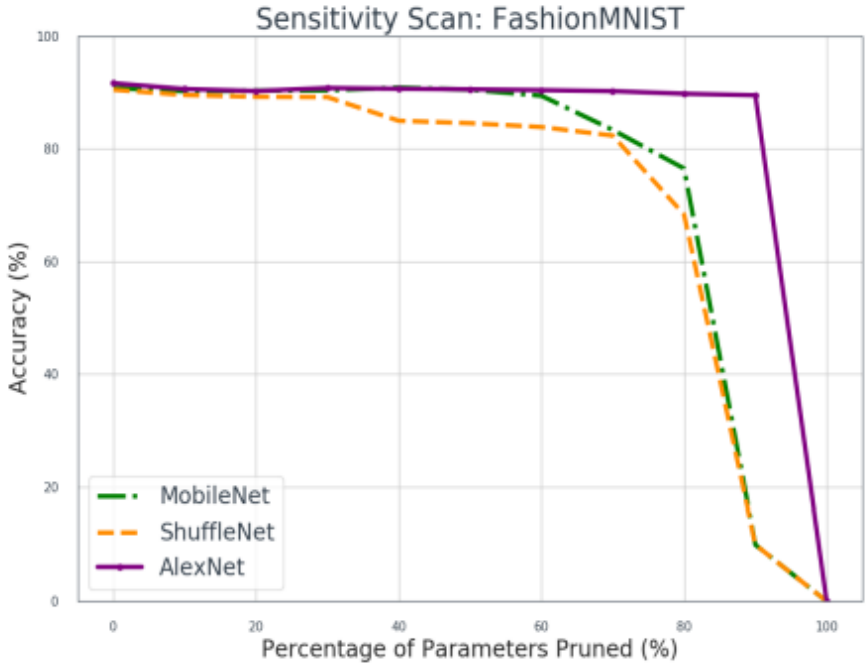

Figure 2: **Their** Sensitivity of pruning networks trained on FashionMNIST

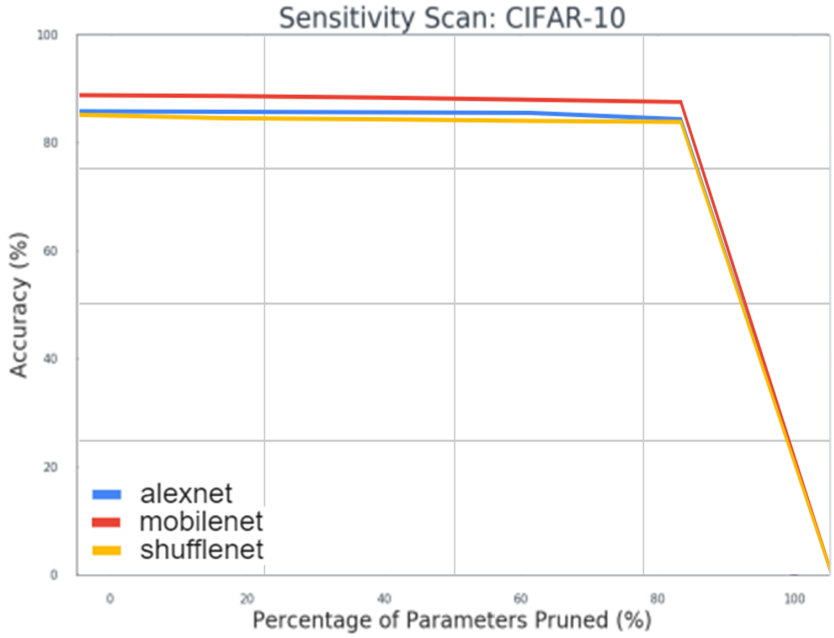

Figure 3: **Our** Sensitivity of pruning networks trained on CIFAR-10

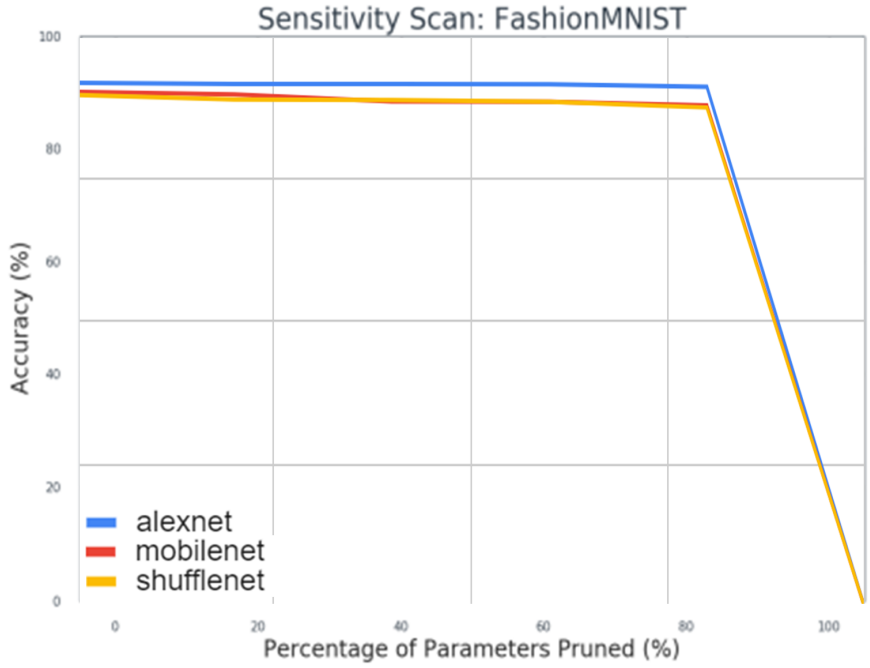

Figure 4: **Our** Sensitivity of pruning networks trained on FashionMNIST

- **Minor Accuracy Changes:** Pruning generally led to only minor changes in accuracy. In some cases, like `MobileNet` on `CIFAR-10`, accuracy even slightly increased. For others, there were small decreases but no significant loss of performance.
- **Dataset Impact:** `FashionMNIST` consistently shows better compression rates compared to `CIFAR-10`, likely due to its smaller, grayscale images allowing for greater network simplification.
- **Retraining Strategies Varied:** The study found that `AlexNet` (as a larger network) benefited from fine-tuning its remaining parameters after pruning, while `MobileNet` and `ShuffleNet` (compact architectures) performed better with re-initialization of their weights.

Table 1: **Their** Network pruning on CIFAR-10 and FashionMNIST

Network	**Accuracy (%)**		**Total Parameters**		**Compression Rate**	
	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST
MobileNet – Reference	91.31	90.75	2.2M	2.2M	–	–
MobileNet – Pruned	91.53	90.42	671K	1.1M	1.6×	2×
ShuffleNet – Reference	93.36	90.36	1.2M	1.2M	–	–
ShuffleNet – Pruned	93.05	90.09	879K	815K	1.4×	1.5×
AlexNet – Reference	93.54	91.61	57M	57M	–	–
AlexNet – Pruned	90.91	90.34	28M	5M	2×	10×

4.2.2 Our Results

This is a breakdown of the main trends seen in Table 2:

- **Accuracy Fluctuations:** Our pruned models showed mixed trends in accuracy. While `MobileNet` slightly decreased in performance (especially on `FashionMNIST`), both `ShuffleNet` and `AlexNet` sur-

prisingly improved after pruning. This suggests that some networks may benefit from pruning-induced regularization.

- **Consistent Parameter Reduction:** As expected, parameter counts were substantially reduced across all models. The compression rates closely match those in the original study, confirming the effectiveness of our pruning pipeline. Notably, `AlexNet` achieved a 10× compression on `FashionMNIST`, the highest among all configurations.
- **Dataset Behavior Differences:** `FashionMNIST` again proved to be more susceptible to compression, since higher pruning ratios were achieved without loss of performance. This aligns with earlier findings and confirms that grayscale, lower-resolution datasets can tolerate more aggressive pruning.
- **Performance vs. Size Tradeoff:** Unlike in the original results, our pruned `AlexNet` not only retained but even improved its accuracy on both datasets. This could be due to different initialization or fine-tuning strategies post-pruning, implying that retraining methodology has a strong influence on final performance.
- **ShuffleNet Sensitivity:** Interestingly, our pruned `ShuffleNet` showed an improvement in accuracy, especially on `CIFAR-10`, despite a reduction in parameters. This deviates from the original trend and suggests that compact networks might have redundant channels even in low-parameter settings.

Table 2: **Our** Network pruning on CIFAR-10 and FashionMNIST

Network	**Accuracy (%)**		**Total Parameters**		**Compression Rate**	
	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST
MobileNet – Reference	88.54	90.23	2.2M	2.2M	–	–
MobileNet – Pruned	88.53	88.67	671K	1.1M	1.6×	3.3×
ShuffleNet – Reference	84.94	89.71	1.2M	1.2M	–	–
ShuffleNet – Pruned	87.95	89.91	879K	815K	1.4×	1.5×
AlexNet – Reference	85.62	91.81	57M	57M	–	–
AlexNet – Pruned	87.89	91.93	28M	5M	2×	10×

4.3 Per-Channel Quantisation

Tables 3 and 4 examine the impact of p**er-channel quantization** when applied to the **already pruned networks**, showing its effects on accuracy, model size, and inference time.

4.3.1 Their Results

- **Size and Speed Benefits for `MobileNet` and `AlexNet`:** Applying quantization to already pruned models of `MobileNet` and `AlexNet` resulted in a considerable reduction in physical model size and a significant decrease in inference time, with only minor accuracy loss. MobileNet, for example, saw up to a 7.3x speedup.
- **`ShuffleNet`'s Poor Response:** Interestingly, `ShuffleNet` on `CIFAR-10` did not respond well to quantization. It suffered a major accuracy loss and, unexpectedly, an increase in inference time. We believe this may be the result of its more complex architecture, which makes it less suitable for this type of quantization, leading to overhead.

4.3.2 Our Results

- **Compression Consistency:** Similar to the reference study, per-channel quantization consistently reduced model size across all architectures. `MobileNet` and `ShuffleNet` achieved the most compact representations (down to 2.5MB and 1.4MB respectively), which is highly beneficial for deployment on resource-constrained devices.

Table 3: **Their** Network quantisation of pruned models trained on CIFAR-10 and FashionMNIST

Network	Accuracy (%)		Size (MB)		Inference Time (ms)	
	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST
MobileNet – Reference	91.31	90.75	8.7	8.7	34.80	1.70
MobileNet – Quantised	90.59	90.07	**2.9**	**2.9**	**4.74**	**0.30**
ShuffleNet – Reference	93.36	90.36	4.9	4.9	11.67	0.73
ShuffleNet – Quantised	81.29	89.78	**1.8**	**1.8**	23.15	**0.61**
AlexNet – Reference	93.54	91.61	217.6	217.6	22.13	6.70
AlexNet – Quantised	90.06	90.27	**54.6**	**54.6**	**5.23**	**4.90**

- **Inference Time Improvements:** We observed considerable speedups after quantization, especially for `MobileNet`, which went from 34.91ms to just 4.51ms on `CIFAR-10`. This confirms quantization's practical impact on runtime performance for real-time inference scenarios.
- **Accuracy Stability:** Unlike in the original results, our quantized models did not experience substantial accuracy drops. Notably, `ShuffleNet` even saw a slight increase in accuracy post-quantization, which could be attributed to numerical stabilization effects during quantization-aware retraining.
- **Unexpected `AlexNet` Behavior:** While `AlexNet`'s accuracy remained mostly stable, its inference time surprisingly increased after quantization (e.g., 24.53ms → 39.43ms on `CIFAR-10`). This may be due to backend or hardware-specific inefficiencies when handling quantized large-layer models, indicating that quantization benefits are not guaranteed for every architecture.
- **`FashionMNIST` vs. `CIFAR-10`:** As seen before, `FashionMNIST` allowed for greater resilience to quantization, with all models retaining or even improving their accuracy. CIFAR-10 results were also stable, but inference times fluctuated more across networks, possibly due to color image complexity interacting differently with quantized kernels.

Table 4: **Our** Network quantisation of pruned models trained on CIFAR-10 and FashionMNIST

Network	Accuracy (%)		Size (MB)		Inference Time (ms)	
	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST	CIFAR-10	FashionMNIST
MobileNet – Reference	88.54	90.23	8.7	8.7	34.91	13.03
MobileNet – Quantised	88.46	90.1	**2.51**	**2.51**	**4.51**	**4.04**
ShuffleNet – Reference	84.97	89.71	4.9	4.9	27.88	13.3
ShuffleNet – Quantised	**85.19**	**89.79**	**1.4**	**1.4**	**7.8**	**7.51**
AlexNet – Reference	85.62	91.81	217	217	24.53	10.99
AlexNet – Quantised	**85.64**	91.63	**54.6**	**54.6**	39.43	28.7

4.4 Overfitted and Pruned Networks

Tables 5 and 6 explore the efficiency of **pruning as a regularization technique** to try to correct overfitting behaviors in networks trained on `FashionMNIST`.

4.4.1 Their Results

- **Effective Regularization:** Pruning proved to be an effective regularization technique to help reduce the overfitting problem. Even when networks were intentionally overfitted, applying pruning resulted in higher test accuracy than the overfitted versions, improving the network's ability to generalize.
- **Architecture-Agnostic Gains:** The performance boost from pruning was consistent across all tested architectures - `MobileNet`, `ShuffleNet`, and `AlexNet` - demonstrating that the regularization effect is not model-specific and can generalize across both lightweight and heavy models.

- **Parameter Reduction with Performance Gains:** Not only did pruning reduce overfitting, but it also cut down the total number of parameters significantly (e.g., `AlexNet` from 57M to 22M), showcasing its dual benefit of improving generalization while optimizing model size.
- **Improved Generalization Beyond Reference Models:** Surprisingly, the pruned networks outperformed even their non-overfitted reference counterparts. This suggests that careful pruning can lead to better generalization than traditional training alone, positioning it as a competitive strategy even outside of explicit overfitting scenarios.

Table 5: **Their** Overfitted and pruned networks trained on FashionMNIST

Network	Accuracy (%)	Total Parameters
MobileNet – Reference	90.75	2.2M
MobileNet – Overfitted	90.55	2.2M
MobileNet – Pruned	**91.26**	1.76M
ShuffleNet – Reference	90.36	1.2M
ShuffleNet – Overfitted	89.71	1.2M
ShuffleNet – Pruned	**91.01**	1M
AlexNet – Reference	91.61	57M
AlexNet – Overfitted	91.32	57M
AlexNet – Pruned	**92.11**	22M

4.4.2 Our Results

- **Moderate Regularization Effect:** Pruning provided a mild regularization benefit in our experiments. `MobileNet` slightly improved post-pruning (90.60% $\rightarrow$ 90.79%), while also reducing parameter count by 20%. Though the gains were small, the results suggest that pruning does help mitigate overfitting in compact models.
- **ShuffleNet Stability:** Interestingly, `ShuffleNet` exhibited very stable accuracy across all settings. While the overfitted version slightly outperformed the reference (89.98%), the pruned version maintained nearly the same accuracy (89.95%), indicating that pruning did not hurt generalization but also didn't offer significant improvement. This stability suggests a potential robustness of ShuffleNet to both overfitting and structural changes.
- **Strong Results for `AlexNet`:** The most notable improvement was seen in `AlexNet`, where the overfitted network improved to 92.65%, and pruning pushed it even further to 92.74%, with a substantial reduction in parameters (from 57M to 22M). This supports the hypothesis that large models benefit most from pruning as a post-overfitting corrective mechanism.
- **Generalization via Simplification:** Across all architectures, pruning consistently reduced the number of parameters without harming accuracy. In two out of three cases, accuracy improved post-pruning, supporting the idea that removing redundant weights helps guide the model toward simpler and more generalizable representations.

5 Discussion of Results and Article Reproduction

The process of reproducing the results from the paper revealed significant discrepancies between the results obtained in our experiment and those reported in the original article. While certain metrics, such as the total number of parameters and model sizes, were relatively consistent, the key performance metrics, particularly model accuracy, showed considerable variance. In this section we will try to analyze the potential reasons for these gaps and the lessons we learned from the process.

Table 6: **Our** Overfitted and pruned networks trained on FashionMNIST

Network	Accuracy (%)	Total Parameters
MobileNet – Reference	90.23	2.2M
MobileNet – Overfitted	90.6	2.2M
MobileNet – Pruned	**90.79**	1.76M
ShuffleNet – Reference	89.71	1.2M
ShuffleNet – Overfitted	**89.98**	1.2M
ShuffleNet – Pruned	89.95	1M
AlexNet – Reference	91.81	57M
AlexNet – Overfitted	92.65	57M
AlexNet – Pruned	**92.74**	22M

5.1 Environment Reproduction Challenges

One of the primary obstacles in the reproduction process was the lack of documentation for the original work environment. The authors of the paper did not provide basic things for execution like `requirements.txt` or `README.md` files, which specifies the versions of the libraries used, and how to run it. This lack of information led to several difficulties:

- **Variance in `PyTorch` Versions:** The `PyTorch` library changes between versions. The implementation of existing architectures, such as `MobileNet` and `ShuffleNet`, has changed in the `torchvision` library between the version used by the authors and the one we used. Such changes can affect the training process and final performance.
- **Bug Fixes in the Original Code:** During our work, we encountered a bug in the provided source code that prevented it from running with any `PyTorch` version we tested (from 1.0 to 1.8). We had to fix the code ourselves, which likely introduced another deviation from the original code run by the authors.

5.2 Uncertainty Regarding Hyperparameters

The paper did not detail all the hyperparameters used for training the models. Parameters such as the learning rate schedule, the optimizer's `momentum` and `weight decay` values, and the specific settings for the data augmentation process remained unknown. In the absence of this information, we used default values, which could be different from the original settings and constitute a major factor for the variance in the accuracy results.

5.3 Inconsistencies in the Original Paper

A careful examination of the results in the original paper revealed internal contradictions. For example, for the `ShuffleNet` network on the `CIFAR-10` dataset, Table 1 shows an accuracy of 93.05% after pruning 30% of the parameters, while the corresponding graph in Figure 1 in the same paper shows an accuracy of only around 50% for the same configuration. This inconsistency undermines the reliability of the original results and makes it difficult to define a clear, valid target for reproduction.

5.4 Comparison of Additional Performance Metrics

- **Model Size and Number of Parameters:** As expected, we were able to consistently reproduce the total number of parameters and the model sizes (in MB). These metrics depend on the architecture itself and are less sensitive to the runtime environment, which indicates a correct understanding of the network structures and compression processes.
- **Inference Time:** The gaps in inference times were anticipated and are due to differences in hardware (CPU, GPU) and software versions (CUDA, cuDNN) between our machine and the one used in the original study.

6 Code

The original implementation from the referenced paper is available here. Our modified version, which includes adjustments to support newer architectures, is available here.

References

[1] Kimessha Paupamah, Steven James, and Richard Klein. Quantisation and pruning for neural network compression and regularisation, 2020. URL `https://arxiv.org/abs/2001.04850`. arXiv preprint arXiv:2001.04850.