Rethinking the Inception Architecture for Computer Vision

1 Citation

Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

`https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Szegedy_Rethinking_the_Inception_CVPR_2016_paper.pdf`

2 Abstract

When designing a CNN, you want good accuracy while keeping compute cost and number of parameters low. We outline principles to acheive this and present Inception-v3, which gets top-1 (top-5) single frame/single crop error of 21.2% (5.6%) on ImageNet classification with 5B multiply-adds and under 25M parameters.

3 Introduction

VGG is great because it's architecturally simple. GoogLeNet is computationally cheaper (7M parameters vs. 180M parameters in VGG) and gets better accuracy. The problem with GoogLeNet is that it's not clear how to extend or modify the architecture. Let's discuss the principles guiding the GoogLeNet architecture.

4 General Design Principles

First, avoid representational bottlenecks. Think of a neural network as a DAG, and make a cut that separates outputs and inputs. The amount of information flowing through the cut is the representational size. The representation size should gradually decrease as we move from inputs to outputs. It should never suddenly drop.

Second, higher dimensional representations are easier to train. That is, if the number of channels is high, features will be disentangled, making them easier to train.

Third, spatial aggregation over lower dimensions does not lose much representational power. That is, if you want to do a 3×3 conv, feel free to put your input through a 1×1 conv to decrease the number of channels. We think this is because many channels tend to be correlated.

Fourth, balance width (filters per stage) and depth of network. Increase both in parallel when adding compute cost to your network.

5 Factorizing Convolutions with Large Filter Size

Instead of using a 5×5 conv, use two 3×3 convs. They have fewer parameters and the same receptive field size.

In fact, you can go further and replace the 3×3 conv with a 3×1 conv followed by a 1×3 conv. Don't use this on earlier layers though - 3×3 conv tends to work better there.

6 Utility of Auxiliary Classifiers

An auxiliary classifier branches off the middle of the network and does classification. This way, we can propogate the loss down from multiple places in the network, which fights the vanishing gradient problem. We find that they don't really make training faster, but they do regularize the network (especially if you use batch normalization or dropout in the classifier).

7 Efficient Grid Size Reduction

Suppose we want to reduce the grid size of an input without causing a representational bottleneck. In that case, we need to decrease the grid size while increasing the number of channels. For example, a $d \times d \times k$ input should become a $\frac{d}{2} \times \frac{d}{2} \times 2k$ output. You could do this with a 1×1 conv layer and pooling, but that's compute intensive. We actually propose a grid reduction module that does this efficiently. Basically, it branches the input off into three branches. The first applies two 3×3 convs, the second applies one 3×3 conv, and the third applies pooling. The branches all get concatenated together. The compute savings come from the fact that convolutions have stride > 1 and that no branch has the full $2k$ layers.

8 Inception-v3

We start off with three 3×3 conv layers, pooling, and another three 3×3 conv layers.

Then, we go through three Inception-A blocks. We follow this with a grid reduction-A block.

Then we go through five Inception-B blocks and a grid reduction-B block.

Then we go through two Inception-C blocks.

We finish with pooling, a linear layer, and softmax.

Our network is 42 layers deep and 2.5 more compute intensive than GoogLeNet.

9 Model Regularization and Label Smoothing

In classification, the ground truth has one class given weight 1 and the others given weight 0. This is bad because it can encourage the neural network to full probability to the correct logit (right before the softmax) and make it much larger than the others. This can cause overfitting because the model is overconfident. To mitigate this, we propose smoothing the ground truth labels where the correct class gets weight $1 - \epsilon$ and the remaining ϵ probability mass is distributed over the other classes. This is called a label smoothing regularizer. We find that it improves top-1/top-5 error by about 0.2%.

10 Training Methodology

We train on 50 NVIDIA Kepler GPU machines using TensorFlow. We use RMSProp with decay 0.9 and $\epsilon = 0.1$, batch size 32, 100 epochs, and a learning rate of 0.0045 that is decayed by 0.94 every two epochs. Gradient clipping with a threshold of 2.0 also helps.

11 Performance on Lower Resolution Input

In object detection, the objects in the image can be small and low resolution. If you just shrink your neural network naively, then it will not do that well. A better approach is to create a neural network that accepts a low resolution input (e.g. 79×79 instead of 299×299) and then has stride 1 (instead of stride 2) and no max-pooling (instead of max-pooling). This is better than just naively shrinking the entire network and training that. So, if you are doing R-CNN, maybe you want to train one of these lower resolution networks.

12 Experimental Results and Comparisons

For single-frame/single-crop evaluation, GoogLeNet gets top-1 (top-5) error of 29% (9.2%). Our Inception-v3 gets 21.2% (4.8%). This also beats ResNet (and is computationally cheaper). The batch normalized auxiliary classifiers and label smoothing do a good job regularizing and giving us gains on the evaluation set.

13 Conclusions

We've described some principles for creating Inception architectures and applied them to make Inception-v3.