Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning

1 Citation

Szegedy, Christian, et al. "Inception-v4, inception-resnet and the impact of residual connections on learning." AAAI. Vol. 4. 2017.

https://arxiv.org/pdf/1602.07261.pdf

2 Abstract

We combine an Inception network with residual units. This makes training a lot faster, but doesn't really increase accuracy that much. We present streamlined architectures for residual and non-residual Inception networks that give state of the art single frame classification results on ImageNet 2012 data. We show how scaling activations helps train wide Inception residual networks. Our ensemble of 3 residuals and 1 Inception-v4 gets 3.08% top-5 error on ImageNet classification.

3 Introduction

Residual blocks supposedly help train deep networks. Inception networks are deep. Let's see if residual blocks help Inception networks.

We also simplify the Inception architecture, because previous version have some baggage because we had to account for partitioning in the DistBelief neural network system. Now we train on TensorFlow, so we don't have to worry about those performance considerations.

Our baseline is Inception-v3. We create Inception-v4, Inception-ResNet-v1, and Inception-ResNet-v2.

4 Related Work

Residual blocks help train deep models. We can train Inception networks without them, but they do make training a lot faster.

Our original Inception network was refined to incorporate batch normalization (v2) and some factorization ideas (v3).

5 Architectural Choices

We used to make our architecture in a way that could be easily partitioned for the DistBelief system. Now we don't worry about that. We used to avoid making too many architecture changes at once. Now we don't worry about that.

For residual Inception blocks, we use a cheaper block. We also need a 1×1 conv layer to scale up the dimension (the Inception block reduces the dimension) so that we can add it to the identity mapping. Our Inception-ResNet-v1 has similar compute cost to Inception-v3 and our Inception-ResNet-v2 has similar compute cost to Inception-v4.

For our Inception-ResNet networks, we don't do batch normalization on the summations at the end of the residual blocks. This was to avoid using too much GPU memory during training and instead use that memory for more Inception blocks.

Some of our deep networks just died during training (last layer before average pooling just started outputting zeros). Lowering the learning rate and using batch normalization did not help, so we just scale the residuals down by 0.1 or 0.3 to get around this issue. It didn't seem to hurt accuracy (we tested this on smaller networks). It just stabilized training.

6 Training Methodology

We use TensorFlow and train on 20 NVIDIA Kepler machines. We used RMSPROP with decay = 0.9 and $\epsilon = 1.0$. Our learning rate was 0.045 and decayed every two epochs with decay rate 0.94.

7 Experimental Results

For top-1/top-5, single-crop, single-model error, the models (from worst to best) are Inception-v3 (21.2/5.6%), Inception-ResNet-v1 (21.3/5.5%), Inception-v4 (20.0/5.0%), and Inception-ResNet-v2 (19.9/4.9%).

8 Conclusion

We present Inception-ResNet-v1 (like Inception-v3, but with residual blocks), Inception-v4 (better Inception network), Inception-ResNet-v2 (like Inception-v4, but with residual blocks) and all of them beat Inception-v3.

9 Summary of Figures

Most of this paper was actually just diagrams of neural network building blocks. I'll summarize the diagram here.

There's a "stem" network common to both Inception-v4 and Inception-ResNet-v2. It starts at the image and goes through three 3×3 conv layers. It branches off into a max-pooling and conv layer that join at filter concatenation. We branch off again into two paths, each with a few conv layers, coming back into another filter concat. We branch off into two paths for a final time into max-pooling and conv and join at the filter concat.

For Inception-v4, we have three kinds of Inception blocks (called A, B, and C). Each of them starts at a filter concatenation and ends at a filter concatenation. Starting at the first filter concat, they branch off into 4 paths that all merge at the second filter concat. The first two branches are the same among all of them (except for different number of filters). The first branch is average pooling and 1×1 conv. The second branch is 1×1 conv. The third and fourth branches differ for all of them, using different order and sizes for conv layers.

There are two reduction modules for Inception-v4. These are called Reduction-A and Reduction-B blocks. Both start and end at filter concats. They both have three paths to get from one filter concat to the other. The first branch has max pooling, and the other two have a few conv layers.

With those pieces described, here's Inception-v4. We first have the stem. Then we go through several Inception-A blocks followed by a Reduction-A. We then go through several Inception-B blocks followed by a Reduction-B. We then go through a few Inception-C blocks. We finish with average pooling, dropout, and softmax.

We described the Inception-A, B, and C blocks above, so now let's describe the Inception-ResNet-v1 A, B, C blocks and the Inception-ResNet-v2 A, B, and C blocks. For all of them, we start and end at a ReLU activation. For all of them, there are 3 to 4 branches off the starting ReLU activation. The first branch is always a shortcut to the addition block. The remaining branches just have a different series of conv layers and all merge into a filter concat. That filter concat goes into a 1×1 conv layer (so we can upscale). The resulting upscaled maps go into the addition (where they join with the shortcut) and the result then goes into the ReLU activation.

We also tweak the Reduction-B block to get a block for Inception-ResNet-v1 and Inception-ResNet-v2.

The Inception-ResNet architecture is the same as the Inception-v4 architecture, but where we use the appropriate Inception A, B, C and Reduction A, B blocks.

When we do activation scaling, we apply the scaling right before the inception branch enters the addition node to meet the shortcut connection.