Pruning Convolutional Neural Networks Using Filter Embeddings

Tamir Shem Tov[1], Jesse Coulson[2], Hongmin Li[1]

[1] Department of Computer Science, California State University, East Bay, Hayward, USA
[2] California State University, Chico, Chico, USA

Abstract

Deep neural networks are often treated as black boxes because it is difficult to understand the role of individual components. This paper explores whether embeddings of convolutional filters can help reveal their function and importance within a Convolutional Neural Network (CNN). We trained a custom CNN on the CIFAR-10 dataset and generated embeddings based on its filter activation values. To analyze the relationships between filters, we built a graph where each filter is represented by a node, with edges formed from cosine similarity. We then applied Node2Vec, a graph-based embedding method, to learn meaningful representations of the filters and store them as embeddings. Our results show that these embeddings capture important structural relationships between filters, providing insight into how the network processes information. Using similarity metrics and clustering techniques on these embeddings, we identified which filters contributed the least to the model's performance and could be pruned without significantly affecting overall accuracy. This study highlights the potential of filter embeddings for improving neural network interpretability and efficiency. Future work could extend this approach to analyze individual neurons and units or explore optimization techniques to reduce memory usage and computational costs further.

1. Introduction

Although neural networks are widely used in modern Artificial Intelligence, understanding their internal mechanisms remains a challenge, often leading to their perception as black boxes. Analyzing how different components of a neural network contributes to its decision making process remains an important task, particularly when trying to understand the role of individual elements within the network's architecture. To address this, it is important to first understand how a neural network is structured. A neural network consists of interconnected neurons, where each neuron takes feature values, processes them, and results in a singular scalar output. In a CNN, the equivalent of a neuron in fully connected layers is often referred to as a filter in convolutional layers. Unlike individual neurons, which process scalar values, a filter operates on local receptive fields within multi-dimensional inputs (such as an image) and produces a feature map. Each filter learns to detect specific patterns, such as edges or textures, by applying a set of shared weights across the input. Together, both convolutional and fully connected layers form the core architecture of a CNN, as they are responsible for processing and transmitting information throughout the network.

This paper examines whether our novel methodology to create embeddings for filters can improve our understanding of CNNs by offering a deeper perspective on their internal organization and whether pruning redundant filters identified from these embeddings can increase efficiency.

Inspired by techniques like word2vec [1], we explore whether our embedding method can uncover meaningful patterns at the filter level of a CNN. To test this idea, we developed a baseline framework using a CNN trained on the CIFAR-10 dataset, constructing filter embeddings based on a feature map's activation values. Our findings suggest that this method reveals filter relationships in a structured way, providing a foundation for using these embeddings for the purpose of pruning filters. Concretely, our main contributions are as follows:

- We propose a method that creates embeddings for filters in a CNN to identify patterns, enabling model optimization.
- We analyze the effectiveness of our embedding approach by comparing it to basic filter averages and find that our method effectively captures filter relationships in the CNN.
- We therefore test our pruning methodology with similar embeddings to remove redundant filters of the model (reducing the number of parameters).
- We compare our methodology derived from embeddings with random pruning and show that our approach is effective in maintaining the model performance while removing the redundant filters, while random pruning suffers dramatic performance loss.

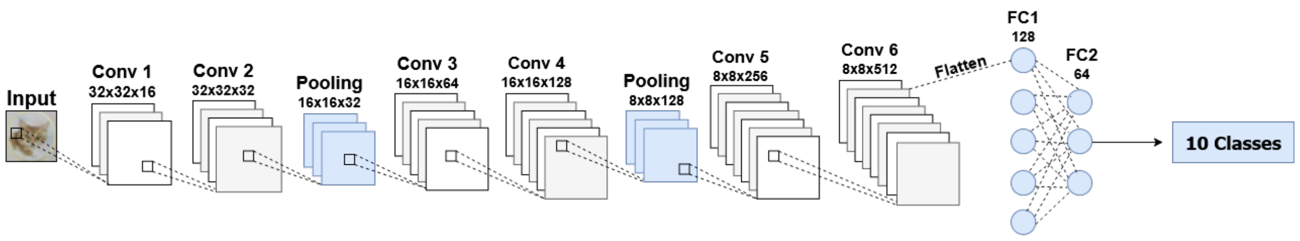

Figure 1. CNN Architecture

2. Literature Review

Several methodologies have been proposed to analyze neural network interpretability. One related work is Network Dissection: Quantifying Interpretability of Deep Visual Representations [2] by Bau et al., which introduces a framework to assign semantic concepts to units in CNNs. This method relies on the Broden dataset, a pixel labeled dataset, to map specific units to predefined concepts. While this approach provides valuable insights, it depends on manual labeling and predefined datasets for assigning these concepts to network units.

Another significant paper is *Quantitative Testing with Concept Activation Vectors (TCAV)* [3] by Kim et al. TCAV evaluates the importance of user defined concepts for specific classifications by generating concept activation vectors. TCAV, like Network Dissection, requires manual input to define and analyze the concepts, limiting its applicability for unsupervised relationship discovery at a granular level.

A different approach, *Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization* [4] by Selvaraju et al., visualizes which portions of an input image contribute to a model's classification decision. Grad-CAM uses activation maps to generate heatmaps, showing the areas most influential for final predictions. While effective for visual interpretability at the feature map level, it does not help quantify these findings for further analysis and use.

Our method differs significantly from these approaches. Unlike Grad-CAM and TCAV, we focus on interpretability at the filter level for optimization. Our method does not rely on manually labeled datasets to find relationships. Instead, we construct embeddings for individual filters using their feature map activation values, allowing even more flexible forms of analysis.

We also looked for research related to neural network pruning and we found a comprehensive survey *A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations* [5] by Cheng et al., which provides an extensive review and comparative analysis of multiple pruning techniques, categorizing them based on criteria such as structured vs. unstructured pruning, pruning before, during, and after training, and integration with other compression methods. Structured pruning methods, which remove entire filters or channels, is very close to our goal of eliminating redundant features while preserving network functionality. Their analysis also highlights the trade off between one-shot and iterative pruning, with iterative methods often yielding better performance at the cost of increased computation resources.

3. Methodology

Our methodology starts with building a CNN. Our architecture is relatively straightforward, consisting of six convolutional layers followed by three fully connected layers, as shown in Figure 1. The CNN is trained on the CIFAR-10 dataset, which contains low-resolution color images with ten different classes. After training the model, the next step involves collecting the average value of units in a feature map, then storing said values in a matrix herein referred to as the activation matrix. For the creation of the activation matrix, we used 10,000 images, with 1000 images being selected from each class. In this matrix, each row corresponds to a filter, each column represents an image, and each cell holds the average value of all units in a feature map when classifying the image, as shown in Figure 2. We used PyTorch hooks and ran a sample of images from the activation dataset through the network to obtain these values.

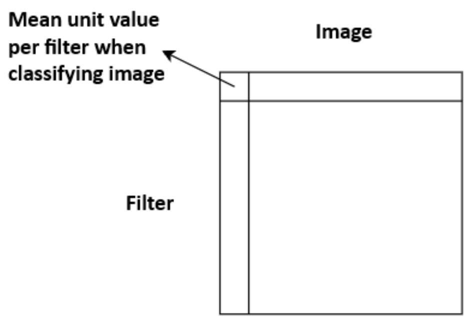

Figure 2. Activation Matrix Architecture

Using the activation matrix, we construct a graph using NetworkX. From the activation matrix, we compute the cosine similarity between every pair of rows. When the similarity between two filters exceeds a threshold of .99, an edge is created between their corresponding nodes in the graph. In this graph, nodes represent individual filters, and edges represent strong similarities between their average activation values across the 10,000 images used. The resulting graph, as shown in Figure 3, is a visual representation of these relationships.

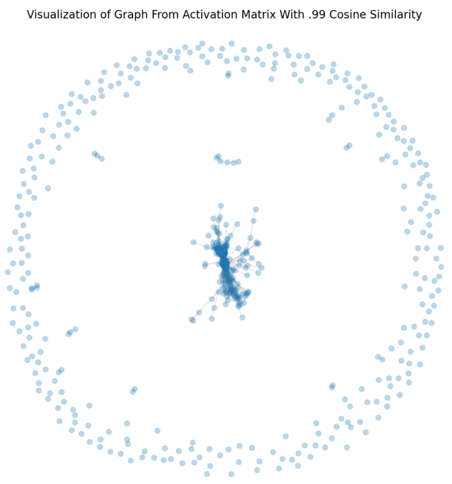
Visualization of Graph From Activation Matrix With .99 Cosine Similarity

Figure 3. Graph Visualization of Activation Matrix

The graph shows dense clusters where many nodes are connected to each other, as well as more isolated nodes and sparsely connected sections. Due to the graph's size, with over 1000 nodes and dense connections, some relationships are hard to see in a single visualization. However, you can still see distinct clusters of nodes forming and overall patterns of relationships being formed. To generate embeddings for each node, we apply Node2Vec [6] on this graph. The random walk parameters in Node2Vec are set to help encourage exploration of the graph. The return parameter p = 0.5 increases the likelihood of revisiting the previous node, while the in-out parameter q = 1 keeps the walk unbiased between staying close or moving farther from the starting position. The dimensionality parameter chosen for Node2Vec is 128, maintaining a balance between capturing enough information about the relationships between embeddings and avoiding excessive noise.

To evaluate the quality of the embeddings, we compare their clustering performance against the clustering results of the original filter activation vectors from the activation matrix. For the comparison, we apply KMeans clustering with k = 10 and use silhouette score to assess the compactness and separation of the clusters.

To proceed with pruning redundant filters, we first calculate the z-score of each embedding. We then select a certain percentile, such as 10%, identifying embeddings whose z-scores are among the 10% most similar. The rationale behind focusing on the most similar embeddings is that filters with highly similar activations are often redundant, meaning that their removal does not significantly impact overall accuracy.

Next, we cluster these z-score values using SciPy's fcluster. After clustering, we compute the mean cluster size across all identified clusters. We then prune filters from the largest clusters, reducing their size until they match the mean cluster size. This approach ensures that we do not excessively prune small clusters, as we observed that removing filters from clusters with very few members drastically reduces model accuracy. By prioritizing the reduction of larger clusters, we eliminate a substantial number of redundant filters while maintaining overall performance.

Using the identified redundant filters, we proceed to prune them from the model's architecture by zeroing out their weights and biases. Finally, we evaluate the pruned model by testing its accuracy against the unpruned model's accuracy. Finally, we compare our methodology to the baseline of random filter dropout of the same number of filters.

4. Data

To train our model, we used the CIFAR-10 dataset available through PyTorch's data set library, example images shown in Figure 4. The CIFAR-10 dataset is made of 60,000 32x32 images with three color channels, split across ten different classes. For our purposes, we used 40,000 images to train our model, 10,000 to capture values for our activation matrix as described in our methodology, and 10,000 images for testing.

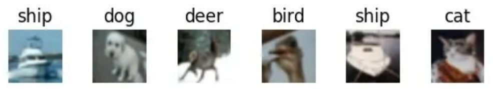

Figure 4. Example CIFAR-10 Images

The final activation matrix has dimensions of 1008 rows (filters) and 10,000 columns (images). This matrix serves as the foundational data for generating embeddings, which form the basis for our pruning methodology.

5. Results

To evaluate the effectiveness of our embedding methodology, we compared the clusters derived from the embeddings with clusters made from the filter activation values. Using KMeans clustering with k = 10, we first calculated the Adjusted Rand Index (ARI) to quantify the similarity between the two clustering results. The ARI score was 0.1155, showing a minimal overlap between the clusters formed by the two methods. On the ARI scale, a value of 0 represents random cluster assignments, while 1 is a perfect match. This score suggests that the embeddings capture significantly different patterns than the raw activation values.

To assess clustering quality, we computed the silhouette score for both methods. The silhouette score measures how effective the clusters are, ranging from -1 (poor clustering) to 1 (perfect clustering). The raw activation values produced a silhouette score of 0.262, indicating weak clustering, while the embeddings achieved a significantly higher silhouette score of 0.567. Although this value is a bit off from the ideal of 1, it shows that the embeddings capture much more cohesive relationships, with a noticeable improvement in clustering quality. You can see this more in Figures 5 and 6, which show the t-SNE plots of the clusters of both methods. The embeddings produce clusters with significantly less overlap compared to the original filter activation values, highlighting their effectiveness in distinguishing distinct filter behaviors.

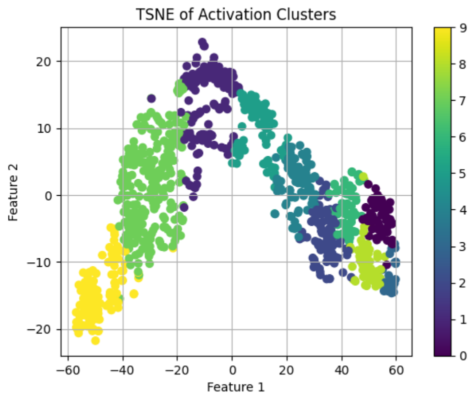

Figure 5. TSNE Visualization of Activation Clusters

To evaluate the effectiveness of our pruning methodology, we measured the number of filters deactivated and assessed the resulting impact on accuracy, comparing it against a random dropout approach. The baseline accuracy of the model without pruning is 75.61%. As shown in Table 1, the "Fraction" column represents the percentile used to determine which filters to prune. However, this does not correspond to pruning exactly that percentage of filters due to our safeguard of avoiding excessive pruning beyond the average cluster size, as explained previously in our methodology.

The table demonstrates that as our pruning methodology removes more filters, the accuracy declines at a steady rate. In contrast, random dropout performs worse in every scenario. When pruning almost any number of significant filters, random dropout achieves only 10% accuracy, equivalent to random guessing, due to the issue highlighted in our methodology, where some filters are crucial for maintaining performance. Our approach mitigates this by selectively pruning redundant filters while preserving essential ones.

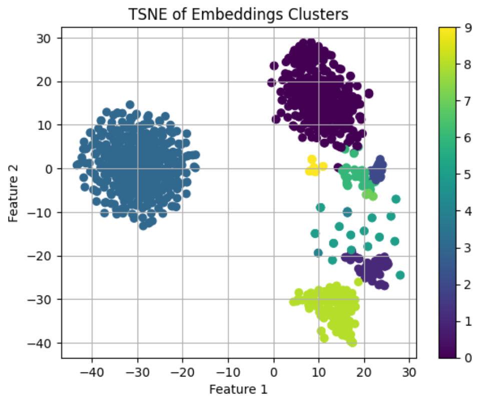

Figure 6. TSNE Visualization of Embedding Clusters

Interestingly, in the Fraction = 0.3 case, random dropout briefly recovers to 72.57% accuracy due to chance. However, our method produces a consistent accuracy decline, starting at 75.42% with 4.96% of filters pruned and gradually decreasing to 11.10% accuracy with 50.60% of filters removed. Going beyond a fraction value of 0.3, random dropout plateaus at 10% accuracy, while our methodology maintains a structured decrease, which demonstrates its effectiveness in selectively removing redundant filters while preserving overall performance for as long as possible. Additionally, this suggests that our embedding method did manage to capture the importance of filters relative to each other in the CNN model we built.

6. Future Work

While our results establish a strong baseline, there are many directions to further explore our embedding methodology. One promising avenue is working at a more granular level with units and neurons. Although analyzing the lowest level of a neural network may introduce challenges, such as larger graph sizes and longer embedding computation times, it could provide deeper insights into neural network-based models, leading to more effective pruning. Currently, we focus on filters, which are inherently important as they encompass multiple units. By applying our methodology at the unit level, we could treat each unit as a standalone representation, offering a clearer understanding of its impact on the network's architecture.

Another important direction is physical pruning, which is the reconstruction of the model's architecture without

Table 1. Results of Embeddings Based Pruning vs Random Pruning

Fraction	Number Filters Dropped	Percent Filters Dropped	Pruned Accuracy	Random Drop Accuracy
0.1	50	4.96%	75.42%	69.08%
0.2	100	9.92%	75.05%	10.00%
0.3	151	14.98%	74.82%	72.57%
0.4	252	25.00%	74.01%	10.00%
0.5	349	34.62%	52.02%	10.00%
0.6	373	37.00%	28.50%	10.00%
0.7	384	38.09%	27.69%	10.00%
0.8	407	40.38%	19.94%	10.00%
0.9	510	50.60%	11.10%	10.00%

the redundant components identified. While masking techniques that zero out filter weights are useful for experimental validation, they are not the ultimate goal of this methodology. The primary objective is to physically prune the model, achieving strong results while reducing computational costs and memory usage.

Finally, once this method has been thoroughly tested on traditional neural networks, extending it to more advanced network based architectures, such as transformers, would be a valuable next step to gain deeper insights into more complex structures.

7. Discussion

Our results demonstrate that filter embeddings reveal meaningful relationships between filters, which can be used to prune CNN models more effectively than traditional random dropout methods. This finding aligns with our early assumptions of how embedding techniques could improve on neural networks' interpretability. In addition, our method offers a structured and automatic way to discover these relationships, with consistent pruning results across multiple experiments.

The implications of these findings can be broken down into two main points. First, embeddings could serve as a valuable tool for enhancing neural network interpretability. By revealing how filters interact with one another, we gain deeper insights into the features the network learns and how these features contribute to the final decision making process. Second, the ability to prune models more effectively presents an opportunity to optimize the computational efficiency and memory usage of neural networks, which is beneficial in resource constrained environments.

8. Conclusion

Through our experimentation with our novel method, this research has demonstrated that filter embeddings provide deeper insights into the inner workings of Convolutional Neural Networks. By taking the mean of a feature map's activation values and storing them in a matrix referred to as the activation matrix, we uncovered relationships between filters that could be leveraged to prune the model with significantly higher consistency than the baseline random dropout methodology.

This finding establishes a foundation for future work, showing that we can explore new avenues such as working at a more granular level to prune neurons or units which can improve interpretability, physically pruning models to optimize computational and memory costs, applying the method to more complex CNN architectures like AlexNet or ResNet, or even extending it to more complex technologies such as transformers.

References

[1] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space, 2013. URL `https://arxiv.org/abs/1301.3781`.

[2] Bau D, Zhou B, Khosla A, Oliva A, Torralba A. Network dissection: Quantifying interpretability of deep visual representations, 2017. URL `https://arxiv.org/abs/1704.05796`.

[3] Kim B, Wattenberg M, Gilmer J, Cai C, Wexler J, Viegas F, Sayres R. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav), 2018. URL `https://arxiv.org/abs/1711.11279`.

[4] Selvaraju RR, Cogswell M, Das A, Vedantam R, Parikh D, Batra D. Grad-cam: Visual explanations from deep networks via gradient-based localization. International Journal of Computer Vision October 2019;128(2):336–359. ISSN 1573-1405. URL `http://dx.doi.org/10.1007/s11263-019-01228-7`.

[5] Cheng H, Zhang M, Shi JQ. A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations, 2024. URL `https://arxiv.org/abs/2308.06767`.

[6] Grover A, Leskovec J. node2vec: Scalable feature learning for networks, 2016. URL `https://arxiv.org/abs/1607.00653`.