Rushil Desai
EML

Final Project Report

Runtime Neural Pruning

Overview of the Methodology

The paper shows that preserving the network's full capability and dynamically pruning it based on input images is preferable for achieving a better balance between speed and accuracy compared to static pruning methods.

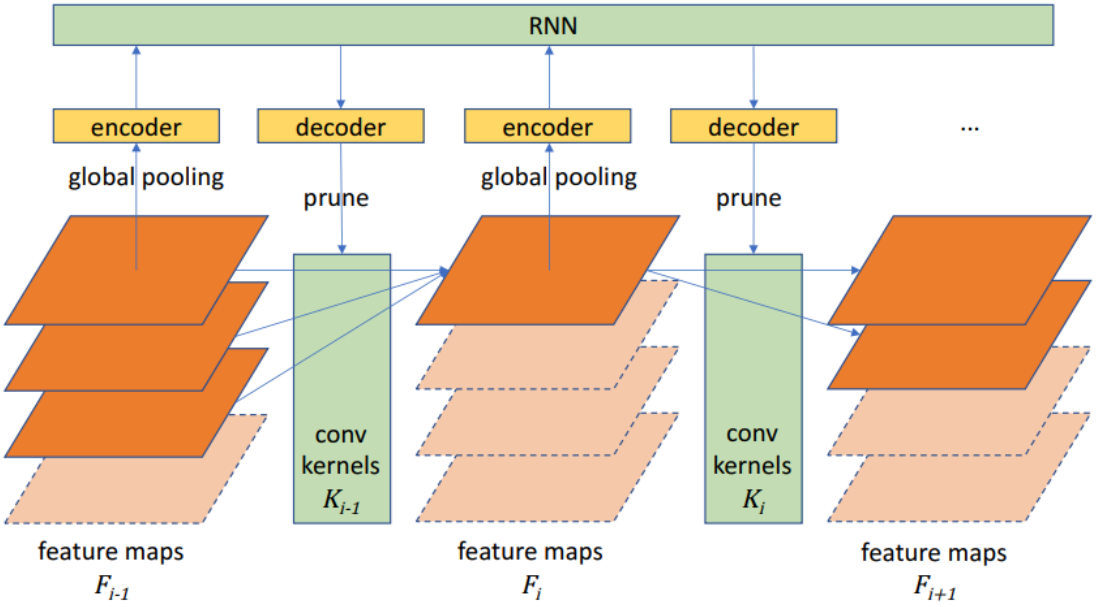

Figure 1: Framework diagram of RNP. Taken from the paper [1]

The overall framework of our RNP is shown in Figure 1. RNP comprises two sub-networks: the backbone CNN network and the decision network. The decision network is responsible for determining the pruning of convolution kernels based on the input image and current feature maps. The decision network consists of an encoder-RNN-decoder structure. The paper adopts deep Q-learning into the RNP framework by formulating the elements as:

1. **State:** Given a set of feature maps, an encoder projects the pooled feature into a fixed-length embedding. This is then passed through the RNN structure to produce the embedded state information.
2. **Action:** This determines which output channels are calculated and which to prune. Formally, taking action a_i retains the channels from 1 through i and prunes the other channels.
3. **Reward:** The reward of each action, a_i, taken at the t-th step is defined as:

$$r_t(a_i) = \begin{cases} -\alpha L_{cls} + (i-1) \times p, & \text{if inference terminates } (t = m-1) \\ (i-1) \times p, & \text{otherwise } (t < m-1) \end{cases}$$

The criterion for training to keep the decision network self-consistent:

$$\min_{\theta} L_{re} = \mathbb{E}[r(s_t, a_i) + \gamma \max_{a_i} Q(s_{t+1}, a_i) - Q(s_t, a_i)]^2$$

Rushil Desai
EML

The backbone CNN network and decision network are trained alternately.

Why is the research paper important and deserving of reproduction?

Neural networks are becoming increasingly complex, requiring significant computational resources for training and inference. Runtime neural pruning offers a method to reduce the computational burden by dynamically removing less relevant neurons or connections during inference, leading to more efficient models without compromising accuracy. Unlike existing methods that produce a fixed-pruned model for deployment, the “Runtime Neural Pruning” paper preserves the full ability of the original network and prunes the neural network according to the input image and current feature maps. The research paper successfully shows that dynamically pruning it based on input images is preferable for achieving better overall performance as compared to other methods.

Reproducing the research on runtime neural pruning holds promise for addressing several key challenges in deep learning, including efficiency, scalability, adaptability, and interpretability, thereby advancing the state-of-the-art and making AI more accessible and impactful across various domains.

What extended applications can be enabled by reproducing this work?

The open-source code can be used by developers to perform runtime pruning on their own model and dataset so that they can deploy it on embedded systems with low computational resources. To further this point, applications in resource-constrained environments such as robotics, drones, and IoT devices can benefit from the ability to run deep learning models efficiently. By using runtime pruning, these devices can conserve energy and computational resources, leading to shorter run times and improved performance. Additionally, researchers can use the open-source code as a foundation for exploring new research directions in the field of model optimization and deep learning. This can hopefully facilitate faster iteration and exploration of dynamic pruning techniques.

In addition to reproducing the results of this research paper, what other applications did you evaluate to demonstrate the wide usage of your reproduced framework?

Apart from reproducing the results, I evaluated the framework on additional CNN architectures to make sure the framework is robust and applicable across a wide range of scenarios. By conducting these evaluations, I ensured that the framework performed reliably and effectively under different conditions.
To elaborate on this, I experimented with different numbers of CNN layers and also studied the effect of other layers such as dropout and maxpool. The outcomes of these experiments are shown in the “Results” section below.

What is your implementation solution and Results?

Tools: PyTorch
Datasets: CIFAR-10 [2], and a custom 3-category dataset using Labeled Faces in the Wild [3] dataset which contains 3000 images of males, 3000 images of females and 3000 images of random backgrounds.

Rushil Desai
EML

The first experiment, which was run on the custom 3-category dataset, shows that the RNP framework successfully learns to prune more channels for “easy to classify” images (background images, in this case) and prunes fewer channels for the more difficult images (images of males and females, in this case). As we can see in Table 1, the number of computations for background images is considerably lower than the Male and female images. This difference can be seen clearer if we focus on just one convolution layer (conv3, in this case).

	Average Multiplions (mil.) per Category			
Network	Average	Male	Female	Background
Whole Network	2.5	2.9	2.9	1.6
Only Conv3 layer	0.4	0.54	0.52	0.15

Table 1. The computation numbers for both, the whole network and the fully pruned convolutional layer conv3.

The next experiment is on the CIFAR-10 dataset. Here, I used a four-layer convolutional network with 3 × 3 kernels. I compared the RNP framework with a vanilla baseline network, where the convolutional channels were reduced directly at the beginning and not dynamically. The full unpruned model has 25399296 (25M) multiplications for one batch of images. We can vary the amount pruned by the RNP framework by varying the hyperparameter ***p*** (given in the reward equation above). This can be changed in the CONSTANTS.py file. The results are shown in Figure 2. We can see that at 4M multiplications, the RNP method performs a lot better than the Vanilla pruned method, and this trend is seen as we increase the number of multiplications.

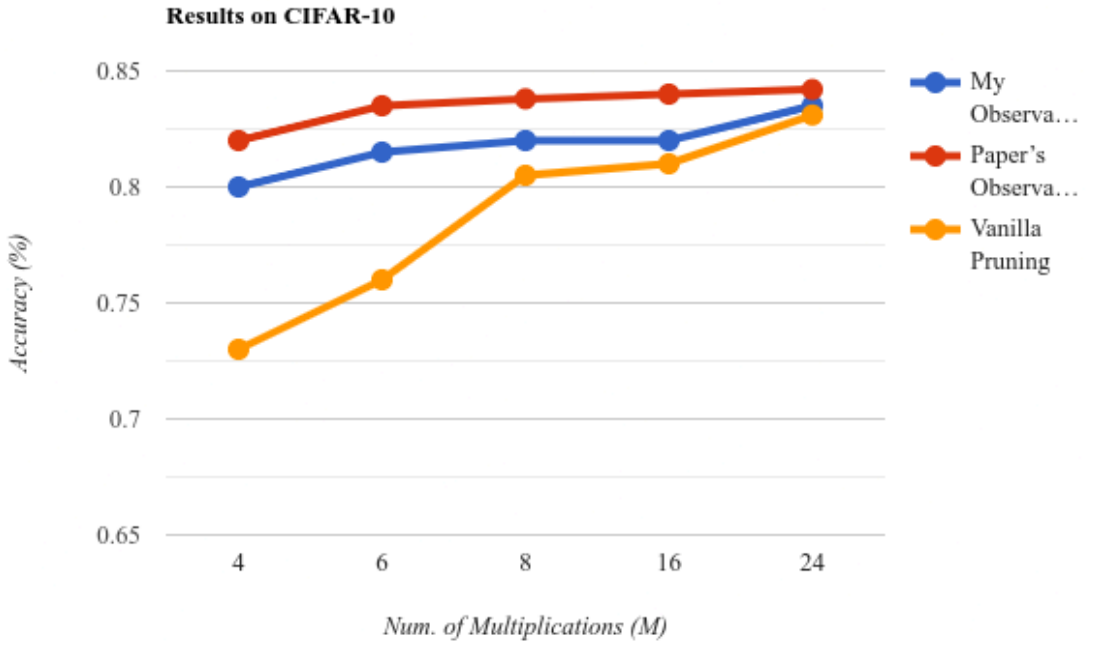

Figure 2: The results on CIFAR-10. The rightmost point is the full model and the leftmost is 25% of the model. RNP outperforms naive channel reduction models consistently by a large margin.

Another set of experiments was performed by changing the underlying CNN network. I ensured that the framework performed reliably and effectively under different conditions by conducting these evaluations.

Model Description	Total Multiplications Before RNP (M)	Accuracy (%)	Total Multiplications after RNP (with p = -0.1) (M)	Accuracy After RNP (%)
4 Convolution Layers	25.4	88.45	8.0	87.38
6 Convolution Layers	51.2	90.07	12.4	92.1
6 Convolution Layers with a Dropout After Each Layer	51.2	93.2	10.7	93.7
8 Convolution Layers	285.8	82.1	63.4	82.7
8 Convolution Layers with a Dropout After Each Layer	285.8	85.4	60.5	86.2

Table 2: Number of multiplications and accuracy across different model structures.

In Table 2, we can see that the total multiplications after RNP is consistently a lot less as compared to the full model, while not degrading the accuracy.
The results of these experiments provided valuable insights into the performance of the framework under different configurations. By conducting these evaluations, we were able to ensure that the framework could be deployed with confidence in various real-world scenarios.

Rushil Desai
EML

References

1. Lin, J., Rao, Y., Lu, J., & Zhou, J. (2017). Runtime Neural Pruning. *Neural Information Processing Systems*.
2. Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009
3. Gary B Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007.