Going deeper with convolutions

The main hallmark of this architecture is the improved utilization of the computing resources inside the network. This was achieved by a carefully crafted design that allows for increasing the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing.

Introduction

GoogLeNet uses 12 times fewer parameters than AlexNet while being significantly more accurate. The efficiency of algorithm especially power and memory use were prioritized instead of accuracy numbers. Models computational budget was fixed to keep a budget of 1.5 billion multiply-adds at inference time, so that they do not end up to be a purely academic curiosity but could be put to real world use.

The word "deep" has two meanings in this paper first meaning in the sense that they introduced "Inception module" and second being increased network depth.

Related Work

Network-in-Network is an approach which uses 1X1 convolutions to increase the representational power of the neural networks. 1X1 convolutions are heavily used in GoogLeNet. It has dual purpose : most critically, they are used as dimension reduction modules to remove computational bottlenecks and secondly to introduce more non-linearity.

Motivation and High Level Considerations

The most straightforward way of improving the performance of deep neural networks is by increasing their size. However this simple solution comes with two major drawbacks. Bigger size typically means a larger number of parameters, which makes the enlarged network more prone to overfitting, especially if the number of labeled examples in the training set is limited.

And creation of high quality datasets can be tricky and expensive. Another drawback of uniformly increased network size is the dramatically increased use of computational resources. Since in practice the computational budget is always finite, an efficient distribution of computing resources is preferred to an indiscriminate increase of size, even when the main objective is to increase the quality of results.

The fundamental way of solving both issues would be by ultimately moving from fully connected to sparsely connected architectures, even inside the convolutions. Besides mimicking biological systems, this would also have the advantage of firmer theoretical underpinnings due to the groundbreaking work of Arora et al. [2]. Their main result states that if the probability distribution of the data-set is representable by a large, very sparse deep neural network, then the optimal network topology can be constructed layer by layer by analyzing the correlation statistics of the activations of the last layer and clustering neurons with highly correlated outputs. Although the strict mathematical proof requires very strong conditions, the fact that this statement resonates with the well known Hebbian principle – neurons that fire together, wire together – suggests that the underlying idea is applicable even under less strict conditions, in practice.

On the downside, todays computing infrastructures are very inefficient when it comes to numerical calculation on non-uniform sparse data structures. Even if the number of arithmetic operations is reduced by 100X, the overhead of lookups and cache misses is so dominant that switching to sparse matrices would not pay off.

Research suggests that clustering sparse matrices into relatively dense submatrices tends to give state of the art practical performance for sparse matrix multiplication.

The Inception architecture started out as a case study of the first author for assessing the hypothetical output of a sophisticated network topology construction algorithm that tries to approximate a sparse structure implied by for vision networks and covering the hypothesized outcome by dense, readily available components.

Architectural Details

Arora et al suggests a layer-by layer construction in which one should analyze the correlation statistics of the last layer and cluster them into groups of units with high correlation. These clusters form the units of the next layer and are connected to the units in the previous layer. We assume that each unit from the earlier layer corresponds to some region of the input image and these units are grouped into filter banks. In the lower layers (the ones close to the input) correlated units would concentrate in local regions. This means, we would end up with a lot of clusters concentrated in a single region and they can be covered by a layer of 1X1 convolutions in the next layer, as suggested in . However, one can also expect that there will be a smaller number of more spatially spread out clusters that can be covered by convolutions over larger patches, and there will be a decreasing number of patches over larger and larger regions. In order to avoid patchalignment issues, current incarnations of the Inception architecture are restricted to filter sizes 1X1, 3X3 and 5X5, however this decision was based more on convenience rather than necessity. It also means that the suggested architecture is a combination of all those layers with their output filter banks concatenated into a single output vector forming the input of the next stage. as features of higher abstraction are captured by higher layers, their spatial concentration is expected to decrease suggesting that the ratio of 3×3 and 5×5 convolutions should increase as we move to higher layers. The merging of the output of the pooling layer with the outputs of convolutional layers would lead to an inevitable increase in the number of outputs from stage to stage. Even while this architecture might cover the optimal sparse structure, it would do it very inefficiently, leading to a computational blow up within a few stages. 1X1 convolutions are used to compute reductions before the expensive 3X3 and 5X5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose. Inception network is a network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride 2 to halve the resolution of the grid.

GoogLeNet

All the convolutions, including those inside the Inception modules, use rectified linear activation. The size of the receptive field in our network is 224×224 taking RGB color channels with mean subtraction.

The network is 22 layers deep when counting only layers with parameters (or 27 layers if we also count pooling). The overall number of layers (independent building blocks) used for the construction of the network is about 100. However this number depends on the machine learning infrastructure system used. The use of average pooling before the classifier is based on [12], although our implementation differs in that we use an extra linear layer. This enables adapting and fine-tuning our networks for other label sets easily, but it is mostly convenience and we do not expect it to have a major effect. It was found that a move from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%, however the use of dropout remained essential even after removing the fully connected layers.

Auxiliary classifiers connected to these intermediate layers, we would expect to encourage discrimination in the lower stages in the classifier, increase the gradient signal that gets propagated back, and provide additional regularization. These classifiers take the form of smaller convolutional networks put on top of the output of the Inception (4a) and (4d) modules. During training, their loss gets added to the total loss of the network with a discount weight (the losses of the auxiliary classifiers were weighted by 0.3). At inference time, these auxiliary networks are discarded.

<u>Training Methodology</u>

Our networks were trained using the DistBelief distributed machine learning system using modest amount of model and data-parallelism. Although we used CPU based implementation only, a rough estimate suggests that the GoogLeNet network could be trained to convergence using few high-end GPUs within a week, the main limitation being the memory usage. Our training used asynchronous stochastic gradient descent with 0.9 momentum, fixed learning rate schedule (decreasing the learning rate by 4% every 8 epochs). Polyak averaging was used to create the final model used at inference time.

sampling of various sized patches of the image whose size is distributed evenly between 8% and 100% of the image area and whose aspect ratio is chosen randomly between 3/4 and 4/3.

Also, they found that the photometric distortions by were useful to combat overfitting to some extent. They also used random interpolation methods (bilinear, area, nearest neighbor and cubic, with equal probability) for resizing relatively late.