Automated Neural Network Pruning via a Genetic Algorithm

Technical Report

1 Abstract

This document outlines a computational framework for the automated compression of Convolutional Neural Networks (CNNs) using a Genetic Algorithm (GA). The system performs structured channel pruning to generate smaller, more computationally efficient models with minimal degradation in accuracy. The core of the methodology is a sophisticated fitness evaluation function that incorporates model fine-tuning to provide a realistic assessment of a pruned architecture's potential. The entire process, from baseline training to the evolution of a final compressed model, is implemented in Python using the PyTorch library.

2 Core Methodology

2.1 Baseline Model Architecture

The baseline network is a standard CNN architecture designed for the MNIST dataset. It consists of four layers with learnable parameters:

- **Convolutional Layer 1 (conv1):** 32 output channels, 5×5 kernel
- **Convolutional Layer 2 (conv2):** 64 output channels, 5×5 kernel
- **Fully-Connected Layer 1 (fc1):** 1024 neurons
- **Fully-Connected Layer 2 (fc2):** 10 neurons (for 10 classes)

ReLU activation functions are used after each convolutional and the first fully-connected layer. MaxPooling is applied after each convolutional layer.

2.2 Pruning Strategy: Structured Channel Pruning

The framework employs structured pruning, where entire channels (filters) of the convolutional layers are removed. This approach was chosen over unstructured (individual weight) pruning because it directly reduces the dimensions of the feature maps and weight tensors. This leads to:

- Tangible decreases in memory usage and model size
- Reduced number of Multiply-Accumulate (MAC) operations
- Speedups in inference without requiring specialized hardware or sparse libraries

2.3 Optimization Algorithm: Genetic Algorithm (GA)

A Genetic Algorithm is used to explore the space of pruned architectures.

Chromosome Representation

Each solution (chromosome) is represented as a binary vector. Each bit corresponds to a prunable channel in the convolutional layers:

- **1:** Keep the channel
- **0:** Prune the channel

Fitness Evaluation

Each chromosome's fitness is evaluated by:

1. Instantiating the pruned model
2. Fine-tuning for a few epochs on a subset of the training data
3. Calculating a weighted score:

$$\text{Fitness} = w_{\text{acc}} \cdot \left(\frac{100 \cdot \text{accuracy}}{100} \right) + w_{\text{macs}} \cdot \left(1 - \frac{\text{MACs}_{\text{pruned}}}{\text{MACs}_{\text{baseline}}} \right) \tag{1}$$

Where:

- w_{acc}: Weight for accuracy
- w_{macs}: Weight for MAC reduction

Evolutionary Operators

- **Selection:** Tournament selection to choose parents based on fitness.
- **Crossover:** Uniform crossover mixes parent genes to produce children.
- **Mutation:** Bit-flip mutation for diversity and exploration.
- **Elitism:** Best individual preserved in each generation.

3 Implementation Details & Workflow

3.1 Software & Libraries

- **Language:** Python 3.8
- **Core Library:** PyTorch
- **Numerical Computation:** NumPy
- **Profiling:** `thop` for MAC counting
- **Visualization:** Matplotlib

3.2 Dataset

The MNIST dataset is split as follows:

- **Training Set:** 50,000 images
- **Validation Set:** 10,000 images
- **Test Set:** 10,000 images

A subset of the training data is used during GA fine-tuning to speed up the evolution process.

3.3 Experimental Workflow

1. Train a baseline CNN on the full training set.
2. Evaluate the baseline on the test set.
3. Initialize the GA with a random population of chromosomes.
4. For a fixed number of generations:
    - Evaluate fitness of each chromosome
    - Apply selection, crossover, and mutation
    - Carry forward elite individuals
5. Cache fitness evaluations using `@lru_cache` to avoid redundant computations.
6. Select the best chromosome and reconstruct the pruned model.
7. Retrain the pruned model on the full dataset.
8. Evaluate the final pruned model on the test set.

4 Conclusion

This framework automates CNN compression using a Genetic Algorithm with structured pruning. The evolutionary strategy ensures an efficient search of the pruning space, balancing accuracy and computational cost. Final models are not only smaller and faster but also maintain performance comparable to the original, making this method highly suitable for deployment on resource-constrained devices.