Edge AI for Industry 4.0 - Optimizing MobileNetV2 via Model Pruning

1. Data Preparation and Experimental Setup

This section details the dataset selection and preprocessing steps used to simulate an industrial defect detection scenario.

The data handling is managed by the `torchvision` pipeline, utilizing CIFAR-10 as a proxy for industrial object classification.

1.1. 1.1. Dataset Selection (CIFAR-10)

To simulate a resource-constrained industrial environment (e.g., classifying parts on a conveyor belt), we utilize the CIFAR-10 dataset.

- **Context:** Industry 4.0 requires rapid classification of objects (defects vs. normal) on edge devices..
- **Dataset Structure:**
  - **Classes:** 10 distinct classes (e.g., plane, car, bird, etc.) representing different industrial categories.
  - **Resolution:** 32×32 pixels, simulating low-bandwidth sensors.
  - **Size:** 50,000 Training images, 10,000 Test images.

1.2. Preprocessing and Normalization

The `transform_train` and `transform_test` pipelines prepare the raw images for the MobileNetV2 architecture.

- **Normalization:**
  - **Mean (μ):** (0.4914, 0.4822, 0.4465)
  - **Standard Deviation (σ):** (0.2023, 0.1994, 0.2010)

1.3. Data Loaders

To mimic edge inference conditions, the test loader is configured to simulate batched processing. * **batch Size:** 64 * **Workers:** 2

2. Model Architecture and Pruning Configuration

The optimization task utilizes a lightweight convolutional neural network, modified for pruning experiments.

2.1. Model Selection

- **Model: MobileNetV2**
- **Pre-training:** Initialized with `MobileNet_V2_Weights.DEFAULT` (ImageNet weights)
- **Final Layer Modification:** The final classifier layer (`model.classifier[1]`) is replaced to output 10 classes instead of 1000.

3. Pruning Strategies

The core of this project is the run_unified_experiment function, which implements a flexible pruning engine capable of executing distinct strategies defined in experiments_config.

Strategy	Heuristic	Logic
Unstructured L1	Magnitude	Removes individual weights with the smallest absokute value.
Structured L2	Norm	Removes entire channels/filters based on their L2 norm. Reduces matrix dimensions for real speedup but causes higher accuracy loss.
Random	Stochastic	Randomly removes weights. Used as a baseline "sanity check" to validate the effectiveness of L1 pruning.

3.1. One-Shot Pruning

The evaluation was performed using the best model checkpoint saved based on validation mAP.

- **Process:** The target sparsity (e.g., 50% or 90%) is applied in a single step.
- **Workflow:**
  - **Apply Mask :** Remove X% of weights.
  - **Fine-tune :** Train for 1 epoch to recover accuracy.
- **Use Case:** Tests the robustness of the model against massive, sudden information loss.

3.2. Iterative Pruning.

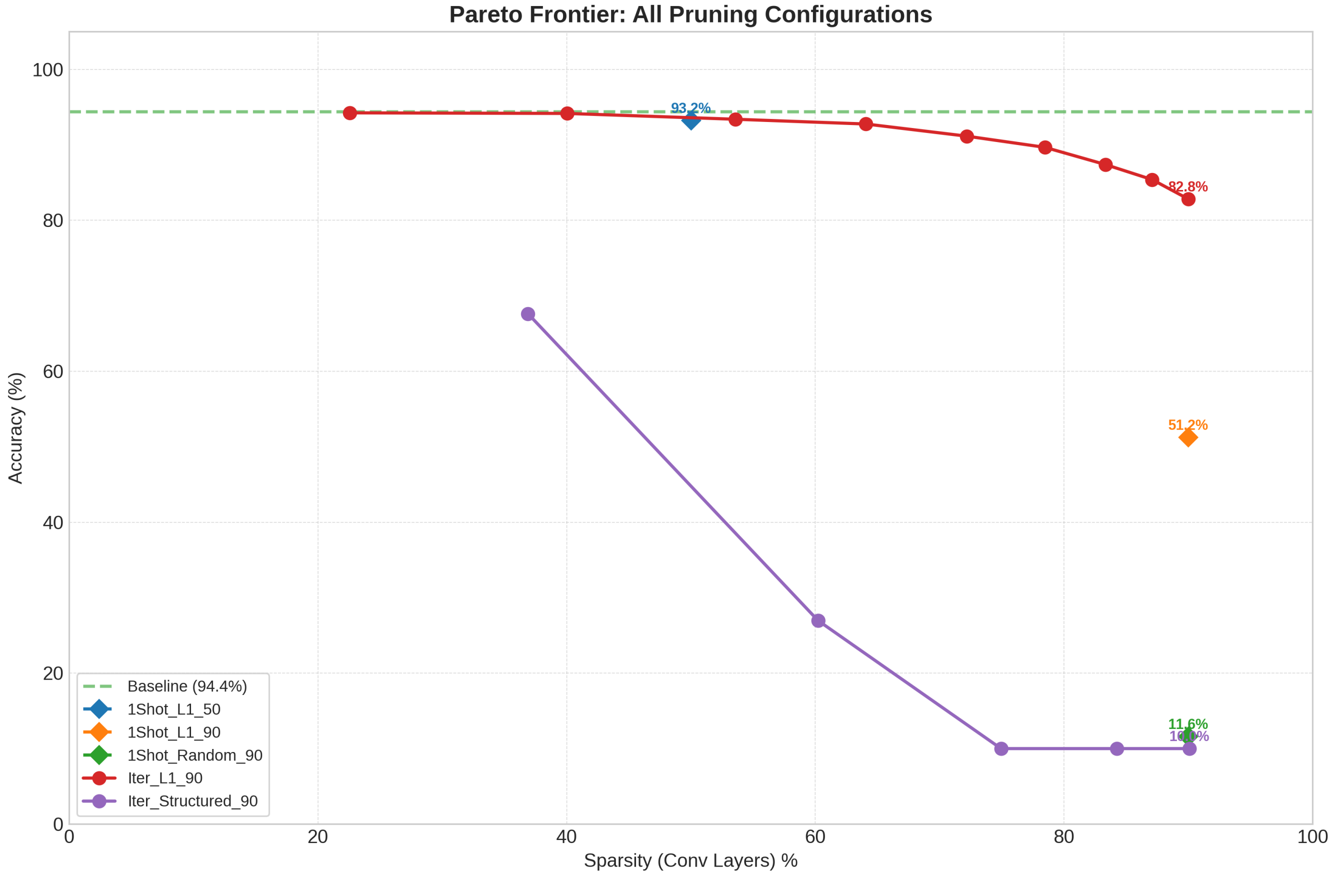
- **Process:** The target sparsity is reached gradually over N steps (e.g., 10 steps).
- **Schedule Formula:** $$S_{step} = 1 - (1 - S_{target})^{\frac{1}{N}}$$
- **Workflow:**
  - Prune a small fraction (e.g., 15%).
  - **Fine-tune :** Train for 1 epoch to recover accuracy.
  - Repeat until target sparsity is reached.
- **Use Case:** Allows the model to adapt its remaining weights to compensate for the loss, essential for high compression rates (80%+).

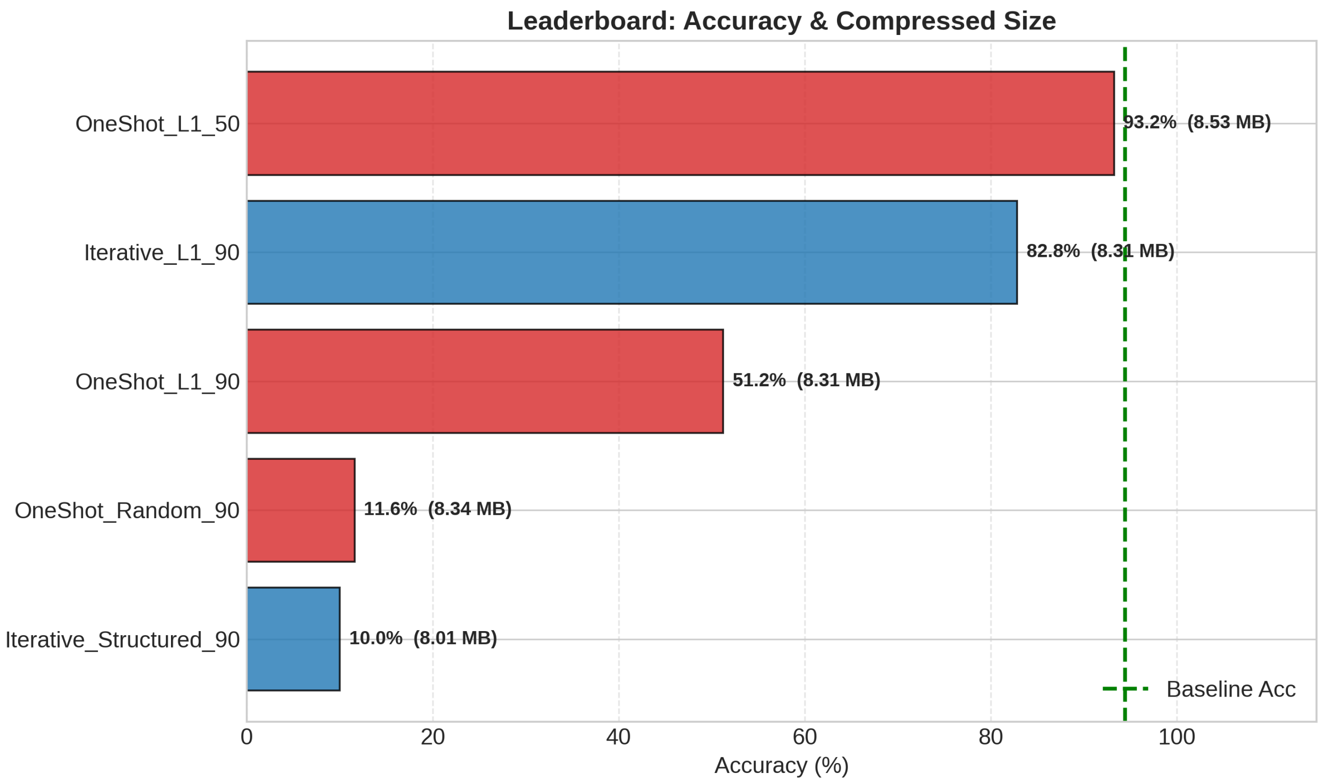
4. Evaluation and Results

The models were evaluated based on three: Sparsity, Top-1 Accuracy, and Inference Time.

Experiment	Sparsity	Accuracy	Model Size (Gzip)	Inference Time
Baseline	0%	**94.4%**	**8.53 MB**	**6.90 ms**
OneShot L1	50%	**93.2%**	8.53 MB	6.78 ms
Iterative L1	**90%**	**82.8%**	**8.31 MB**	**7.03 ms**
OneShot L1	90%	51.2%	8.31 MB	6.85 ms
OneShot Random	90%	11.6%	8.34 MB	6.84 ms

Experiment	Sparsity	Accuracy	Model Size (Gzip)	Inference Time
Iterative Structured L2	90%	10.0%	8.01 MB	6.92 ms





4.1. Analysis (Why do the results look like this?)

This section details the theoretical reasons behind our experimental findings.

4.1.1. Why did Iterative Pruning beat One-Shot by 30%?

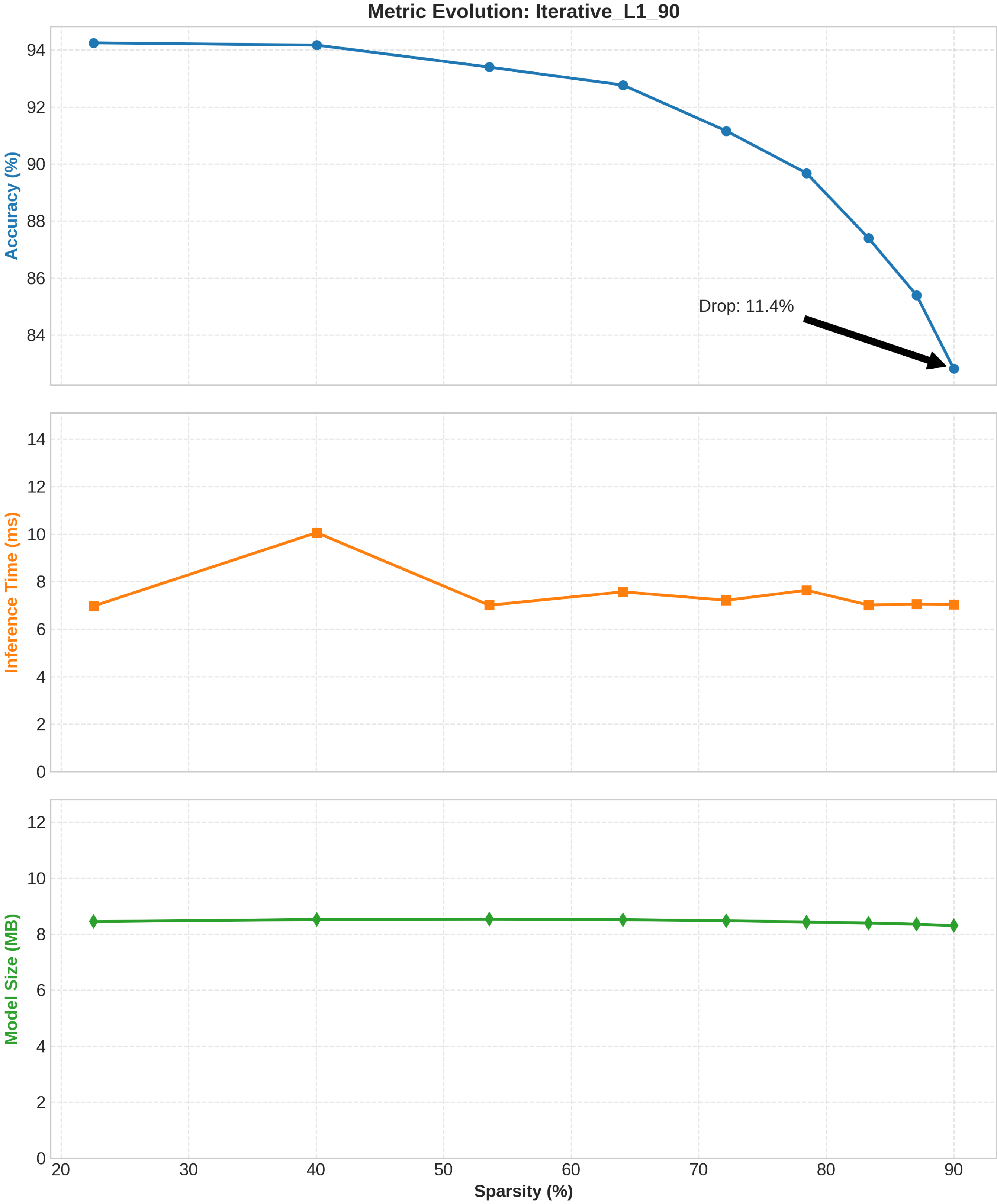
The **"Lottery Ticket Hypothesis"** suggests that dense networks contain sparse subnetworks that can be trained to high accuracy.

- **One-Shot (90%):** We abruptly cut the connections. The remaining 10% of weights were forced to take over instantly, which was too difficult a jump in the loss landscape.
- **Iterative:** By pruning 10-20% at a time and retraining, we allowed the surviving weights to adjust their values gradually. This "healing" process guided the optimization trajectory into a basin where a 90% sparse solution exists.

4.1.2 Why did Structured Pruning fail completely?

Structured pruning removes entire filters.

- **The MobileNet Factor:** MobileNetV2 is designed to be efficient. It uses **Depthwise Separable Convolutions**, which already significantly reduce the parameter count (redundancy).
- **The Collapse:** Unlike VGG or ResNet, which have massive redundancy, MobileNetV2 has very little "fat" to trim. When we removed 90% of the *channels* structurally, we destroyed the network's topology, breaking the flow of information completely.



4.1.3. Why didn't Inference Time decrease?

Despite removing 90% of the weights, our inference time remained ~7ms.

- **The Reason:** Despite removing 90% of the weights, our inference time remained constant (~7ms). This is a known characteristic of the **PyTorch prune** module, which implements Masking rather than Physical Removal.

4.1.4 The File Size Paradox

Observation: Despite removing 90% of parameters, the disk size only dropped by ~0.2 MB.

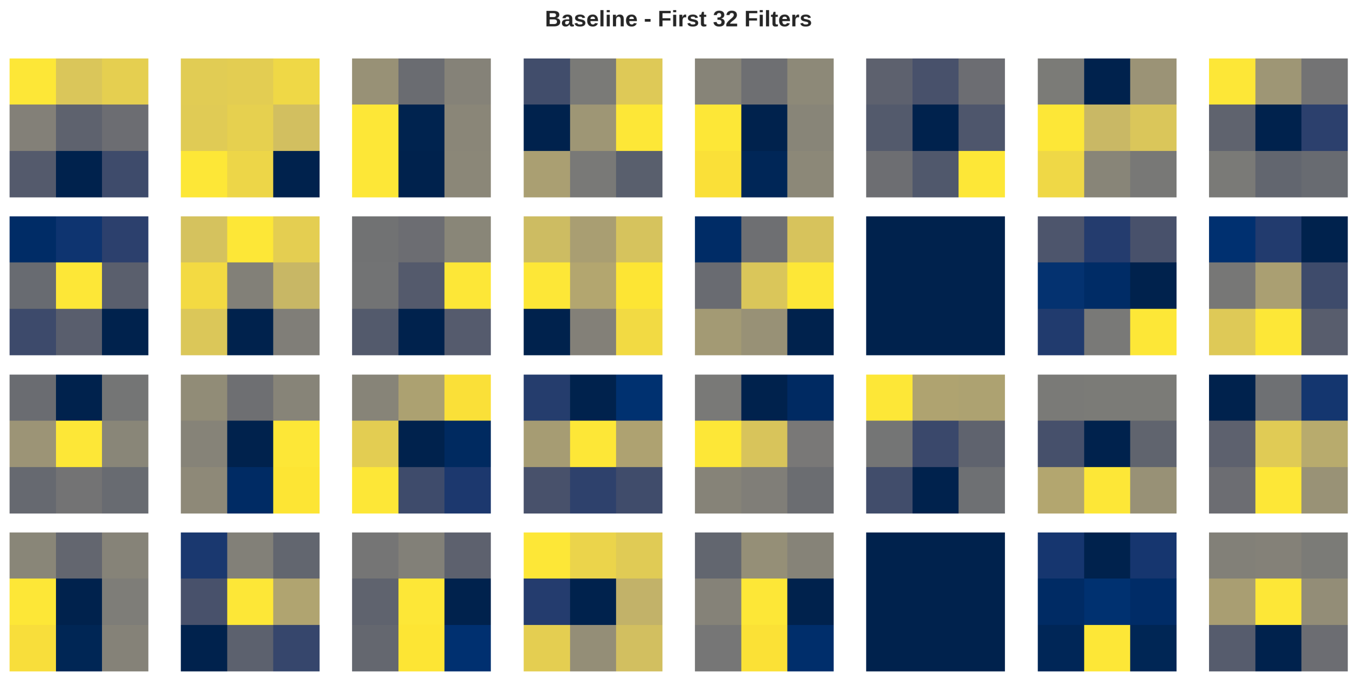
Explanation: We utilized PyTorch Masking, which zeroes out weights but preserves the original tensor shapes (Dense Format).

1. **Dense Tensor**: Stores every value, including zeros.
2. **Sparse Tensor**: Stores only indices (x, y) and values of non-zero weights.
3. **Insight**: To realize storage benefits in a production environment, we would need to export this model using a Sparse Format (CSR/CSC) or specialized hardware encodings.

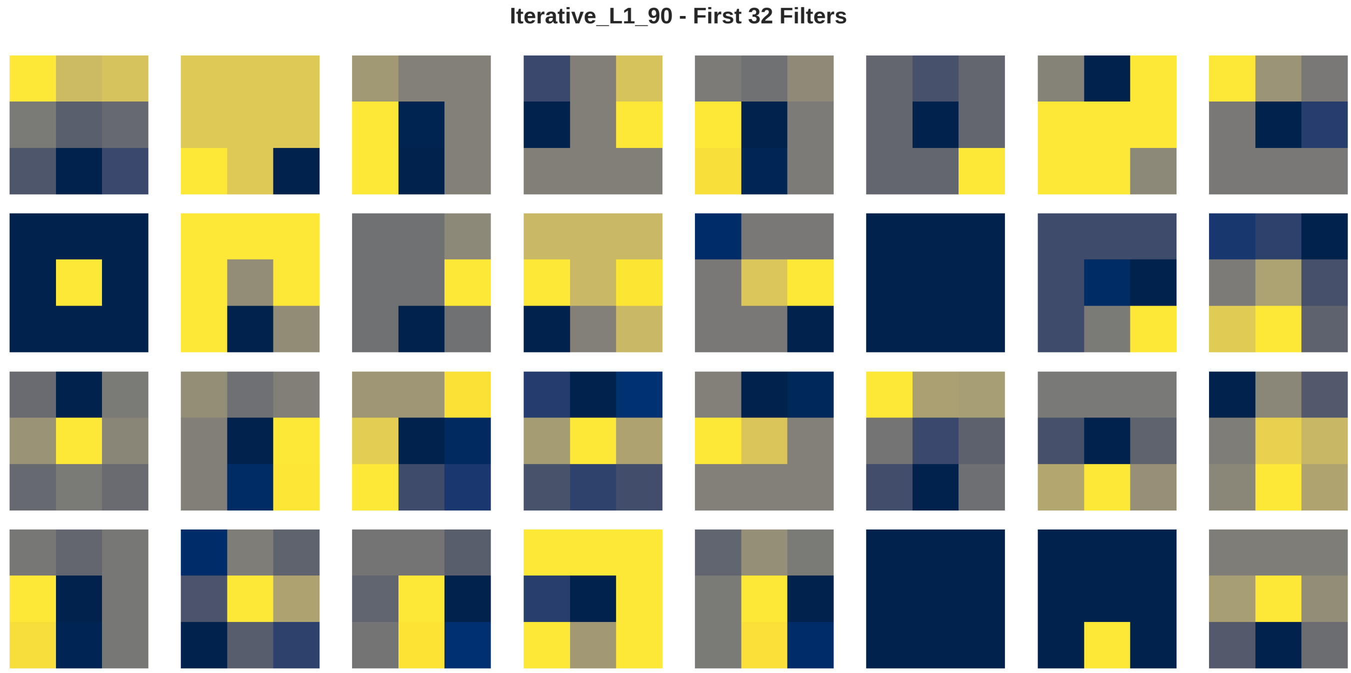
4.1.5 Kernel Visualization Analysis

To understand what the model actually removed, we visualized the weights of the first convolutional layer (32 filters). This visual evidence explains the drastic difference in accuracy between strategies.

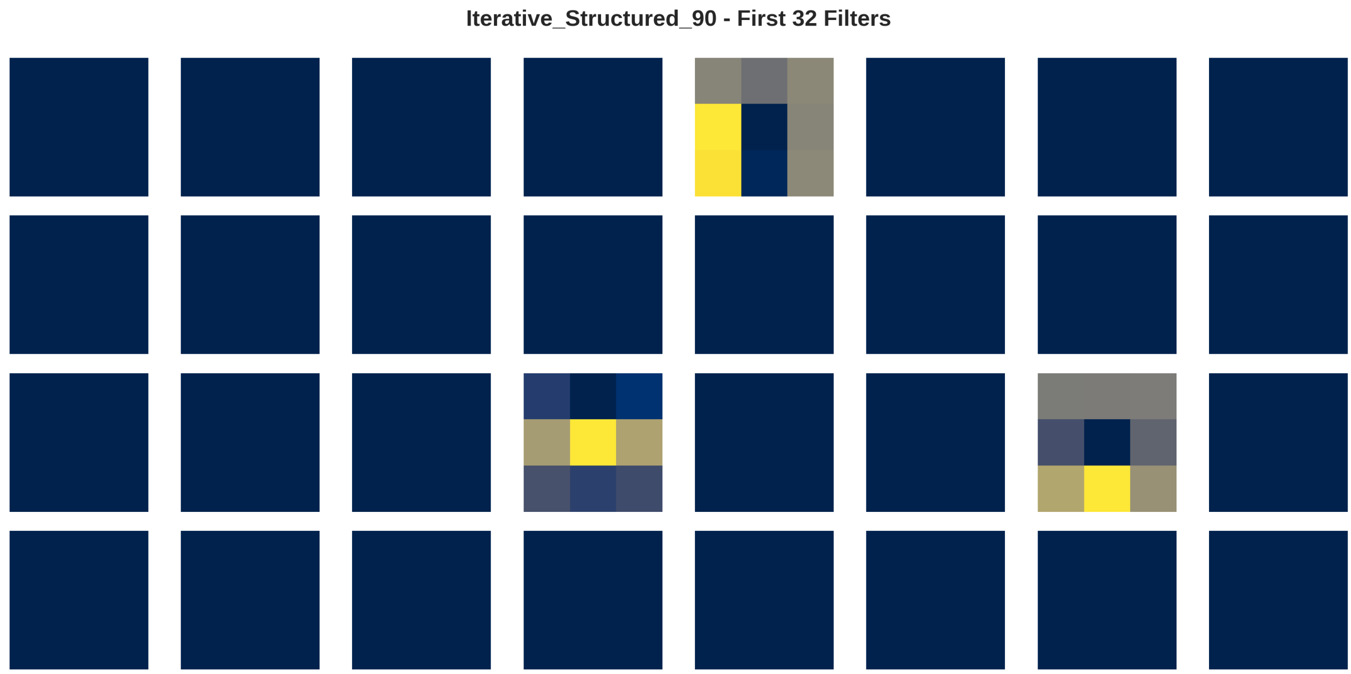
Baseline: The original filters contain rich, dense patterns used for edge and color detection.



Iterative L1 (90% - 82.8% Acc): The "Swiss Cheese" effect. The L1 algorithm surgically removed individual unimportant pixels (weights) while preserving the high-magnitude structures. This allows the filter to still function, albeit with less fidelity.

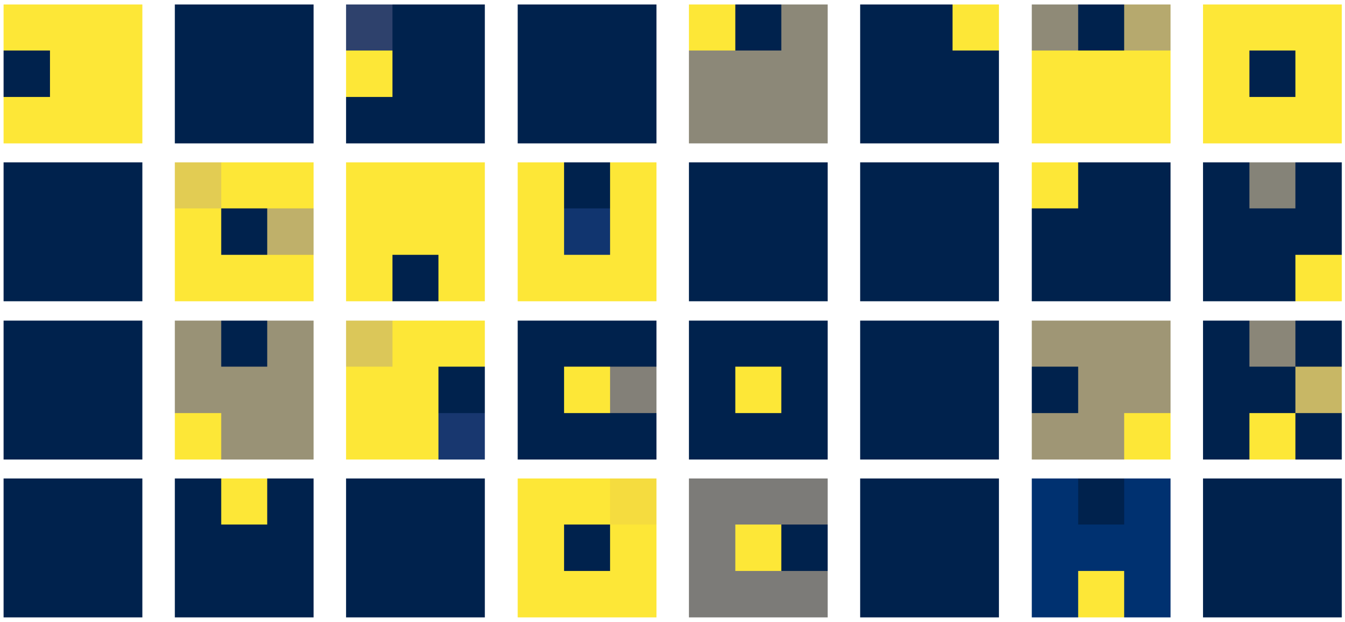


Structured L2 Pruning (90% - 10.0% Acc): The "Blackout". This method killed entire filters (the black squares). Because MobileNetV2 is already compact, killing 90% of the filters destroyed the model's ability to extract features entirely, resulting in random guessing.



Random Pruning (90% - 11.6% Acc): The "Static". Unlike L1 pruning which preserved structure, random pruning destroyed the coherent patterns necessary for convolution, resulting in noise.

OneShot_Random_90 - First 32 Filters



5. Conclusion

This project successfully demonstrated that Iterative Unstructured Pruning is the optimal strategy for compressing MobileNetV2 on CIFAR-10, achieving 90% sparsity with only an 11% drop in accuracy.

Key Takeaways: **Gradual is Better**: The "healing" phase in iterative pruning is critical. One-shot pruning at high sparsity levels causes irreversible brain damage to the model.

Architecture Matters: MobileNetV2 is highly sensitive to Structured Pruning. Unlike VGG or ResNet, it lacks the channel redundancy required to survive the removal of entire filters.

The Hardware Gap: While we achieved theoretical compression (sparsity), realizing actual gains in speed (latency) and storage requires specialized deployment steps (Sparse Formats and Hardware) beyond standard PyTorch masking.