Google AI Blog

The latest news from Google AI

MorphNet: Towards Faster and Smaller Neural Networks

Wednesday, April 17, 2019

Posted by Andrew Poon, Senior Software Engineer and Dhyanesh Narayanan, Product Manager, Google AI Perception

Deep neural networks (DNNs) have demonstrated remarkable effectiveness in solving hard problems of practical relevance such as image classification, text recognition and speech transcription. However, designing a suitable DNN architecture for a given problem continues to be a challenging task. Given the large search space of possible architectures, designing a network from scratch for your specific application can be prohibitively expensive in terms of computational resources and time. Approaches such as Neural Architecture Search and AdaNet use machine learning to search the design space in order to find improved architectures. An alternative is to take an existing architecture for a similar problem and, in one shot, optimize it for the task at hand.

Here we describe MorphNet, a sophisticated technique for neural network model refinement, which takes the latter approach. Originally presented in our paper, “MorphNet: Fast & Simple Resource-Constrained Structure Learning of Deep Networks”, MorphNet takes an existing neural network as input and produces a new neural network that is smaller, faster, and yields better performance tailored to a new problem. We've applied the technique to Google-scale problems to design production-serving networks that are both smaller and more accurate, and now we have open sourced the TensorFlow implementation of MorphNet to the community so that you can use it to make your models more efficient.

How it Works

MorphNet optimizes a neural network through a cycle of *shrinking* and *expanding* phases. In the shrinking phase, MorphNet identifies inefficient neurons and prunes them from the network by applying a sparsifying regularizer such that the total loss function of the network includes a cost for each neuron. However, rather than applying a uniform cost per neuron, MorphNet calculates a neuron cost with respect to the targeted resource. As training progresses, the optimizer is aware of the resource cost when calculating gradients, and thus learns which neurons are resource-efficient and which can be removed.

As an example, consider how MorphNet calculates the computation cost (e.g., FLOPs) of a neural network. For simplicity, let's think of a neural network layer represented as a matrix multiplication. In this case, the layer has 2 inputs (*x_n*), 6 weights (*a*,*b*,...,*f*), and 3 outputs (*y_n*; neurons). Using the standard textbook method of multiplying rows and columns, you can work out that evaluating this layer requires 6 multiplications.

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a & b \\ 0 & d \\ e & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Computation cost of neurons.

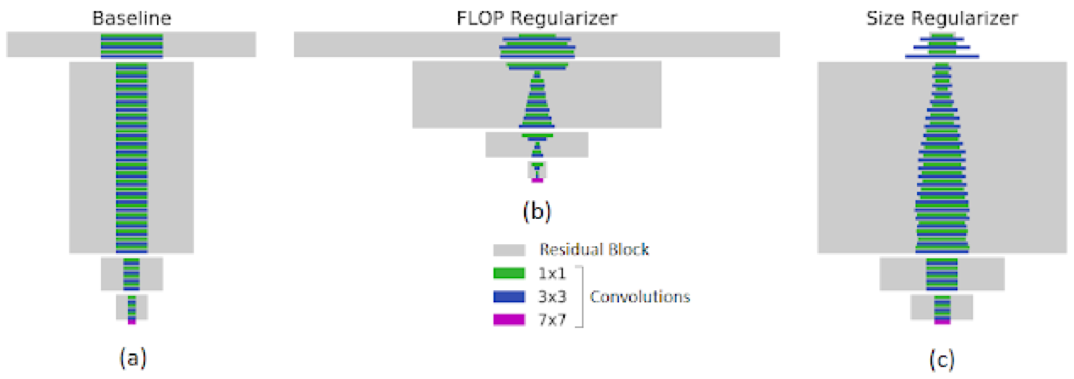
MorphNet calculates this as the product of input count and output count. Note that although the example on the left shows weight sparsity where two of the weights are 0, we still need to perform all the multiplications to evaluate this layer. However, the middle example shows structured sparsity, where all the weights in the row for neuron *y_n* are 0. MorphNet recognizes that the new output count for this layer is 2, and the number of multiplications for this layer dropped from 6 to 4. Using this idea, MorphNet can determine the incremental cost of every neuron in the network to produce a more efficient model (right) where neuron *y_3* has been removed.

In the *expanding phase*, we use a width multiplier to uniformly expand all layer sizes. For example, if we expand by 50%, then an inefficient layer that started with 100 neurons and shrank to 10 would only expand back to 15, while an important layer that only shrank to 80 neurons might expand to 120 and have more resources with which to work. The net effect is re-allocation of computational resources from less efficient parts of the network to parts of the network where they might be more useful.

One could halt MorphNet after the shrinking phase to simply cut back the network to meet a tighter resource budget. This results in a more efficient network in terms of the targeted cost, but can sometimes yield a degradation in accuracy. Alternatively, the user could also complete the expansion phase, which would match the original target resource cost but with improved accuracy. We'll cover an example of this full implementation later.

Why MorphNet?
There are four key value propositions offered by MorphNet:

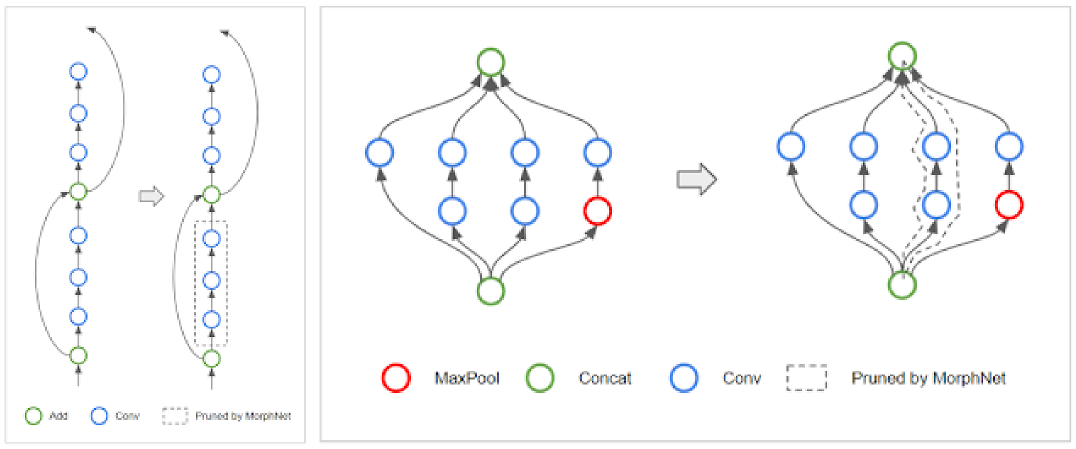
- **Targeted Regularization:** The approach that MorphNet takes towards regularization is more intentional than other sparsifying regularizers. In particular, the MorphNet approach to induce better sparsification is ***targeted*** at the reduction of a particular resource (such as FLOPs per inference or model size). This enables better control of the network structures induced by MorphNet, which can be markedly different depending on the application domain and associated constraints. For example, the left panel of the figure below presents a baseline network with the commonly used ResNet-101 architecture trained on JFT. The structures generated by MorphNet when targeting FLOPs (center, with 40% fewer FLOPs) or model size (right, with 43% fewer weights) are dramatically different. When optimizing for computation cost, higher-resolution neurons in the lower layers of the network tend to be pruned more than lower-resolution neurons in the upper layers. When targeting smaller model size, the pruning tradeoff is the opposite.



Targeted Regularization by MorphNet. Rectangle width is proportional to the number of channels in the layer. The purple bar at the bottom is the input layer. **Left**: Baseline network used as input to MorphNet. **Center**: Output applying FLOP regularizer. **Right**: Output applying size regularizer.

MorphNet stands out as one of the few solutions available that can target a particular parameter for optimization. This enables it to target parameters for a specific implementation. For example, one could target *latency* as a first-order optimization parameter in a principled manner by incorporating device-specific compute-time and memory-time.

- **Topology Morphing:** As MorphNet learns the number of neurons per layer, the algorithm could encounter a special case of sparsifying all the neurons in a layer. When a layer has 0 neurons, this effectively changes the topology of the network by cutting the affected branch from the network. For example, in the case of a ResNet architecture, MorphNet might keep the skip-connection but remove the residual block as shown below (left). For Inception-style architectures, MorphNet might remove entire parallel towers as shown on the right.

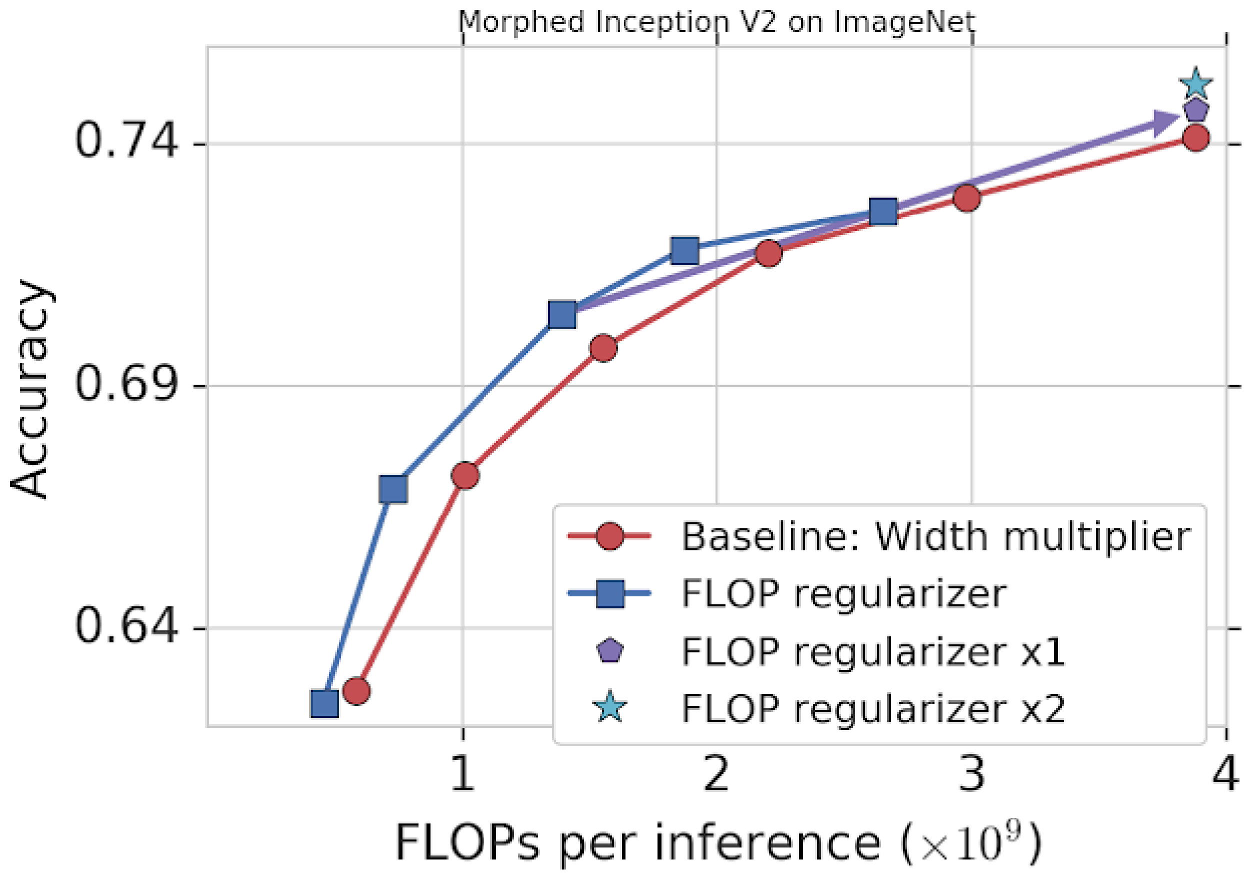


Left: MorphNet can remove residual connections in ResNet-style networks. **Right**: It can also remove parallel towers in Inception-style networks.

- **Scalability:** MorphNet learns the new structure in a single training run and is a great approach when your training budget is limited. MorphNet can also be applied directly to expensive networks and datasets. For example, in the comparison above, MorphNet was applied directly to ResNet-101, which was originally trained on JFT at a cost of 100s of GPU-months.
- **Portability:** MorphNet produces networks that are "portable" in the sense that they are intended to be retrained from scratch and the weights are not tied to the

architecture learning procedure. You don't have to worry about copying checkpoints or following special training recipes. Simply train your new network as you normally would!

Morphing Networks
As a demonstration, we applied MorphNet to Inception V2 trained on ImageNet by targeting FLOPs (see below). The baseline approach is to use a width multiplier to trade off accuracy and FLOPs by uniformly scaling down the number of outputs for each convolution (red). The MorphNet approach targets FLOPs directly and produces a better trade-off curve when shrinking the model (blue). In this case, FLOP cost is reduced 11% to 15% with the same accuracy as compared to the baseline.



MorphNet applied to Inception V2 on ImageNet. Applying the flop regularizer alone (blue) improves the performance relative to baseline (red) by 11-15%. A full cycle, including both the regularizer and width multiplier, yields an increase in accuracy for the same cost ("x1"; purple), with continued improvement from a second cycle ("x2"; cyan).

At this point, you could choose one of the MorphNet networks to meet a smaller FLOP budget. Alternatively, you could complete the cycle by expanding the network back to the original FLOP cost to achieve better accuracy for the same cost (purple). Repeating the MorphNet shrink/expand cycle again results in another accuracy increase (cyan), leading to a total accuracy gain of 1.1%.

Conclusion
We've applied MorphNet to several production-scale image processing models at Google. Using MorphNet resulted in significant reduction in model-size/FLOPs with little to no loss in quality. We invite you to try MorphNet—the open source TensorFlow implementation can be found here, and you can also read the MorphNet paper for more details.

Acknowledgements

This project is a joint effort of the core team including: Elad Eban, Ariel Gordon, Max Moroz, Yair Movshovitz-Attias, and Andrew Poon. We also extend a special thanks to our collaborators, residents and interns: Shraman Ray Chaudhuri, Bo Chen, Edward Choi, Jesse Dodge, Yonatan Geifman, Hernan Moraldo, Ofir Nachum, Hao Wu, and Tien-Ju Yang for their contributions to this project.