Pruning and Sparsity

SAiDL 2024 Assignment

Sasmit Datta

1 Introduction

In the pursuit of creating more efficient and scalable deep learning models, pruning stands out as an essential technique to reduce model size and computational complexity without significantly compromising accuracy. Along with **fine-grained** pruning on a CNN trained on the CIFAR-10 dataset, I implement **kernel** and **filter** pruning and present a comprehensive analysis on the same.

2 Pruning Strategies

All the pruning strategies are implemented using magnitude based approaches - more specifically by calculating the norm of a weight or a series of weights.

2.1 Fine-grained

Fine-grained pruning, often referred to as unstructured pruning, involves the selective removal of individual weights across the network based on their magnitudes.

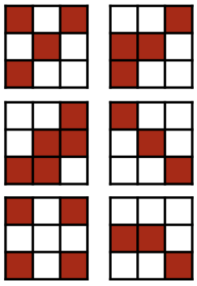


Figure 1: Fine-grained Pruning

Provided sparsity value $s\%$ for a particular layer, $s\%$ of weights with lower absolute values are pruned.

$$\text{Importance} = |W| \tag{1}$$

2.2 Kernel

Kernel pruning extends the principle of fine-grained pruning from individual weights to entire kernels within convolutional layers based on the norm of each kernel.

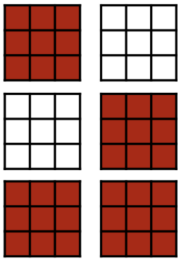


Figure 2: Kernel Pruning

We first take each kernel and calculate their frobenius norms.

$$||\mathbf{A}||_F = \sqrt{\sum_{i,j} a_{ij}^2} \tag{2}$$

Now provided a sparsity value $s\%$ for a particular layer, $s\%$ of kernels with lower frobenius norms are pruned.

$$\text{Importance} = ||\mathbf{W}_k||_F \tag{3}$$

where, $\mathbf{W}_k$ is a kernel of a particular layer.

2.3 Filter

Filter pruning takes a more holistic approach by evaluating and eliminating entire filters from convolutional layers. This method not only prunes weights but also reduces the overall depth of the model.

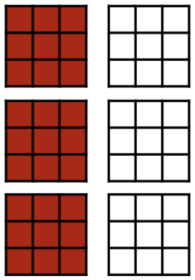


Figure 3: Filter Pruning

Since filters are tensors, we first take each filter and unroll them to a 2-D matrix. We then calculate the frobenius norms of each of these matrices and prune them based on the magnitudes of their norms.

$$\text{Importance} = ||\mathbf{W}_{\text{flattened}}||_F \tag{4}$$

where $||\mathbf{W}_{\text{flattened}}||_F$ is the flattened 2-D matrix of a filter.

3 Model

3.1 Architecture

The architecture of the model used resembles a VGG-16 architecture with 4 convolutional blocks and a feed forward network as its head. In Fig 4, each convolutional layer has a **kernel size** of 3×3, **stride** of 1 and **padding** of 1.

All the convolutional layers in Fig 5 are named from 'conv0 to 'conv11'. So there are a total of 12 convolutional layers.

3.2 Hyper-parameters

Some of the common hyper-parameters I used to train the un-pruned model and fine-tune the pruned ones are the following:

- Optimizer: **Adam** with default Pytorch settings except for the learning rate.
- Learning Rate = 0.0001
- Number of Parameters = 8,391,882

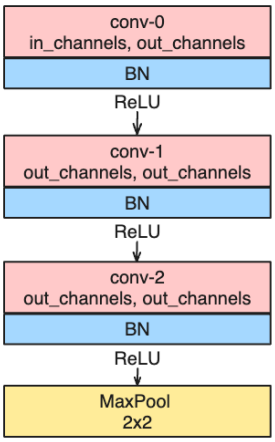


Figure 4: A single convolutional block

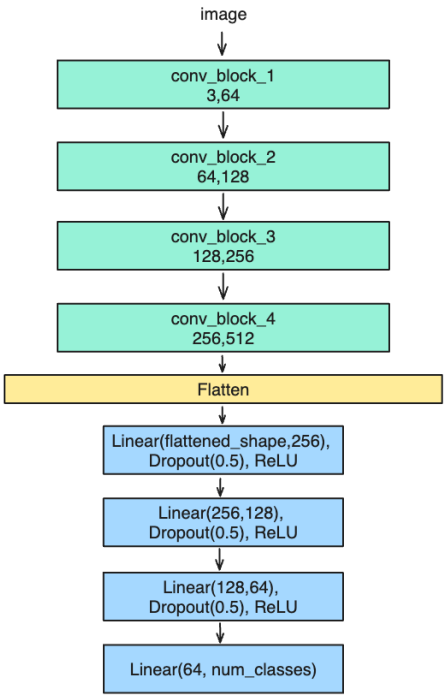


Figure 5: CNN Architecture

- Batch Size = 64

I trained the **un-pruned model** for **100 epochs** while the **pruned models** were fine-tuned for **20 epochs** each.

4 Sensitivity Scanning

Sensitivity scanning refers to a systematic approach for identifying how sensitive different layers or components of a neural network are to pruning. The primary goal of sensitivity scanning is to determine which weights, neurons, or layers can be removed with minimal impact on the model's performance.

4.1 Implementation

For each pruning technique, I iterate over the weight tensor of each convolutional layer, prune them with sparsity values ranging from 0 to 0.9 with a step size of 0.1 and plot the accuracy of each instance of pruning.

4.2 Results

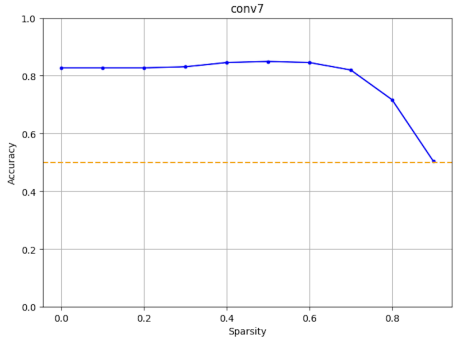


Figure 6: Sensitivity Scan

For all the sensitivity scans, we can see that the initial convolutional layers are more important than the latter ones even though they have significantly lesser parameters (Fig 7). In some instances the accuracy doesn't even budge before the 0.7 or 0.8 range while pruning these deeper layers (Fig 8). All graphs are in Section 10.1.

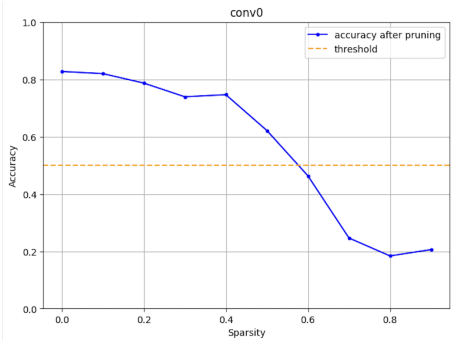


Figure 7: Sensitivity Scan of an Earlier Layer

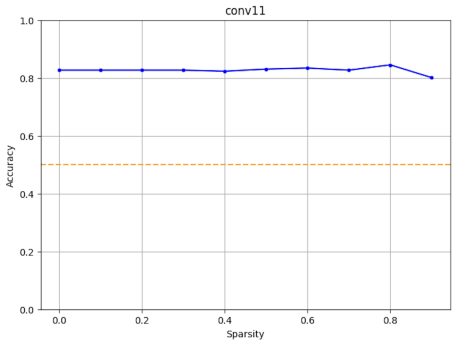


Figure 8: Sensitivity Scan of a Deeper Layer

The first few layers are responsible for capturing low level features that are important for subsequent layers to build upon. Pruning can lead to failure of detecting key features which can drop accuracy significantly.

Since the initial layers have lesser parameters, when pruned there are lesser parameters to compensate for the loss in performance.

The deeper layers may be over parametrised, meaning they have more parameters than necessary, hence leading to less loss of performance when pruned.

Also note that for some layers, the accuracy increases when its pruned. This could be due to the fact that the more the model we prune, the more it is able to generalise (I am plotting accuracy of these pruned models on the test dataset).

5 Pruning Heuristic

After performing the sensitivity scans and going over the graphs, I decided to assign a **sparsity value** to each layer just before they dipped below the **50% accuracy mark**. The sparsity levels are displayed in Table 1.

Layer	Fine-grained	Kernel	Filter
conv0	0.5	0.3	0.4
conv1	0.8	0.6	0.3
conv2	0.7	0.3	0.3
conv3	0.7	0.5	0.3
conv4	0.7	0.5	0.2
conv5	0.8	0.7	0.4
conv6	0.7	0.6	0.2
conv7	0.8	0.6	0.4
conv8	0.8	0.7	0.6
conv9	0.8	0.8	0.5
conv10	0.9	0.9	0.6
conv11	0.9	0.9	0.9

Table 1: Sparsity Levels for Different Pruning Methods

6 Results

As discussed before I trained the **un-pruned model** for **100 epochs** while the **pruned models** were fine-tuned for **20 epochs** each.

The **accuracy** of the **un-pruned model** on the test dataset is **85.03%**.

Strategy	% Parameters Remaining	Accuracy After Pruning(%)	Accuracy After Fine-tuning(%)
Fine-grained	20.4959	**40.81**	**82.69**
Kernel	23.9542	17.94	80.70
Filter	41.4917	14.05	82.44

Table 2: Impact of Pruning Strategies on Model

Fine-grained pruning retains the most accuracy after being applied while filter saw the most drop.

Since fine-grained pruning operates at a granular level, it can often really narrow in on each individual weight and gauge their importance to the network. On the other hand, filter and kernel pruning removes entire tensors all at once which can lead to important feature detectors being removed.

Fine-grained pruners are like expert sculptors, having justified intention to chisel away each flake of wood to perfect his creation while kernel and filter pruners are amateurs sculptors making broader strokes, that may shape the figure quickly but risk losing some finer, nuanced detail.

Although, the kernel and filter pruning strategies bring the down the accuracies by a significant margin, all three models are easily able to gain their performance back with proper fine-tuning.

7 Weight Distributions

I plotted the weights of all the convolutional layers of seven different variations of the model. The un-pruned model and models on which the three pruning strategies are applied, both before and after fine-tuning. All the distributions are in Section 10.2.

7.1 Un-pruned

Fig 9 is the weight distribution of conv7 of the trained model.

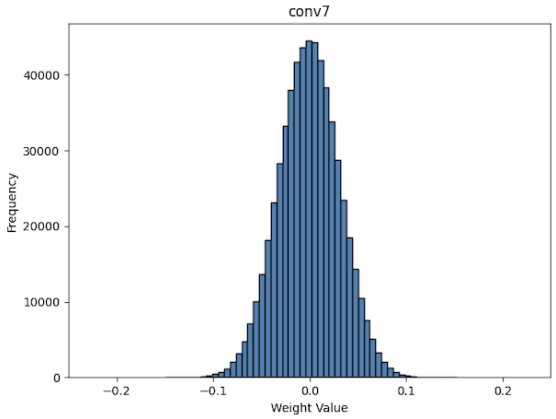
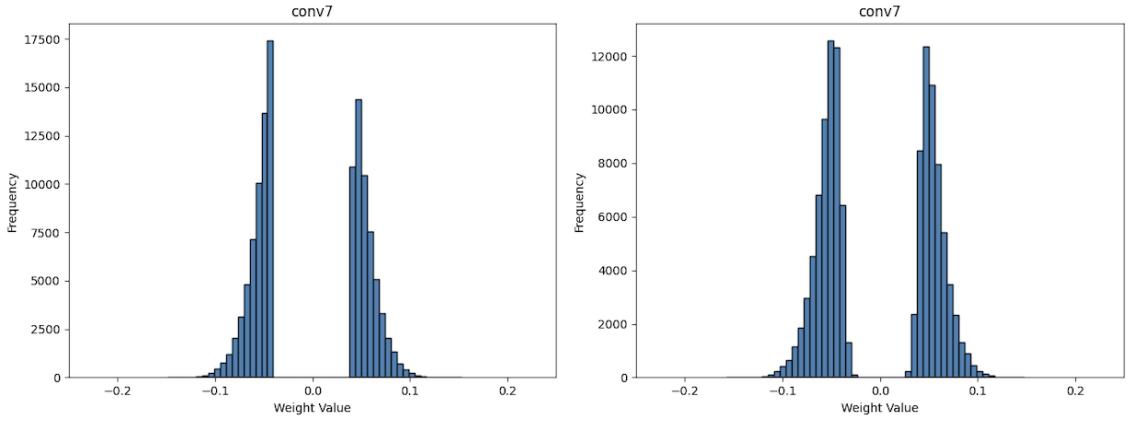


Figure 9: Distribution of Weights of a Layer in the Un-pruned Model

7.2 Fine-grained Pruning

Since fine-grained pruning involves pruning based on magnitude, we see that weights with values near zero being heavily filtered out. The 'conv7' layer for fine-grained pruning has a sparsity value of 0.8, so there is a significant decrease in number of weights as well. (Fig 10)



(a) Pre-Fine-tuning

(b) Post-Fine-tuning

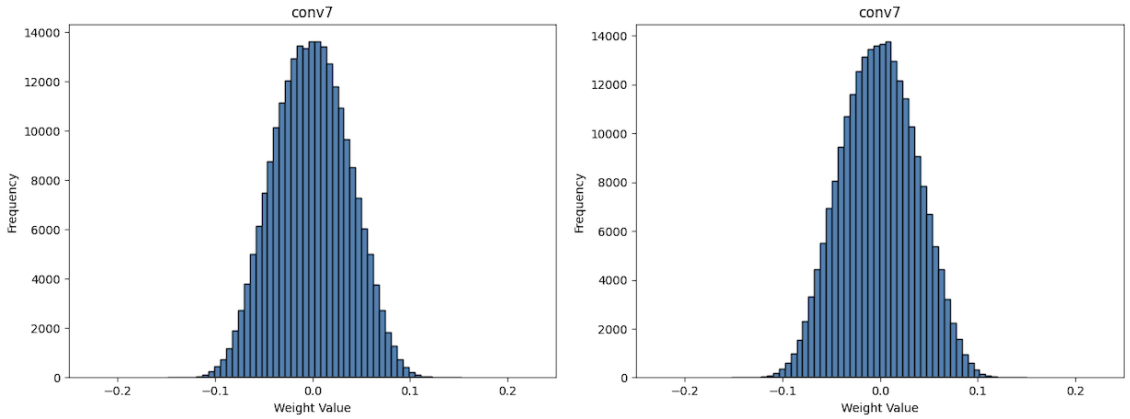
Figure 10: Fine-grained Pruning Weight Distribution

The distribution of the pre-fine-tuned weights have high bias due to most of the values near zero being removed. When we fine-tune, the weights gain some variance to compensate for learning capacity but never matches the distribution of the un-pruned model.

7.3 Kernel Pruning

After kernel pruning, the tail ends of the distribution tends to gain some variance in its weights alluding to the fact that weights closer to zero have been removed but not as aggressively as fine-grained pruning. (Fig 11)

Fine-tuning doesn't change the distribution much. It just re-purposes existing weights to fit the task.

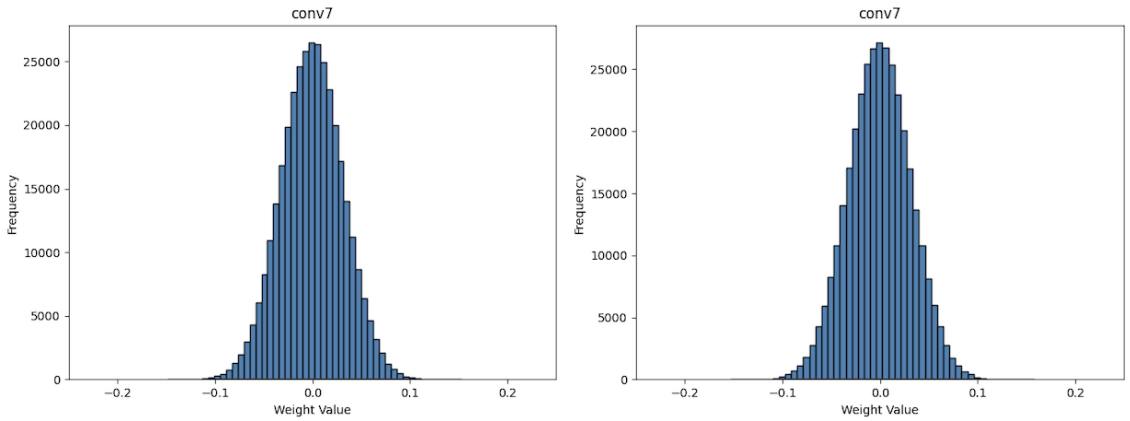


(a) Before Fine-tuning (b) After Fine-tuning

Figure 11: Kernel Pruning Weight Distribution

7.4 Filter Pruning

Distribution of weights remains the most constant with the original distribution. Since we are using much larger heuristic to prune weights, the number of weights actually closer to zero becomes significantly lower. Hence, very little change to the distribution. (Fig 12)



(a) Before Fine-tuning (b) After Fine-tuning

Figure 12: Filter Pruning Weight Distribution

8 Activation Balances

I passed Fig 13 into all the models and plotted a histogram of the activations. I pruned all the activations that equalled zero for each layer. All the distributions are

in Section 10.3.

Figure 13: Input Image to the Models

9 Conclusion

We can see most performance being perserved by fine-grained pruning but its not much of a viable method because of how unstructured it is. GPUs can have hard time navigating through it. Kernel and channel pruning are quite structured and regain most of its performance in this report, but rigorous testing is required on how far can we go with them, how important the weights being pruned are, and whether using norms are a good method for these pruning strategy.

10 Appendix

10.1 Sensitivity Scans

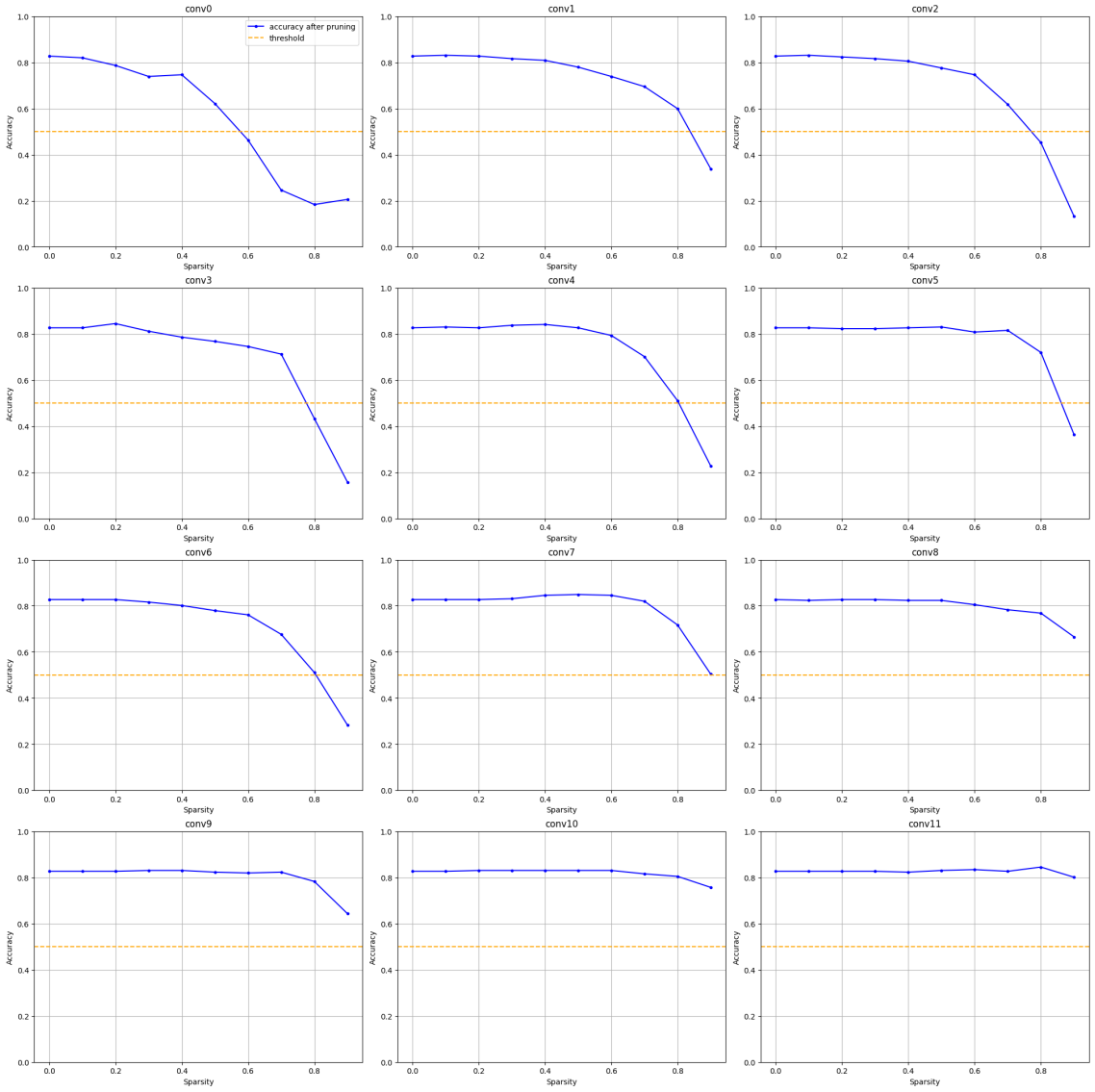


Figure 14: Fine-grained Pruning Sensitivity Scan

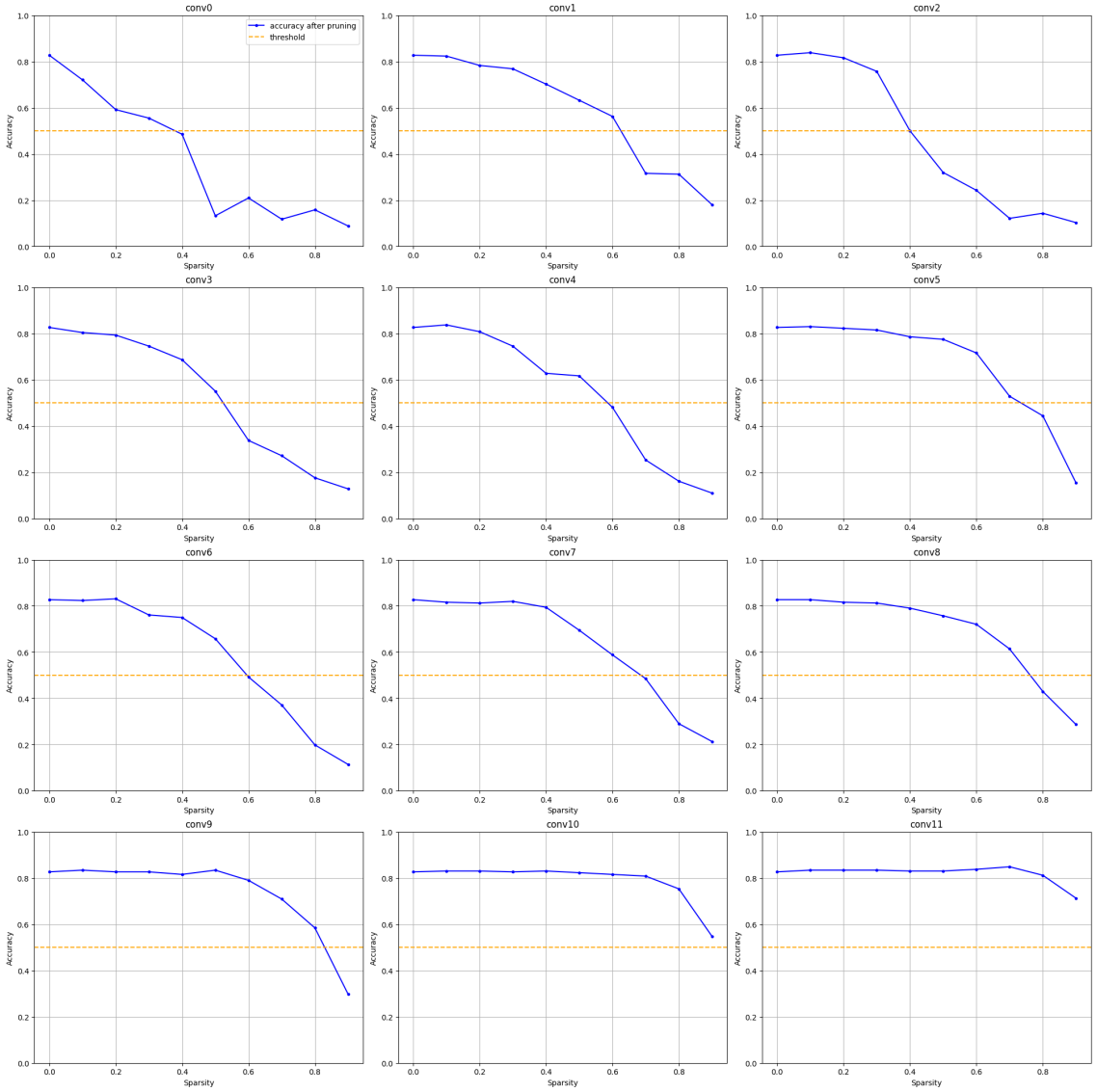


Figure 15: Kernel Pruning Sensitivity Scan

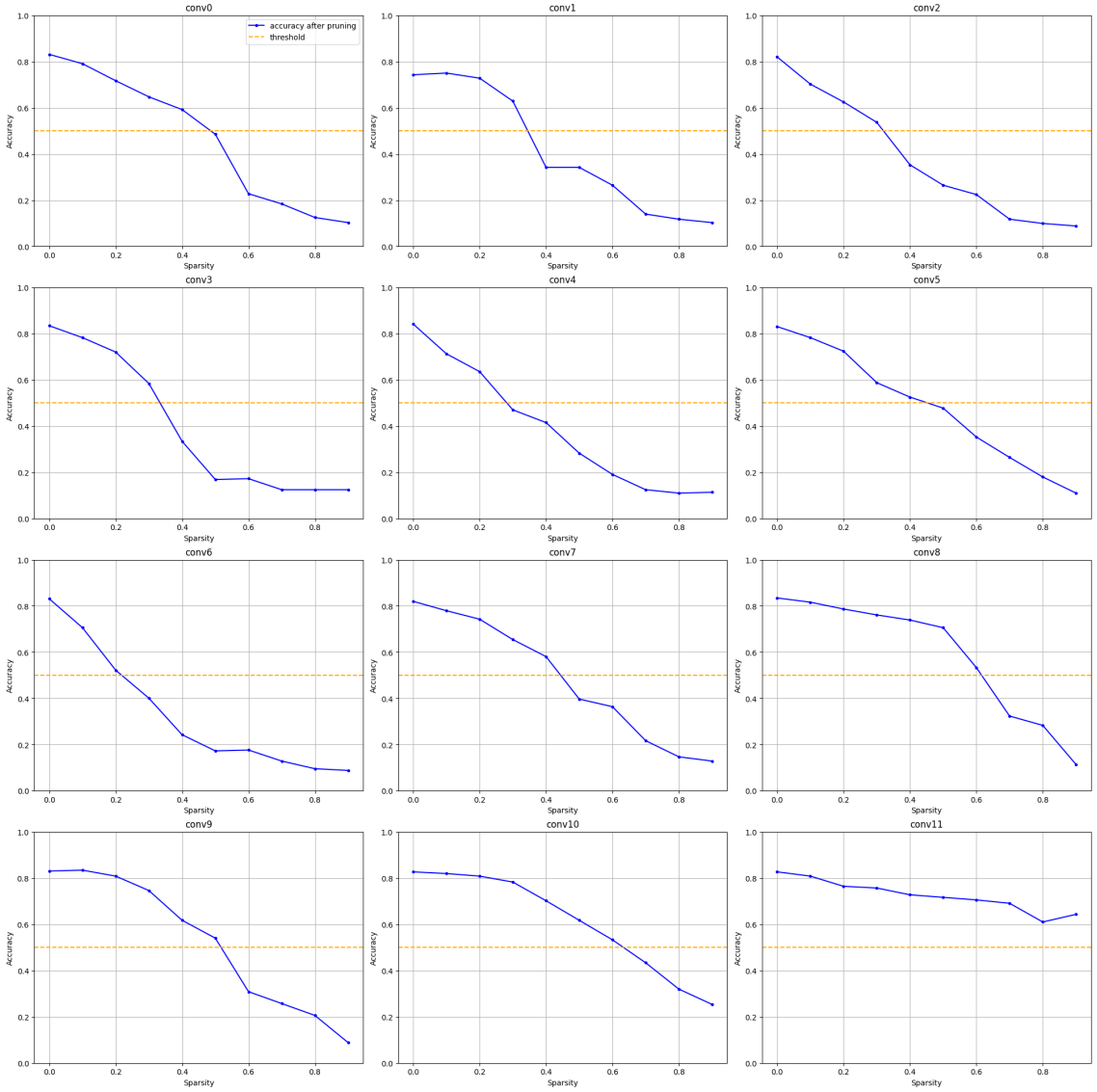


Figure 16: Filter Pruning Sensitivity Scan

10.2 Weight Distributions

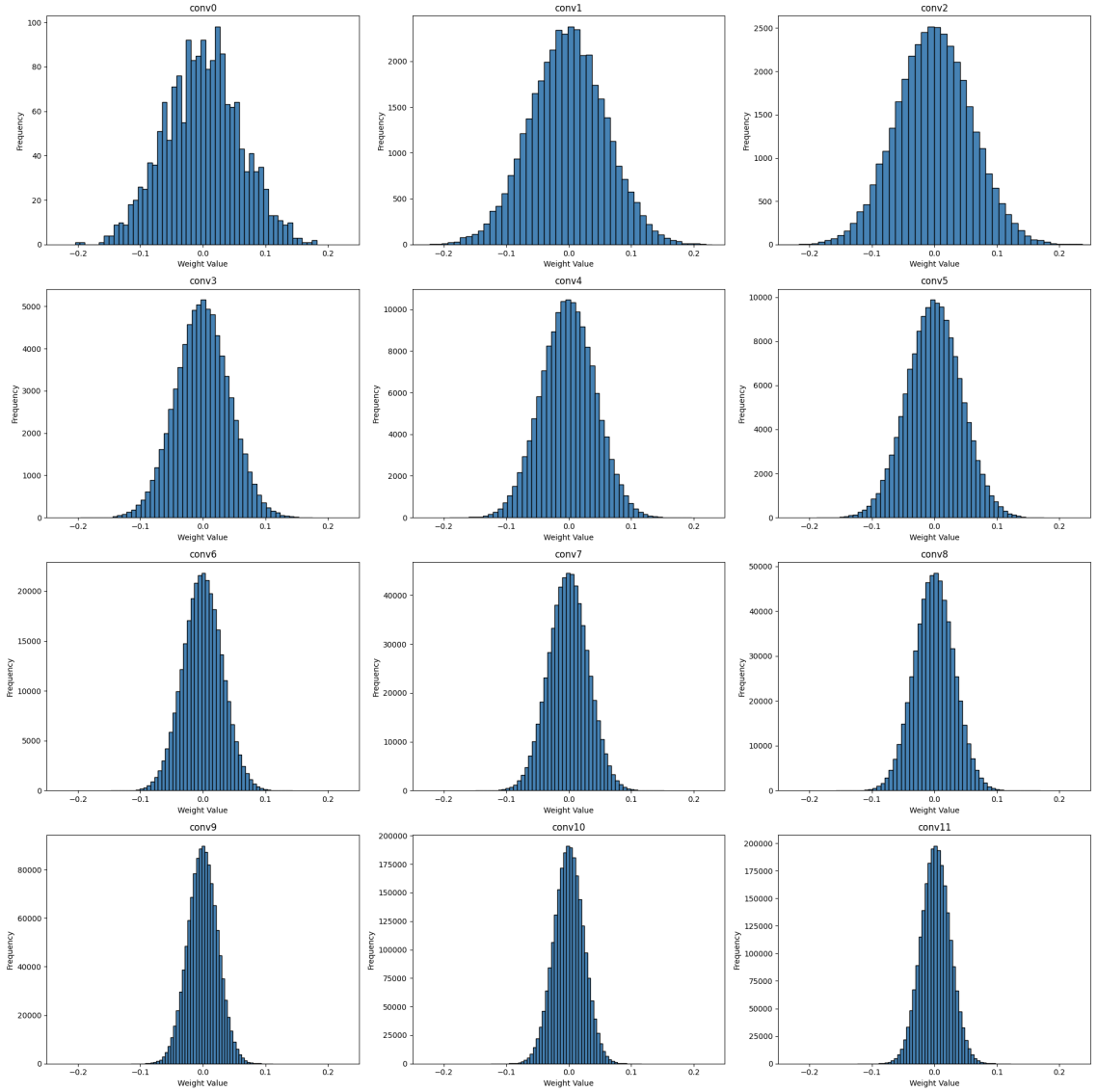


Figure 17: Un-pruned Model Weights Distribution

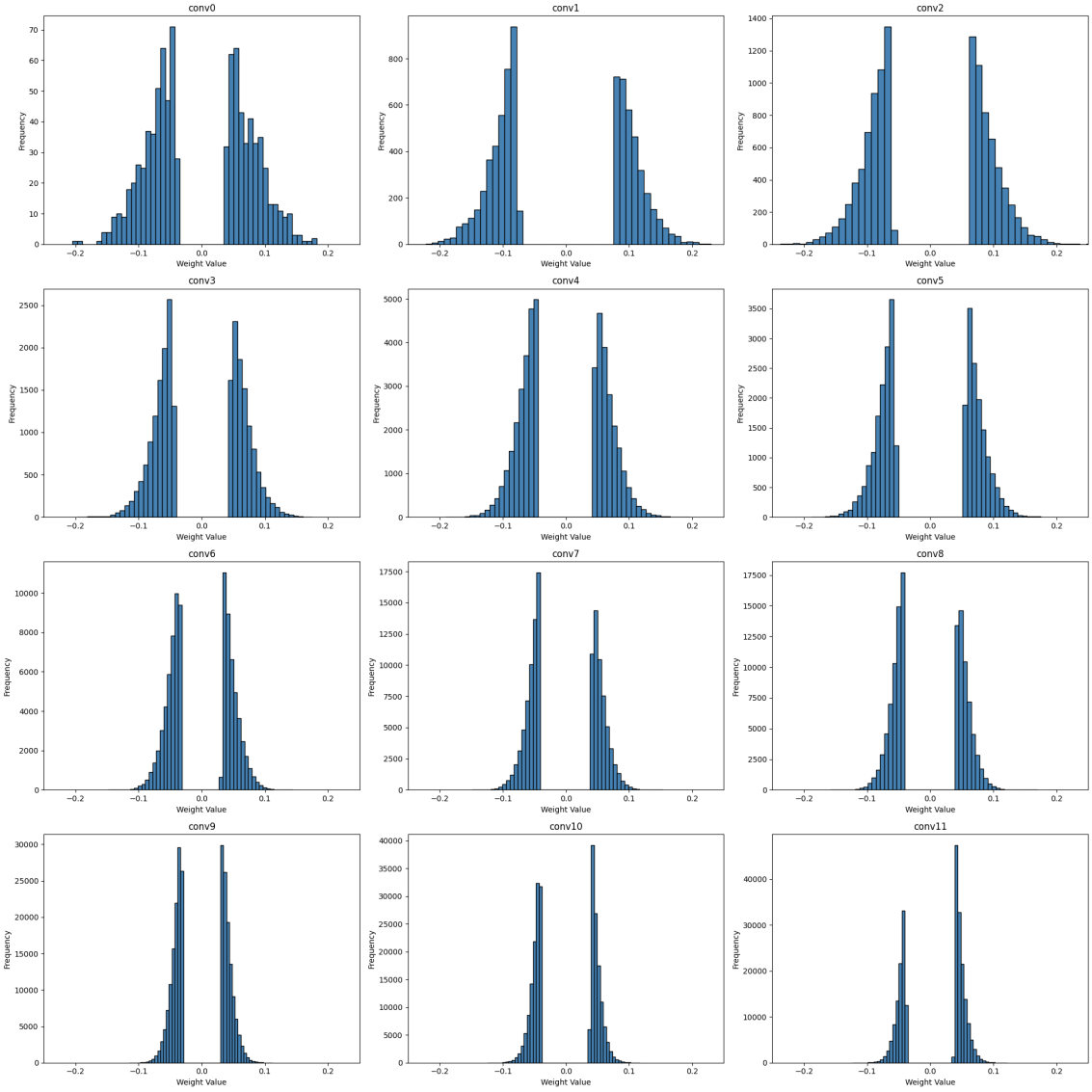


Figure 18: Fine-grained Pruned Model Weights Distribution (Pre-Fine-tuning)

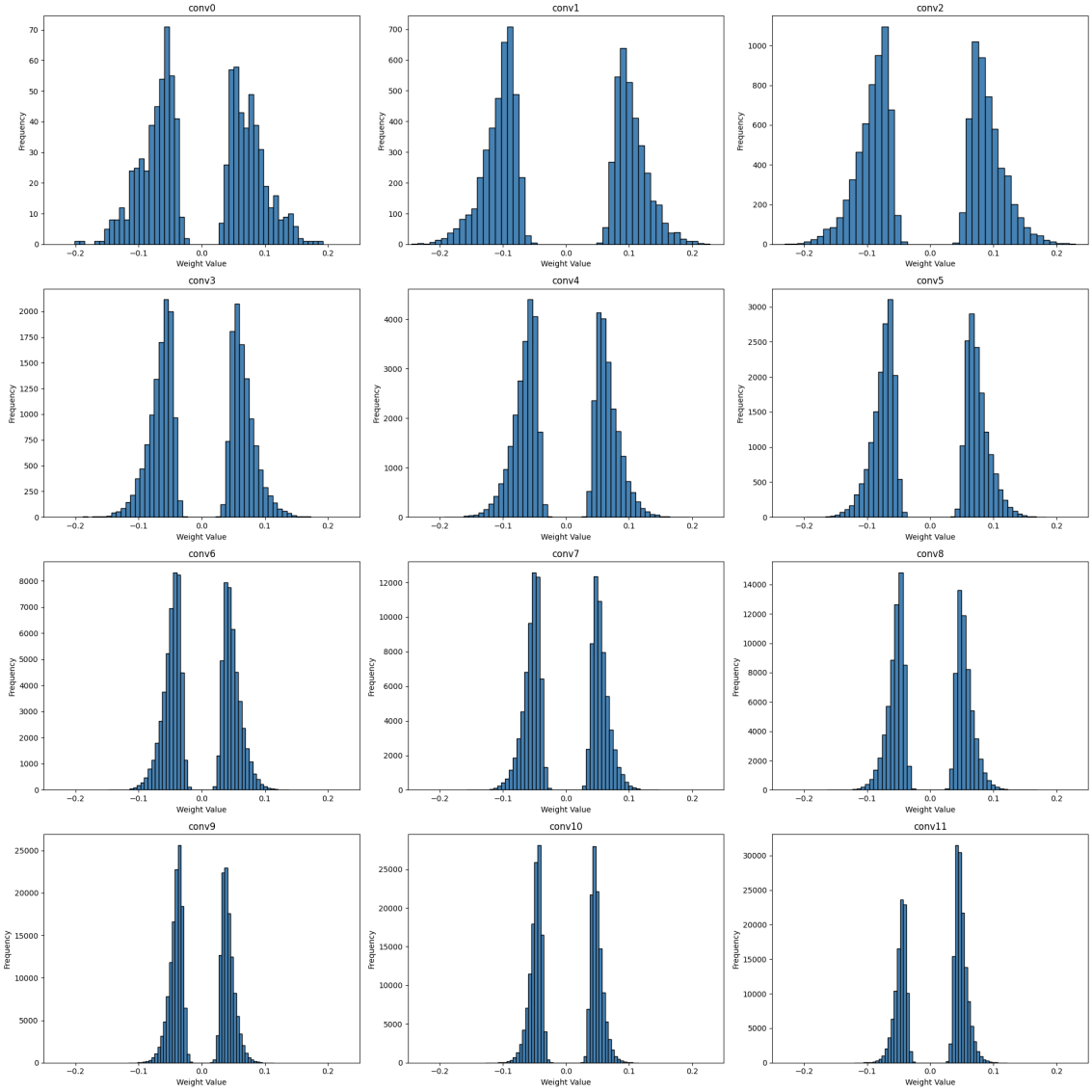


Figure 19: Fine-grained Pruned Model Weights Distribution (Post-Fine-tuned)

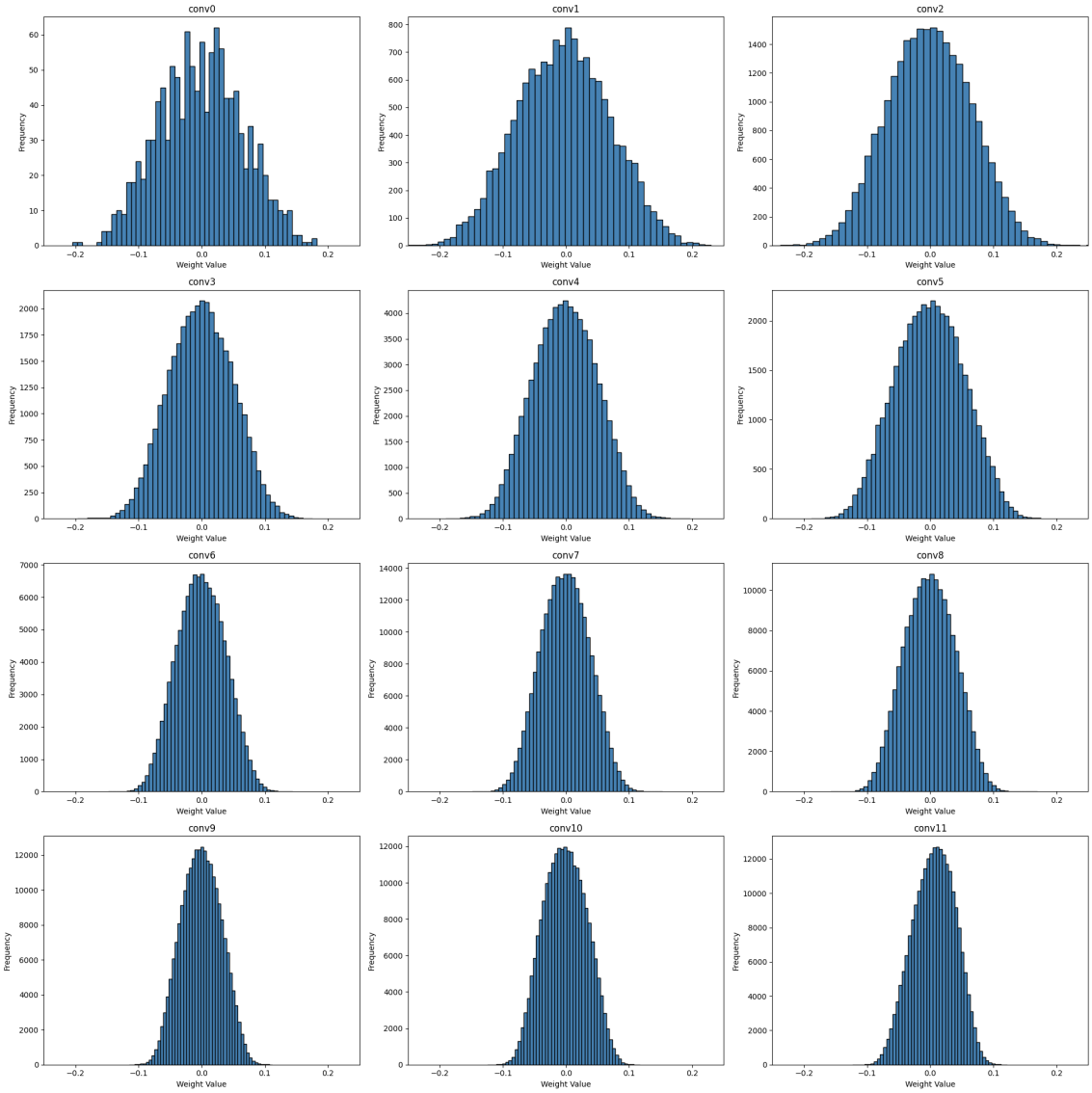


Figure 20: Kernel Pruned Model Weights Distribution (Pre-Fine-tuning)

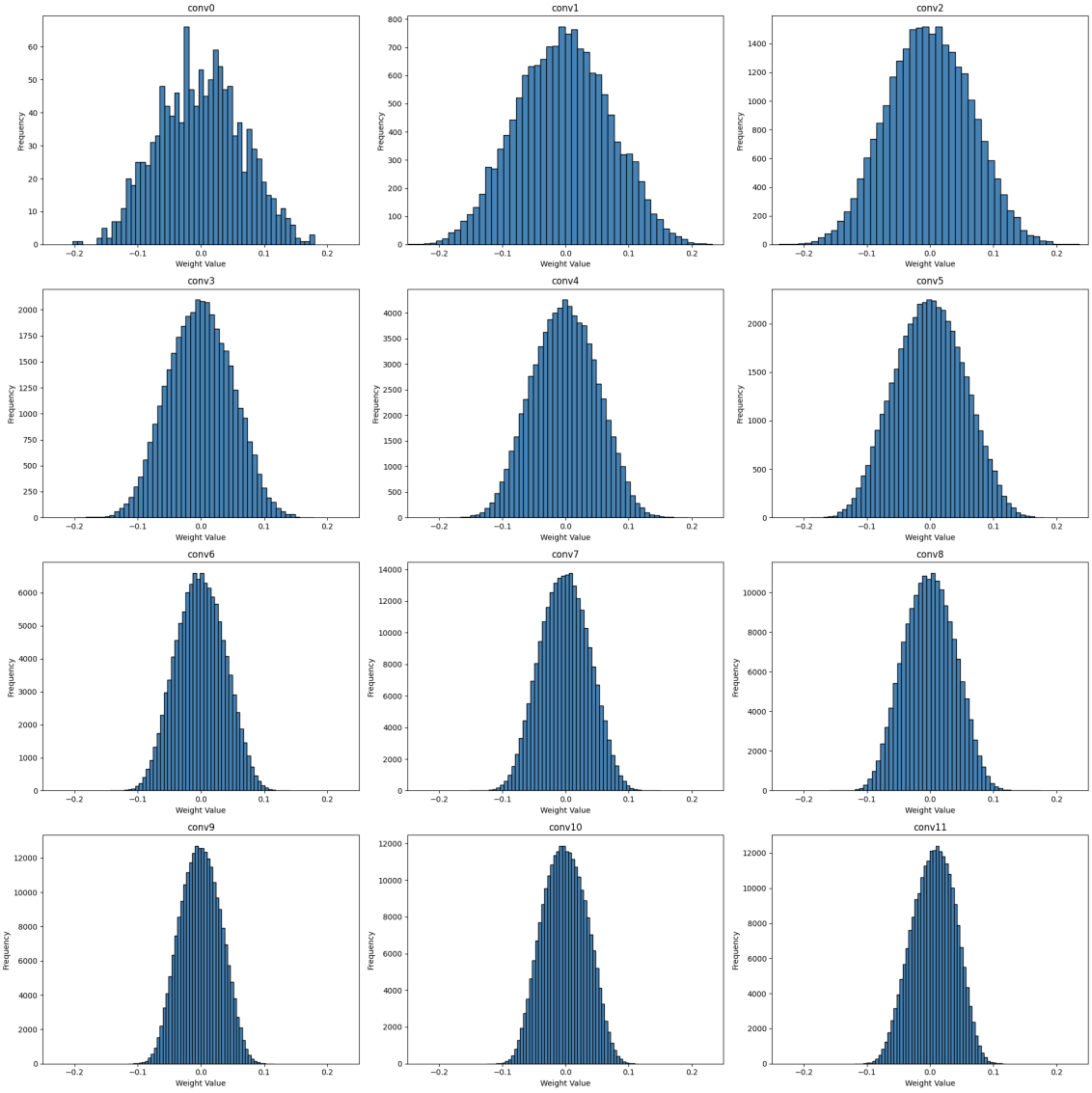


Figure 21: Kernel Pruned Model Weights Distribution (Post-Fine-tuning)

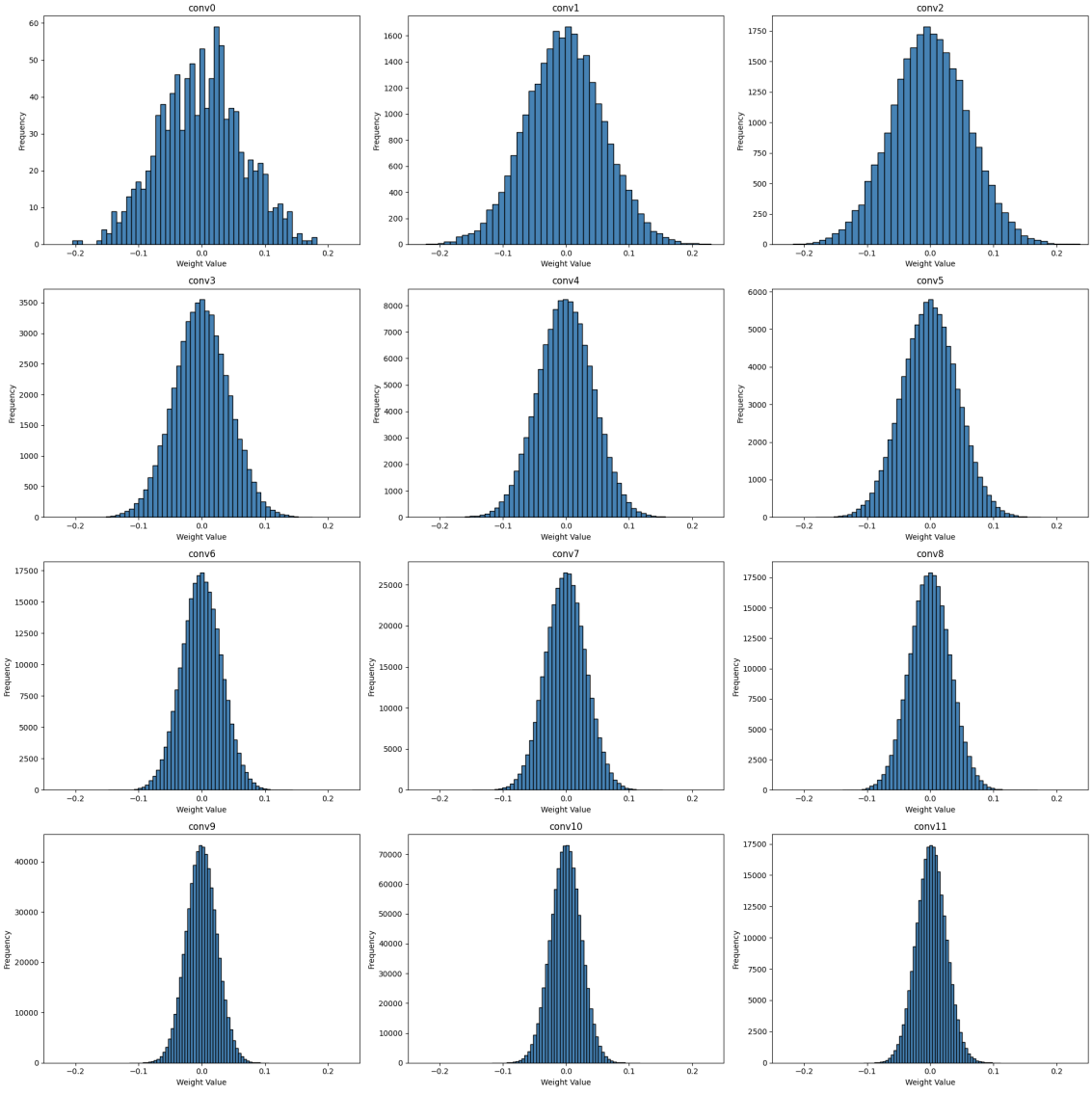


Figure 22: Filter Pruned Model Weights Distribution (Pre-Fine-tuning)

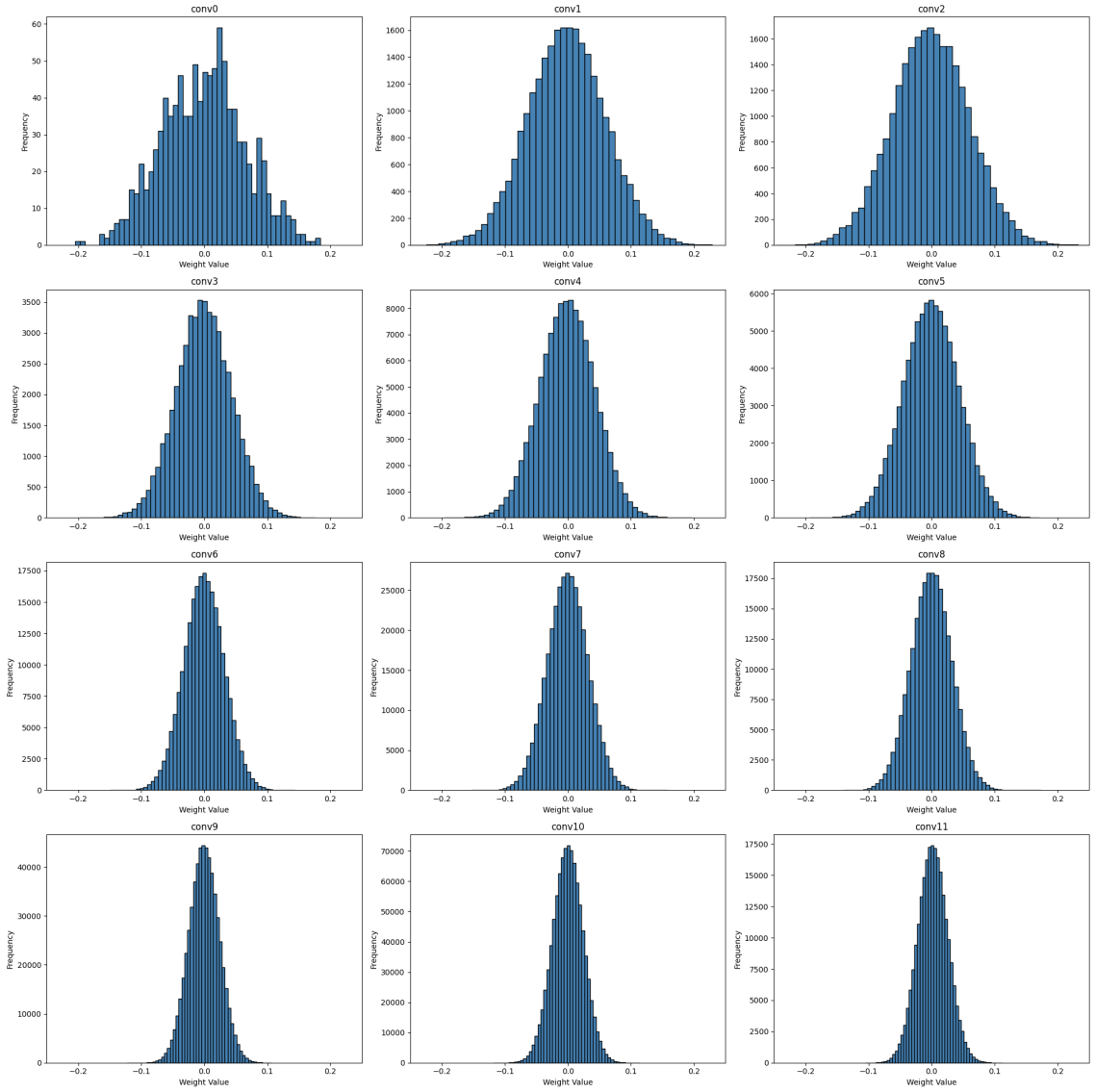


Figure 23: Filter Pruned Model Weights Distribution (Post-Fine-tuning)

10.3 Activation Balances

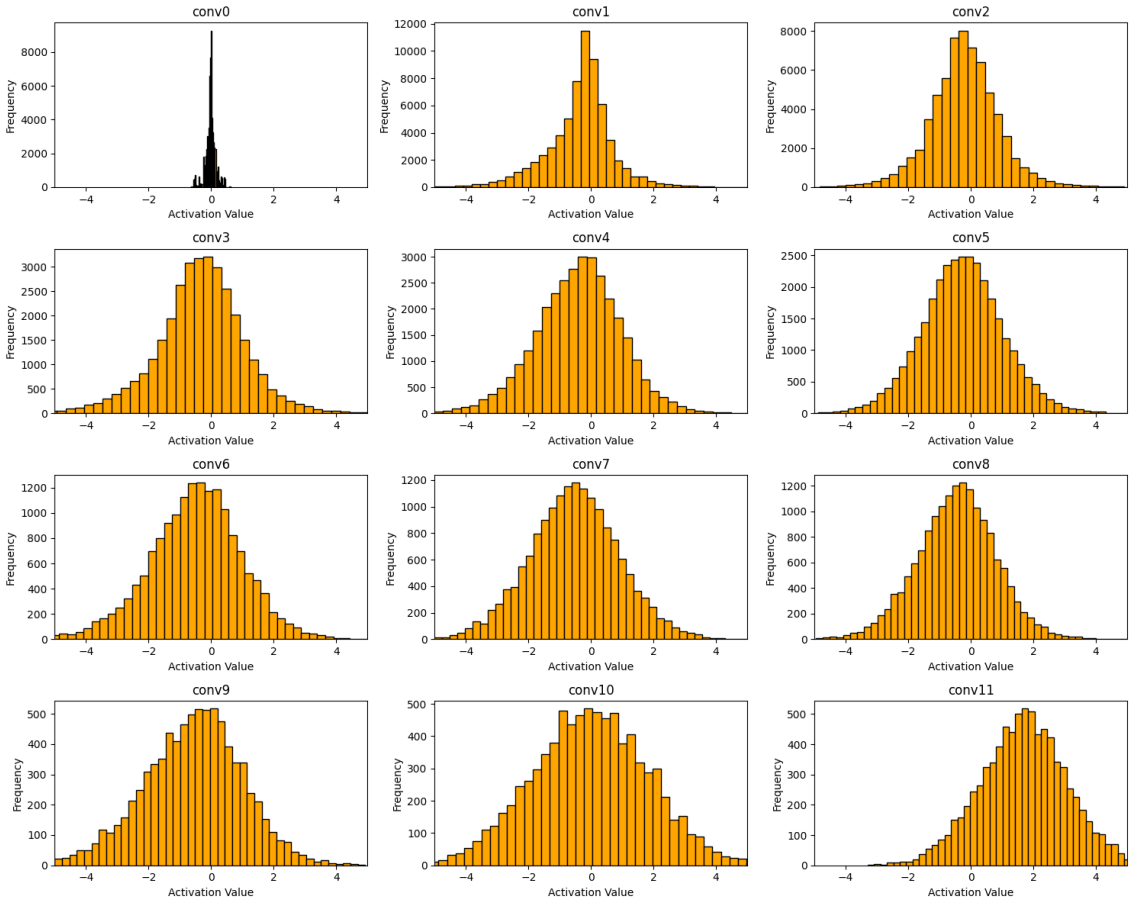


Figure 24: Un-pruned Model Activation Balance

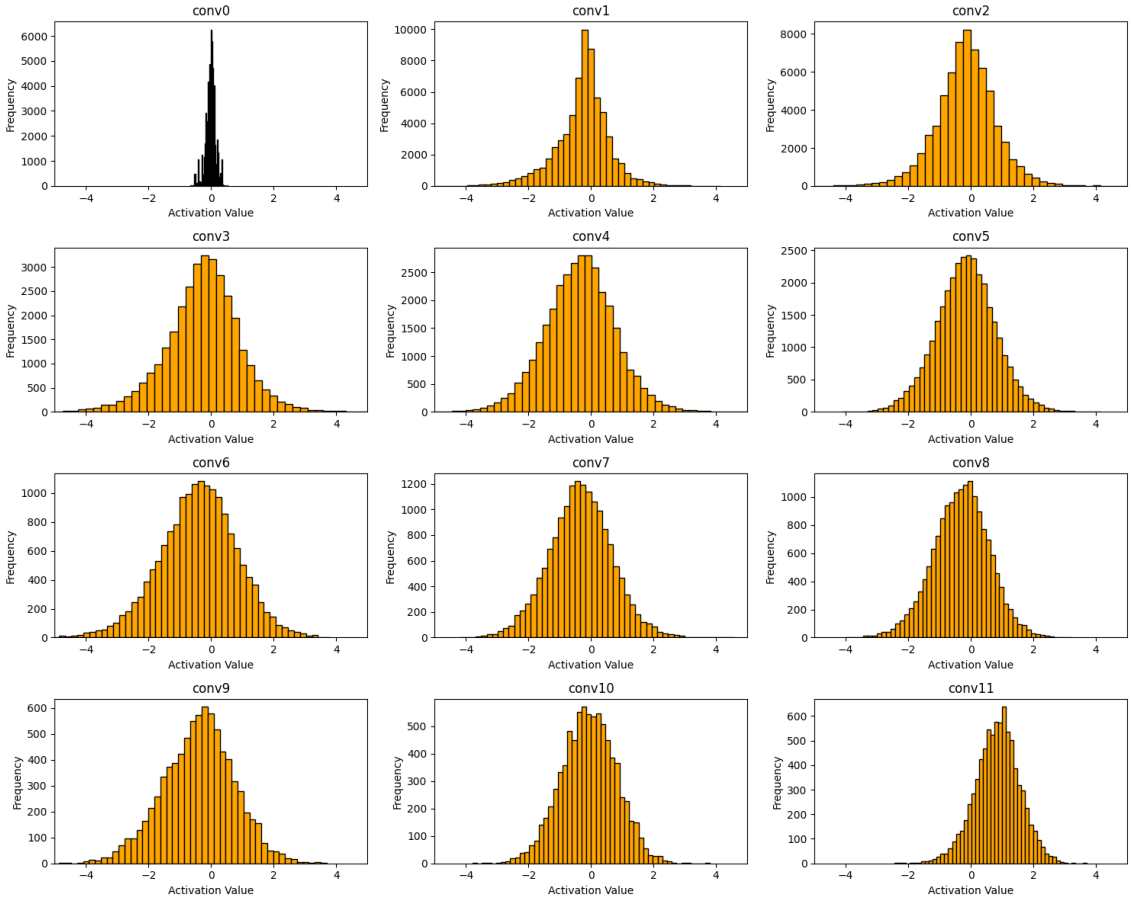


Figure 25: Fine-grained Model Activation Balance (Pre-Fine-tuning)

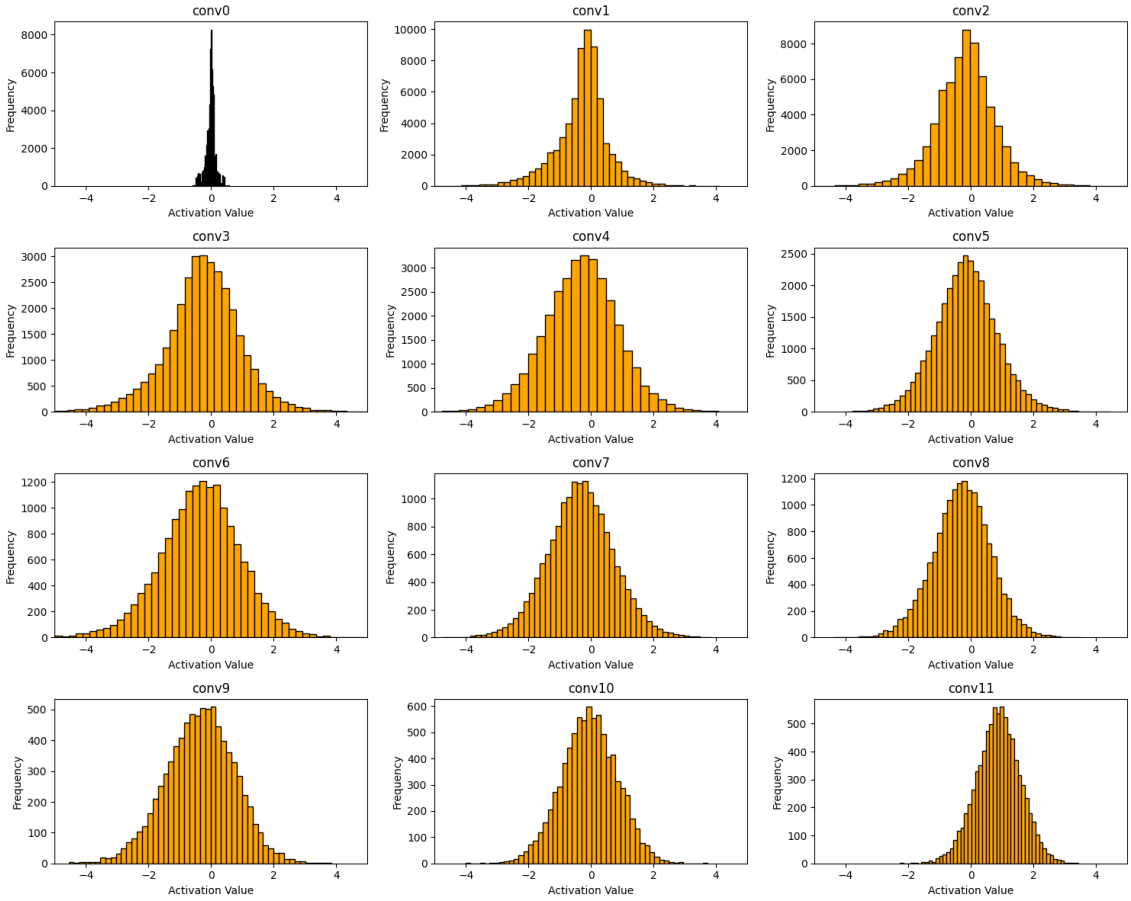


Figure 26: Fine-grained Pruned Model Activation Balance (Post-Fine-tuned)

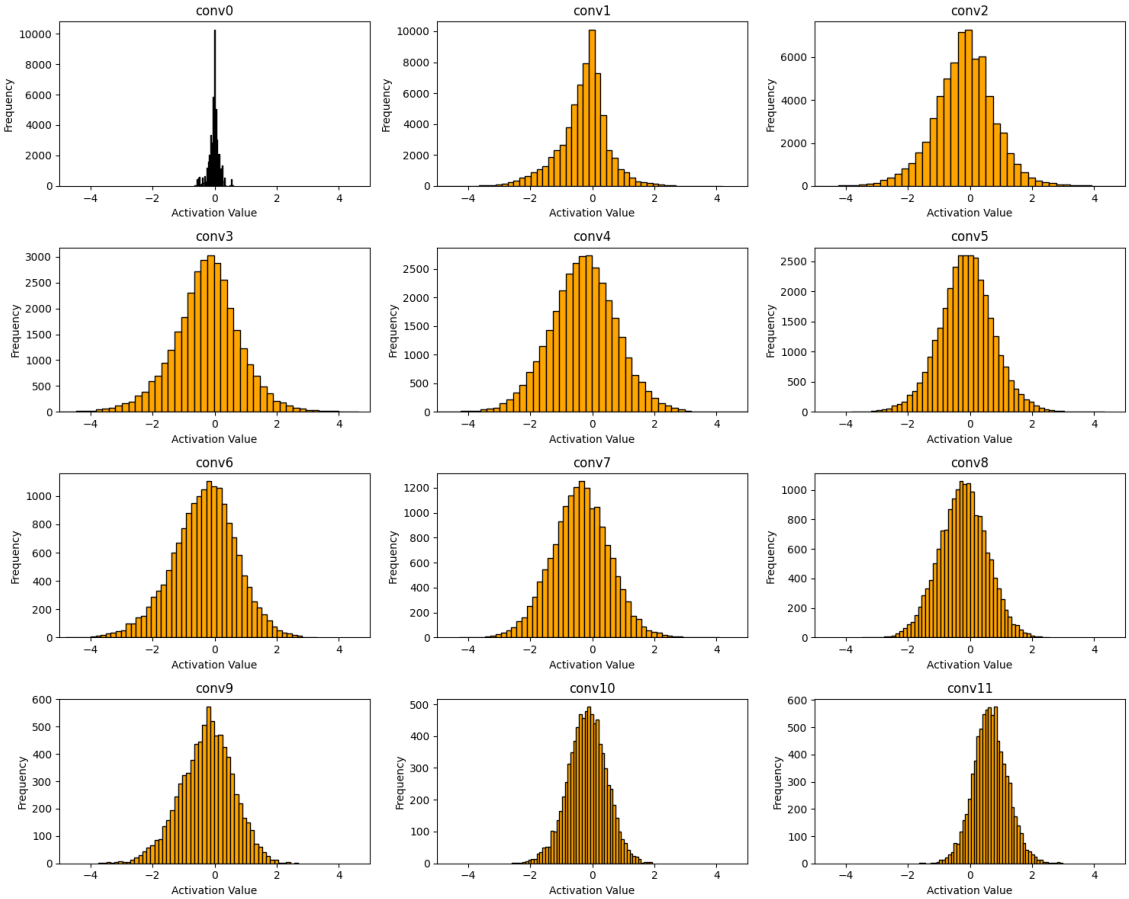


Figure 27: Kernel Pruned Model Activation Balance (Pre-Fine-tuning)

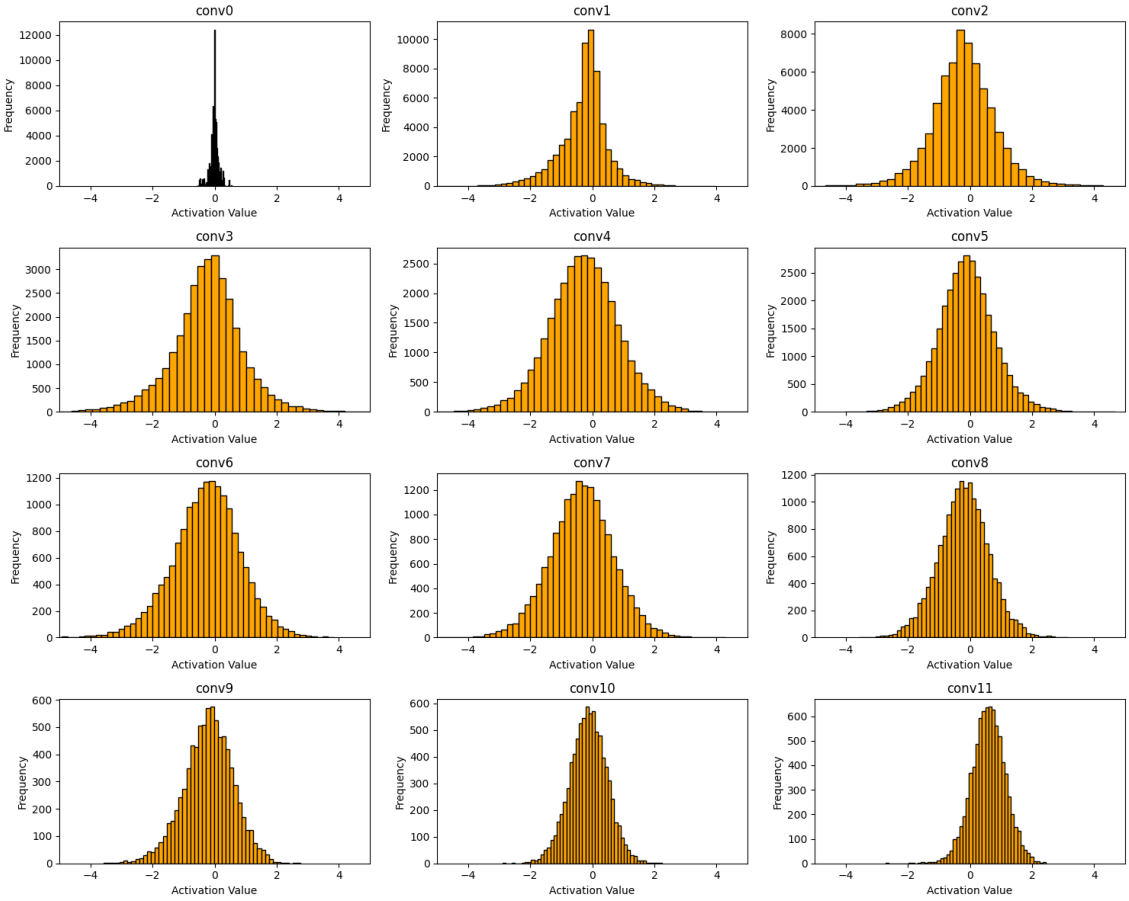


Figure 28: Kernel Pruned Model Activation Balance (Post-Fine-tuning)

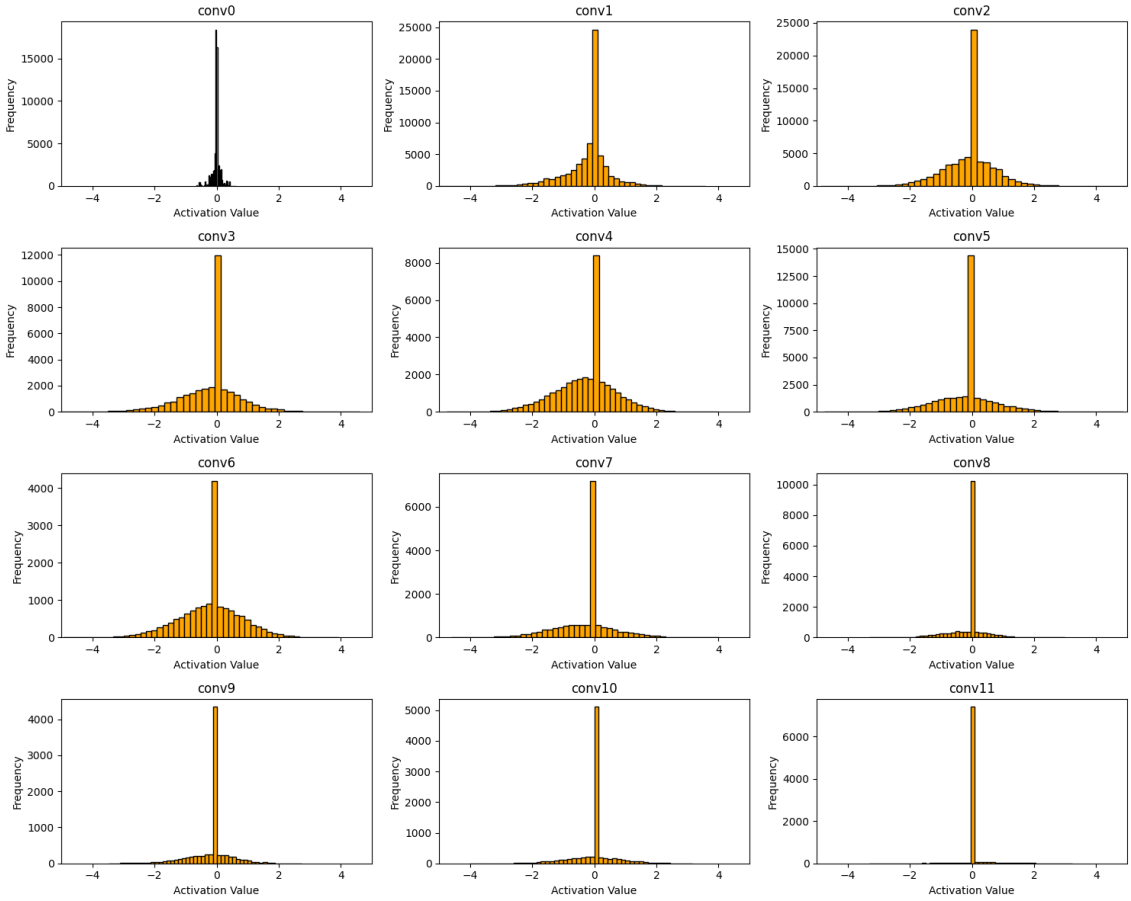


Figure 29: Filter Pruned Model Activation Balance (Pre-Fine-tuning)

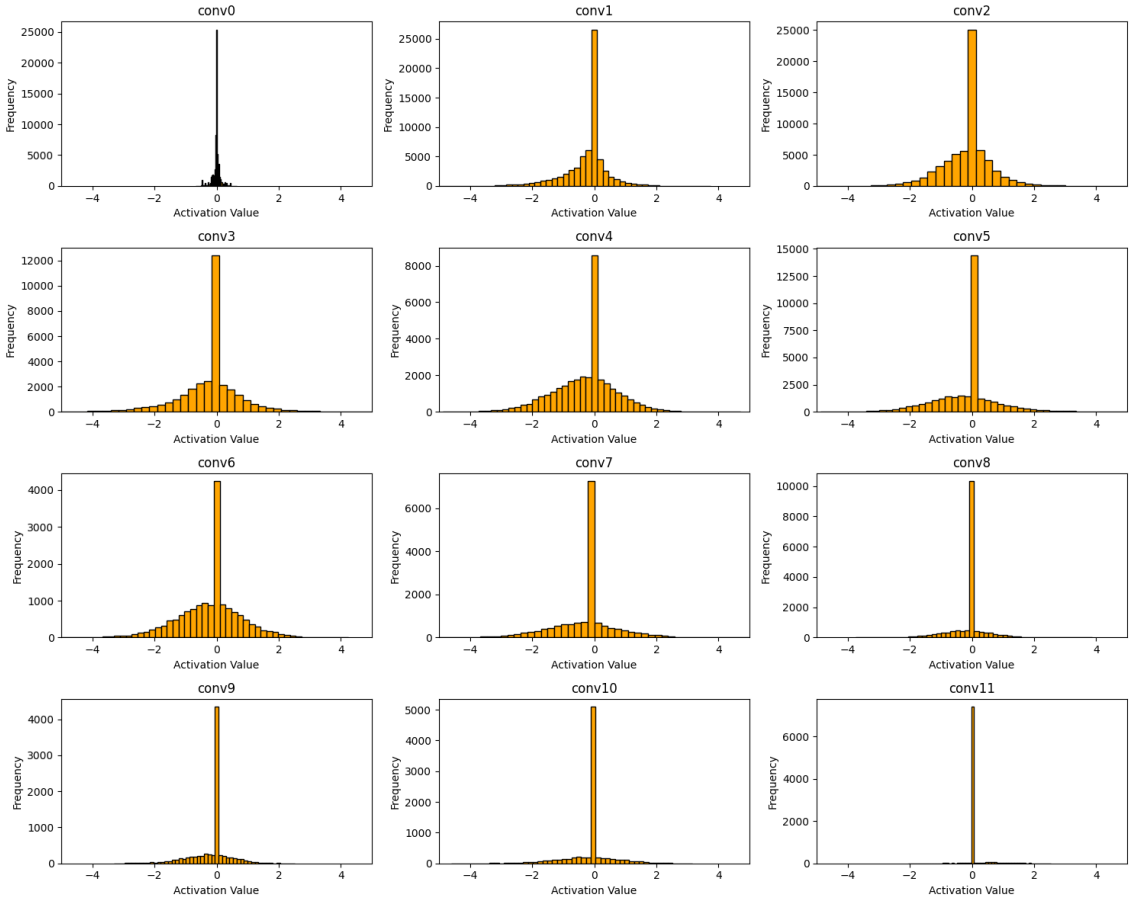


Figure 30: Filter Pruned Model Activation Balance (Post-Fine-tuning)