Enhancing CNN Efficiency for Time Series Forecasting: Pruning by Weight Magnitude

Raahul Esakiraja
DLQF Quantitative Researcher

Abstract

Time series forecasting plays a critical role in various domains, from finance to supply chain management. In recent years Convolutional Neural Networks (CNNs) have become more prevalent in time series forecasting due to their ability to understand "spatial" structure of continuous coordinates. A time coordinate works just as well as a spatial coordinate thus allowing CNNs to be leveraged in this field. However, deep learning models like CNNs often face deployment challenges due to their high memory and computational demands. This paper explores a magnitude-based weight pruning technique applied to a pre-trained CNN, reducing model complexity by sparsifying low-magnitude weights and reapplying to model structure to create a compact model. We evaluate the method using metrics like memory usage, execution time, and predictive accuracy. Our results reveal that significant model compression can be achieved without substantial loss in accuracy, offering a practical solution for real-world forecasting applications.

Contents

1 Introduction

1.1 Literature Review

Convolutional Neural Networks (CNNs) have become a ubiquitous tool for time series analysis due to their ability to automatically extract features, promote sparsity and weight sharing, and their end-to-end trainability. As CNNs grow in size and complexity to handle increasingly intricate tasks, concerns about their computational and memory demands arise, especially for time series data, where datasets can be both large and complex. Model compression through pruning has emerged as a powerful approach to address these concerns by eliminating redundant or less important connections within the network, resulting in smaller models with fewer parameters. Various techniques have been developed for efficient pruning in CNNs. Earlier work, such as Optimal Brain Damage and Optimal Brain Surgeon (Hassibi & Stork, 1993[1]), utilized second-order Taylor expansions to identify and remove parameters based on their impact on the cost function. However, these methods often faced scalability issues due to the computational complexity of calculating the Hessian matrix.

More recent work has introduced new criteria based on approximations of the cost function's change due to pruning. Computationally efficient magnitude-based pruning techniques have gained traction, which rank parameters (e.g., weights, kernels, or feature maps) based on magnitude, pruning those with the smallest values. For instance, Convolutional Neural Network Compression via Dynamic Parameter Rank Pruning (Sharma & Heard et al., 2024[2]) proposes a gradual pruning technique where the sparsity level increases during training, allowing the network to adapt to changes and maintain performance.

While much of the research on CNN pruning has focused on image-based tasks, its application to time series data is gaining attention. For example, To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression (Zhu & Gupta, 2017[3]) demonstrated the benefits of pruning Recurrent Neural Networks (RNNs) for speech recognition, which involves temporal data. Their findings highlighted that pruned RNNs could outperform dense RNNs of similar size, suggesting potential benefits for time series tasks. They also introduced a gradual pruning scheme to balance sparsity and performance.

1.2 The Approach Taken in This Study

Building on these ideas, this study explores the use of pruning CNNs for time series forecasting, particularly in financial data. We propose a pruning method that increases model sparsity by zeroing out low-magnitude weights in a pretrained CNN. Our approach evaluates trade-offs among memory efficiency, execution time, and predictive accuracy, aiming to establish a balanced framework for compressing CNN models while retaining high performance in financial time series forecasting.

References

[1] Hassibi, Babak, and David G. Stork. *Optimal Brain Damage.* Advances in Neural Information Processing Systems, 1993.

[2] Sharma, Manish, and Jamison Heard. *Convolutional Neural Network Compression via Dynamic Parameter Rank Pruning.* arXiv, 15 Jan. 2024. Available at: https://arxiv.org/pdf/2401.08014.pdf.

[3] Zhu, Michael H., and Suyog Gupta. *Exploring the Efficacy of Pruning for Model Compression.* arXiv, 13 Nov. 2017. Available at: https://arxiv.org/pdf/1710.01878.

2 Data Description

This study approaches enhancing CNN efficiency for time series forecasting by pruning weights by magnitude required the construction of a financial forecasting CNN. This required training and testing data on the SPY.

2.1 Training Data

We used historical price data from the S&P 500 ETF (SPY) spanning five years (1,250 trading days), obtained via the `yfinance` API. Adjusted closing prices, accounting for stock splits and dividend distributions, were normalized using a `MinMaxScaler` to scale values between 0 and 1. This normalization ensured stable neural network training.

The normalized data was structured into sequences with a 14-day lookback window, where each input contained 14 days of prices, and the target was the price on the following day. This approach captures both short-term price movements and longer-term patterns. The dataset was then split into training (80%) and validation (20%) sets, facilitating effective model development and tuning.

2.2 Testing Data

For evaluation, we use a separate dataset, processed identically to the training data to ensure consistency. This includes normalization and sequence generation with the 14-day window. Predictions are inverse-transformed to the original scale for interpretability, and performance is assessed using metrics like Mean Squared Error (MSE) and Mean Absolute Error (MAE).

2.3 Comparison Data

To compare models, we test the original, pruned, and compact versions using the same data sequences. This ensures a fair evaluation of accuracy, efficiency, and resource utilization. Metrics like MSE, MAE, memory usage, and inference speed are analyzed uniformly across models.

Weight distribution analysis explores the effects of pruning, identifying preserved pathways critical to prediction. The comparison also examines robustness during volatile market conditions and evaluates practical trade-offs between model size and performance.

By maintaining consistent preprocessing and evaluation standards, this methodology ensures reliable and meaningful insights into the trade-offs between model complexity, efficiency, and predictive accuracy.

3 Methodology

3.1 Building CNN Model Architecture

For this study, to explore the ability of comparing two models that are nearly identical, it required the ground up building of a 1-dimensional convolutional neuronal network for financial forecasting. The best suited framework for this scenario was Keras which is a is a high-level API for building and training deep learning models. It offers easy integration with TensorFlow, a pre-existing frame-

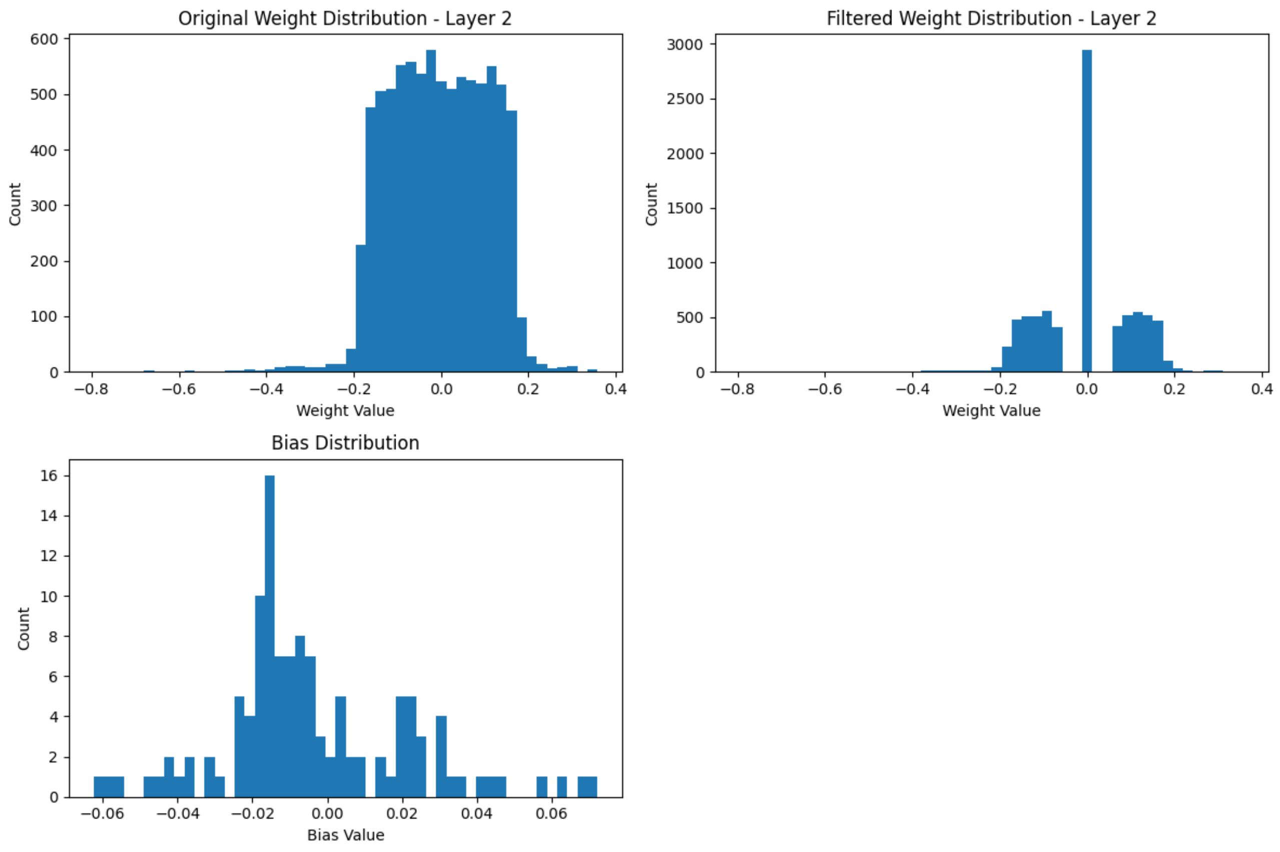

Figure 1: DenseLayer1 Weight Distribution before and after pruning methodology

work.

The model is a one-dimensional CNN beginning with a Conv1D layer, which applies 5 filters of kernel size of 1 with a stride of 1 and ReLU activation to the input data. After the convolution, the output is flattened using the Flatten layer to transition to a fully connected network. The model then has two fully connected (Dense) layers: the first with 120 units and ReLU activation, followed by an output layer with a single neuron and ReLU activation. This was the ideal model for our sequential data with our expected value, a continuous financial forested value. Then the model was trained using the preprocessed data mentioned in the previous section.

3.2 Pruning Strategy

The model previously described undergoes a pruning process where weights are ranked by their magnitude, and only the upper 75th percentile is retained to create a more compact model.

The overall neural network layer can be expressed as:

$$y_j = \sigma\left(\sum_{i=1}^{N} w_{ij}x_i + b_j\right)$$

Where:

y_j is the output of the neuron j,
x_i is the input from neuron i,
w_{ij} represents individual weights.
b_j is the bias for neuron j,
N is the total numbe of input nuerons.
σ represents the activation function

This can be expressed formally as:

$$W_{kept} = \{w_{ij} \in W : |w_{ij}| > P_{35}(|W|)\}$$

Where:

W is the original weight matrix,

P_{35} represents the 35th percentile,
w_{ij} represents individual weights.

In the case of the SPYPredictor model, the weights in the `Conv1D` and `Flatten` layers are preserved. The dense layer, named `dense`, is pruned, reducing its non-zero parameters from 8,400 before pruning to 5,460 after pruning, consistent with W_{kept}. The subsequent dense layer is excluded from pruning because it processes only one non-zero parameter as it serves as the output layer.

These weights are then set to a dense layer in a identical model named `compactModel` outstanding of the pruned weights. During this whole process, the bias terms are left unaltered, as they do most of the value additions.

The weight distribution is shown in Figure 1 & 2:

3.3 Performance Metrics

model comparison evaluates performance between the two models, balancing predictive accuracy and computational efficiency. Key metrics include Mean Squared Error (MSE), for its sensitivity to large errors, and Mean Absolute Error (MAE), for its interpretability in dollar terms to compare directly to the realm of financial forecasting. To assess efficiency, the framework uses compression ratio and storage savings metrics to quantify pruning effectiveness, alongside layer-wise sparsity and weight distribution analysis for detailed insights. R-squared (R^2) evaluates trend capture, while layer-wise parameter analysis examines structural changes.

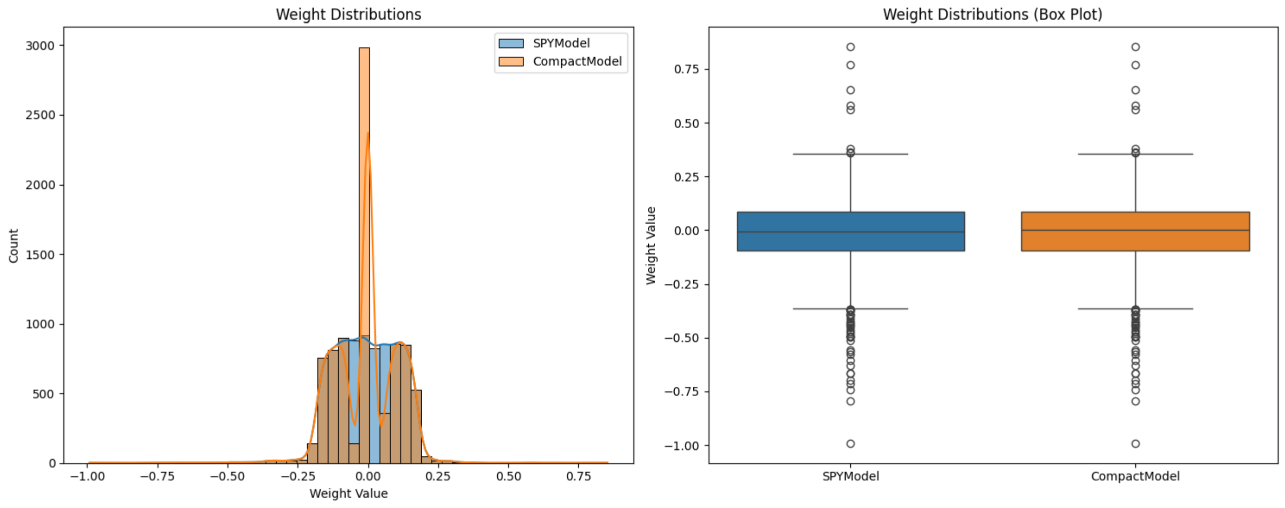

Figure 2: Weight Distribution

4 Results

4.1 Execution Time vs Data Size

The execution times for both the `compactModel` and `SPYPredictor` models are mostly consistent, ranging between 1-2 seconds, but there are noticeable spikes. Around the 12,500-day mark, the `compactModel` peaks at 3.73 seconds, and the `SPYPredictor` model reaches 2.47 seconds, indicating occasional bottlenecks unrelated to data size. Overall, both models show similar execution time patterns, suggesting comparable computational efficiency.

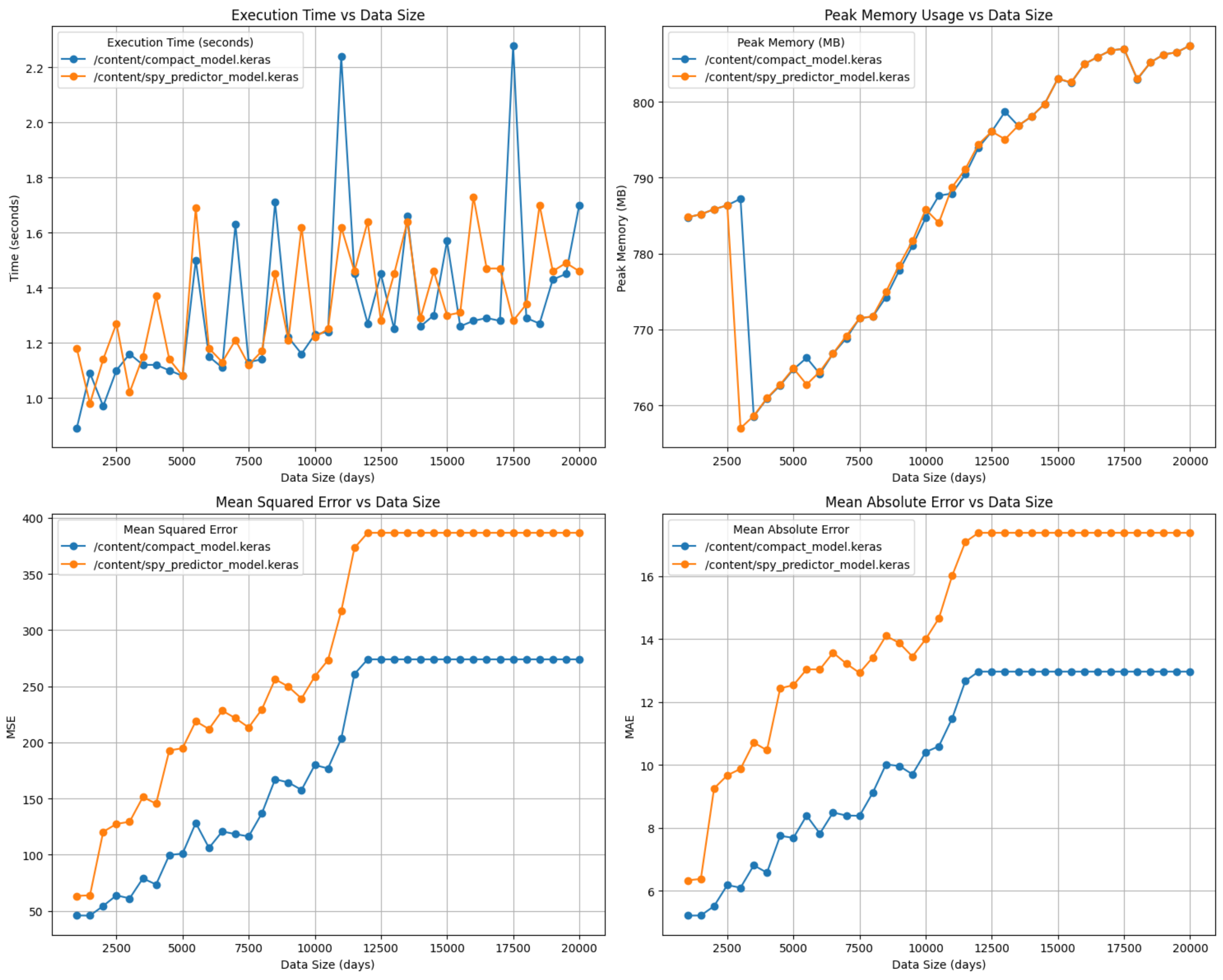

Figure 3: Evaluation Metrics

4.2 Peak Memory vs Data Size

Memory usage for both models increases steadily with data size, starting at 760MB and reaching about 830MB by 20,000 days. A dip occurs around 3,000-3,500 days, dropping to 755MB before climbing again. This trend indicates both models are memory-efficient for smaller datasets but require more memory as data size grows.

4.3 Accuracy

The error metrics reveal an interesting pattern of degrading performance as the dataset size increases. For both implementations (`compactModel` and `SPYPredictor`), there's a clear upward trend in both Mean Squared Error (MSE) and Mean Absolute Error (MAE), suggesting that accuracy diminishes with larger datasets. The `compactModel` consistently demonstrates better performance, particularly in smaller datasets where its MSE starts around 45-54 compared to `SPYPredictor`'s 60-120. This advantage is maintained throughout the scaling, though the gap stabilizes at larger data sizes.

The stabilization pattern that emerges around the 15,000 data point mark. After this point, both implementations show relatively consistent error rates, with MSE (`compactModel`) hovering around 273 and MSE (`SPYPredictor`) at 386, while MAE values converge to approximately 17.3-17.4 for both implementations. This stabilization suggests that beyond a certain dataset size, the error rates reach a plateau, indicating that further increases in data size may not significantly impact the error metrics. This could be valuable information for optimization decisions, as it suggests there might be diminishing returns in terms of accuracy improvement beyond certain data thresholds.

These values are shown in Figure 2 Above:

5 Conclusions and Discussion

We proposed a pruning method that increases model sparsity by zeroing out low-magnitude weights in a pretrained CNN. Our approach evaluated trade-offs among memory efficiency, execution time, and predictive accuracy, aiming to establish a balanced framework for compressing CNN models while retaining high performance in financial time series forecasting. The results of our findings shocased that the compact model performs similarily to the orgiinal model with occaisonal spikes around the 12,500 days of data points. This offers room for further testing as this breif spike values could affect the overall time series forcasting in a finacial setting becauuse finacial data can tend to be very high frequency if this model is used on intra-day momentum strategies. The memory of both models behave similarily as data grows which points to the idea that both models are memory efficient for smaller data set and taper of in their efficieny the larger the data set. This could also be further tested with high frequency data as more data values would be present in datasets. The accuracy of the compact model is what stood out in the study, in which the compactModel consistently demonstrates better performance, particularly in smaller datasets where its MSE starts around 45-54 compared to SPYPredictor's 60-120. This trend was also seen in the Mean absolute error test which simulated the inaccuracy similarly to difference in dollar amounts. The compact model also had a slightly better R^2 score compared to the original model as shown by Table 1 below:

This can also be shown in the visualization of predications on a sliding window basis every 14 days in figure 4 below:

In conclusion, this study highlights the potential of the weight based pruning as a computationally efficient alternative for small level financial time series forecasting. While both models exhibit similar execution time and memory usage trends, the compactModel demonstrates a notable edge in predictive accuracy, particularly for smaller datasets. The analysis also suggests that both models reach a plateau in error metrics for larger datasets, implying that further scaling of data size may have diminishing returns in accuracy improvement.

The observed memory efficiency and execution time stability make these models suitable for real-time applications, but the occasional execution time spikes warrant further investigation, especially for high-frequency trading strategies where consistency is crucial. The compactModel's superior R^2 score and lower error metrics establish it as a promising candidate for scenarios demanding a balance between computational resource efficiency and predictive accuracy.

Future work could focus on exploring the pruning performance under diverse financial scenarios, such as intra-day trading or multi-asset forecasting, where high-frequency data and complex correlations come into play. Additionally, further optimization of the model architecture could be tested with larger layers or more layer in general to truly test the efficacy of this pruning approuch.

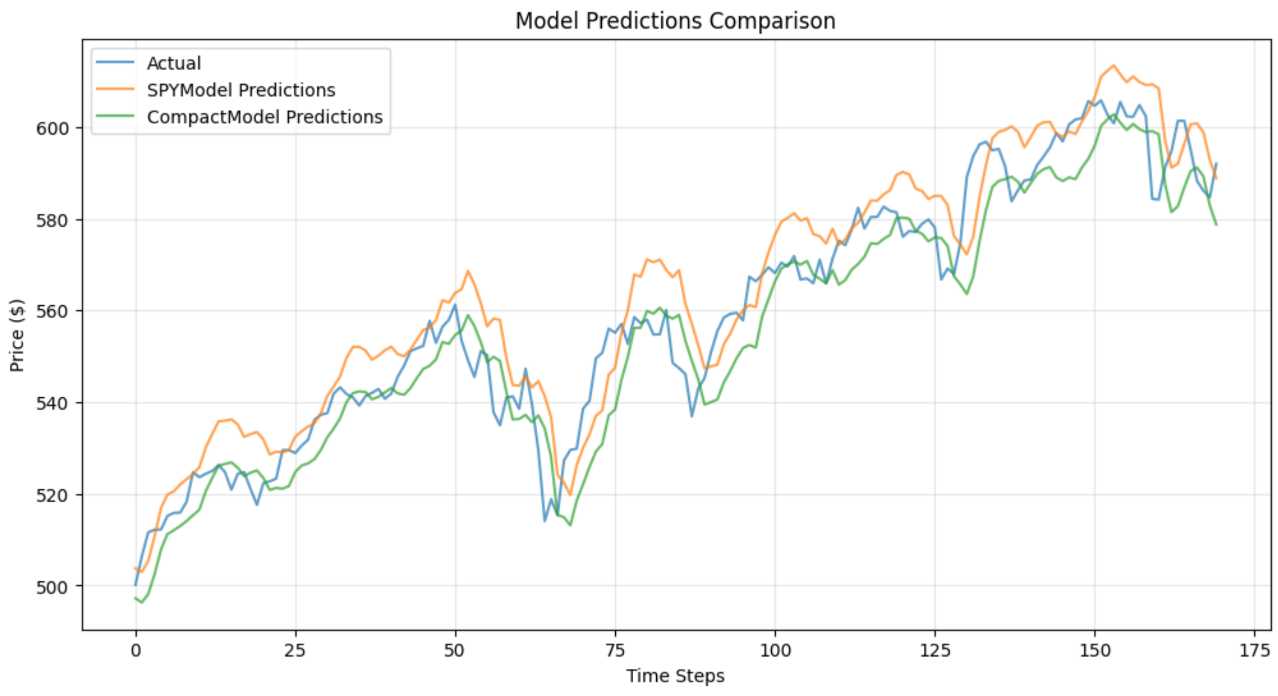

Figure 4: Model Comparison

Model	MSE	RMSE	MAE	R2
SPYModel	91.444878	9.562682	7.735817	0.876112
CompactModel	75.170326	8.670082	6.779068	0.898160

Table 1: Model performance metrics

References

1. Zhu, Michael H., and Suyog Gupta. "Exploring the Efficacy of Pruning for Model Compression." *arXiv*, 13 Nov. 2017. Available at: https://arxiv.org/pdf/1710.01878.
2. Sharma, Manish, and Jamison Heard. "Convolutional Neural Network Compression via Dynamic ..." *arXiv*, 15 Jan. 2024. Available at: https://arxiv.org/pdf/2401.08014.pdf.
3. Molchanov, Pavlo, and Stephen Tyree. "Pruning Convolutional Neural Networks." *arXiv*, 8 June 2017. Available at: https://arxiv.org/pdf/1611.06440.
4. Kusupati, Aditya, and Vivek Ramanujan. "Soft Threshold Weight Reparameterization for Learnable ..." *arXiv*, 22 June 2020. Available at: https://arxiv.org/pdf/2002.03231.
5. Hassibi, Babak, and David G. Stork. Optimal Brain Damage. Advances in Neural Information Processing Systems, *NeurIPS*, 15 Jan. 2024. 1993. https://proceedings.neurips.cc/paper_files/paper/1992/file/303ed4c69846ab36c2904d3ba8573050-Paper.pdf