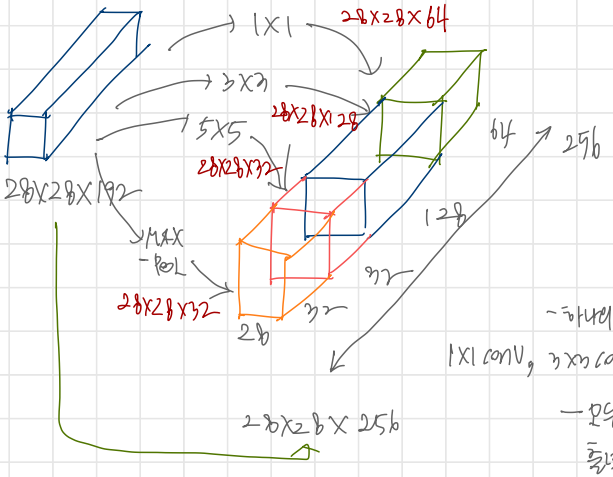
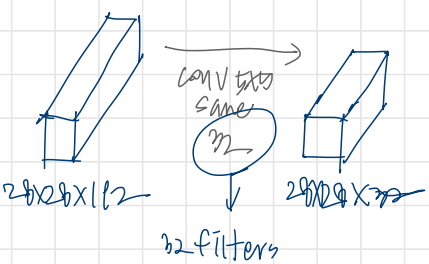
1. Motivation for Inception Network



- 하나의 데이터에 대해서 여러번 작업을 수행하고싶을때에 1X1 conv, 3X3 conv, 5X5 conv, maxpool 전부 한다는 의의이다.
- 모든 output을 stack / concat하여 하나에 합친다.

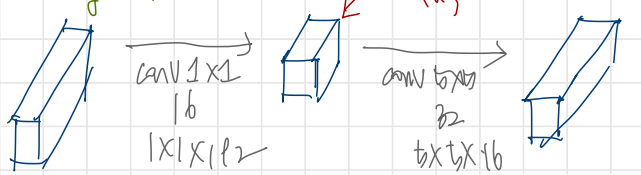
28X28X192X256X28X28 = 120 M.

2. The problem of Convolutional Cost



Using 1X1 convolution

bottleneck layer (=smallest part of the layer)



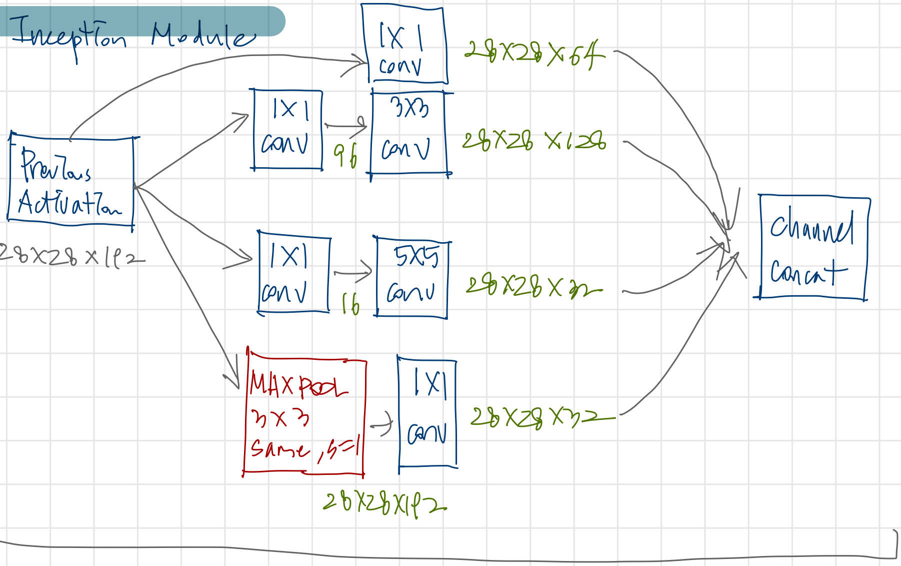
28X28X16X192 = 2.4M

28X28X16X5X5X16 = 10.0M

12.4M

IMPROVED COMPUTATIONAL COSTS

3. Inception Module



하나의 Inception Module

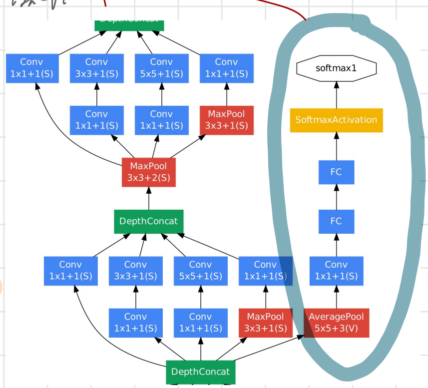
↓ 多

Inception Network

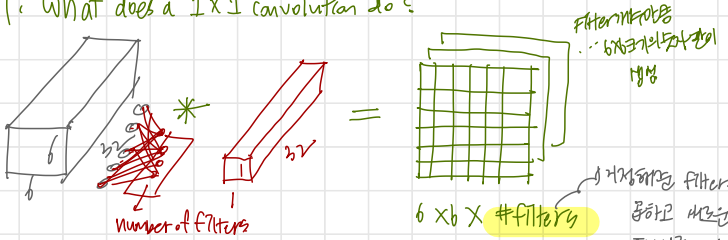
- Inception Network을 전부 살펴보게 되면 마지막 출력층으로 Fully Connected Layer + Softmax가 존재한다. 그런데 특이한 점은 중간에 곁가지로 각각의 Inception Module마다 예측값을 반환하는 FC layer + Softmax가 존재함을 알수있다.

이렇게 중간에 사용한 예측값들 또한 이미지의 결과를 예측하는데에 아주 나쁘지 않다는 것을 확인한다.

→ **REGULARIZATION(정규화)** 의 효과를 줌으로써 overfitting을 방지하는 효과를 가져온다.



Cf. What does a 1×1 convolution do?

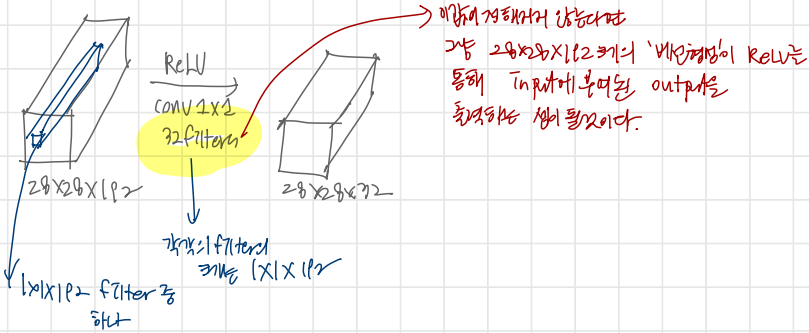


filter개수만큼
∴ 6×6크기의 feature
생성

→ 3개의 위치를 선정하고 그위의 32개의 숫자를 filter의 32개의 숫자와 곱해준다.

가정해줌 filter의개수만큼 곱하고 싶은 6×6크기의 데이터를 생성하도록 하는것이 가능하다.

∴ Fully Connected NN applied to the 32 numbers × number of filters



이렇게 정의해서 만들다면
그럼 28×28×192크기의 '비선형'이 ReLU를 통해 Input에 곱해져 output을 출력하는 셈이 될것이다.

Abstract

We propose a deep convolutional neural network architecture codenamed Inception, which was responsible for setting the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). 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. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

이 논문에서는 "Inception"이라는 새로운 deep convolution NN architecture를 제안한다.
ILSVRC 2014에 사용했던 모델(GoogLeNet)는 이 Inception 구조를 구현했다.

cf. Hebbian Principle = "neurons that fire together, wire together"
= 동시에 활성화된 노드들끼리 연관성이 강해

1 Introduction

In the last three years, mainly due to the advances of deep learning, more concretely convolutional networks [10], the quality of image recognition and object detection has been progressing at a dramatic pace. One encouraging news is that most of this progress is not just the result of more powerful hardware, larger datasets and bigger models, but mainly a consequence of new ideas, algorithms and improved network architectures. No new data sources were used, for example, by the top entries in the ILSVRC 2014 competition besides the classification dataset of the same competition for detection purposes. Our GoogLeNet submission to ILSVRC 2014 actually uses 12× fewer parameters than the winning architecture of Krizhevsky et al [9] from two years ago, while being significantly more accurate. The biggest gains in object-detection have not come from the utilization of deep networks alone or bigger models, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

- 정정 CNN과 딥러닝기술, 특히 detection과 classification 분야에서 기술이 발전하였지, 이는 데이터의 양의 증가에서 기인한 것이 아닌 발전된 네트워크 기술 덕이다.
  특히나 GoogLeNet에 사용한 모델의 경우 12개 정도의 parameter만 사용하여도 좋은 성능을 보인다.
- 또한 모바일과 임베디드 환경에서 작동은 같이 하기에는 전력, 메모리 효율이 중요하여서 "성능향상"이 중요
  ① 네트워크가 깊어짐
  ② Inception Module에서 비용을 고려함

2 Related Work

Starting with LeNet-5 [10], convolutional neural networks (CNN) have typically had a standard structure – stacked convolutional layers (optionally followed by contrast normalization and max-pooling) are followed by one or more fully-connected layers. Variants of this basic design are prevalent in the image classification literature and have yielded the best results to-date on MNIST, CIFAR and most notably on the ImageNet classification challenge [9, 21]. For larger datasets such as Imagenet, the recent trend has been to increase the number of layers [12] and layer size [21, 14], while using dropout [7] to address the problem of overfitting.

Despite concerns that max-pooling layers result in loss of accurate spatial information, the same convolutional network architecture as [9] has also been successfully employed for localization [9, 14], object detection [6, 14, 18, 5] and human pose estimation [19]. Inspired by a neuroscience model of the primate visual cortex, Serre et al. [15] use a series of fixed Gabor filters of different sizes in order to handle multiple scales, similarly to the Inception model. However, contrary to the fixed 2-layer deep model of [15], all filters in the Inception model are learned. Furthermore, Inception layers are repeated many times, leading to a 22-layer deep model in the case of the GoogLeNet model.

Network-in-Network is an approach proposed by Lin et al. [12] in order to increase the representational power of neural networks. When applied to convolutional layers, the method could be viewed as additional 1×1 convolutional layers followed typically by the rectified linear activation [9]. This enables it to be easily integrated in the current CNN pipelines. We use this approach heavily in our architecture. However, in our setting, 1×1 convolutions have dual purpose: most critically, they are used mainly as dimension reduction modules to remove computational bottlenecks, that would otherwise limit the size of our networks. This allows for not just increasing the depth, but also the width of our networks without significant performance penalty.

The current leading approach for object detection is the Regions with Convolutional Neural Networks (R-CNN) proposed by Girshick et al. [6]. R-CNN decomposes the overall detection problem into two subproblems: to first utilize low-level cues such as color and superpixel consistency for potential object proposals in a category-agnostic fashion, and to then use CNN classifiers to identify object categories at those locations. Such a two stage approach leverages the accuracy of bounding box segmentation with low-level cues, as well as the highly powerful classification power of state-of-the-art CNNs. We adopted a similar pipeline in our detection submissions, but have explored enhancements in both stages, such as multi-box [5] prediction for higher object bounding box recall, and ensemble approaches for better categorization of bounding box proposals.

LeNet

AlexNet

Serre 연구 — Inception과 비슷하게 다양한 크기의 filter을 사용하지만, Inception과 달리 고정된 filter 사용 (Inception은 22-layer 깊이의 모델을 이용하면서도 모든 filter이 학습을 한다)

NIN

R-CNN — 전체적인 detection 문제를 2개로 나눔, low-level feature을 사용해 category에 구애받지 않고
① color/texture 같은 object local proposal 생성
② CNN 분류기를 사용하여 정보 특정 분류

① LeNet-5 + AlexNet

Stacked Conv Layer
+
Contrast Normalization / Max-Pooling Layer
+
Fully Connected Layer

- 당시 ImageNet과 같은 큰 dataset을 다룰 경우, Layer의 수와 크기를 늘리면서 dropout을 적용해서 과적합을 막았으며 Max-pooling layer 적용함에 accurate spatial info를 잃을 수 있다는 우려에도 불구하고 AlexNet은 높은 성과를 보인다.

② NIN

- 네트워크의 표현력을 위한 접근법
- 1X1 Conv layer을 네트워크에 추가하여 깊이를 증가시킨다.
→ 사용 이유: ① computational bottleneck을 제거하기 위해서 차원축소에 사용
② 큰 성능의 저하 없이 depth와 width를 모두 늘리기 위하여

③ R-CNN

① Multi-Box Prediction → bounding box의 회수율을 높이기 위해서 selective search 알고리즘을 사용하는데, 이는 exhaustive search와 segmentation의 장점을 결합한 것이다.

② Ensemble Approach → 실제로 True인 데이터를 True로 예측한 비율의 정확도를 높이기 위해서 동일한 input에 대해서 multiple instance를 detect함

3 Motivation and High Level Considerations

The most straightforward way of improving the performance of deep neural networks is by increasing their size. This includes both increasing the depth – the number of levels – of the network and its width: the number of units at each level. This is as an easy and safe way of training higher quality models, especially given the availability of a large amount of labeled training data. 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. This can become a major bottleneck, since the creation of high quality training sets can be tricky and expensive, especially if expert human raters are necessary to distinguish between fine-grained visual categories like those in ImageNet (even in the 1000-class ILSVRC subset) as demonstrated by Figure 1.

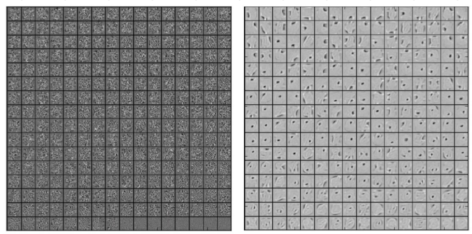
Another drawback of uniformly increased network size is the dramatically increased use of computational resources. For example, in a deep vision network, if two convolutional layers are chained, any uniform increase in the number of their filters results in a quadratic increase of computation. If the added capacity is used inefficiently (for example, if most weights end up to be close to zero), then a lot of computation is wasted. 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 $100\times$, the overhead of lookups and cache misses is so dominant that switching to sparse matrices would not pay off. The gap is widened even further by the use of steadily improving, highly tuned, numerical libraries that allow for extremely fast dense matrix multiplication, exploiting the minute details of the underlying CPU or GPU hardware [16, 9]. Also, non-uniform sparse models require more sophisticated engineering and computing infrastructure. Most current vision oriented machine learning systems utilize sparsity in the spatial domain just by the virtue of employing convolutions. However, convolutions are implemented as collections of dense connections to the patches in the earlier layer. ConvNets have traditionally used random and sparse connection tables in the feature dimensions since [11] in order to break the symmetry and improve learning, the trend changed back to full connections with [9] in order to better optimize parallel computing. The uniformity of the structure and a large number of filters and greater batch size allow for utilizing efficient dense computation.

This raises the question whether there is any hope for a next, intermediate step: an architecture that makes use of the extra sparsity, even at filter level, as suggested by the theory, but exploits our current hardware by utilizing computations on dense matrices. The vast literature on sparse matrix computations (e.g. [3]) suggests that clustering sparse matrices into relatively dense submatrices tends to give state of the art practical performance for sparse matrix multiplication. It does not seem far-fetched to think that similar methods would be utilized for the automated construction of non-uniform deep-learning architectures in the near future.

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 [2] for vision networks and covering the hypothesized outcome by dense, readily available components. Despite being a highly speculative undertaking, only after two iterations on the exact choice of topology, we could already see modest gains against the reference architecture based on [12]. After further tuning of learning rate, hyperparameters and improved training methodology, we established that the resulting Inception architecture was especially useful in the context of localization and object detection as the base network for [6] and [5]. Interestingly, while most of the original architectural choices have been questioned and tested thoroughly, they turned out to be at least locally optimal.

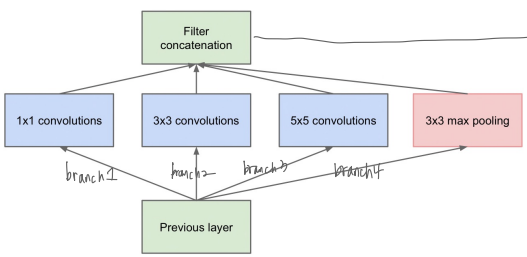


(a) Without dropout (b) Dropout with $p = 0.5$.

4 Architectural Details

The main idea of the Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated and covered by readily available dense components. Note that assuming translation invariance means that our network will be built from convolutional building blocks. All we need is to find the optimal local construction and to repeat it spatially. Arora et al. [2] 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 1×1 convolutions in the next layer, as suggested in [12]. 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 patch-alignment issues, current incarnations of the Inception architecture are restricted to filter sizes 1×1, 3×3 and 5×5, 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. Additionally, since pooling operations have been essential for the success in current state of the art convolutional networks, it suggests that adding an alternative parallel pooling path in each such stage should have additional beneficial effect, too (see Figure 2(a)).

As these "Inception modules" are stacked on top of each other, their output correlation statistics are bound to vary: 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.



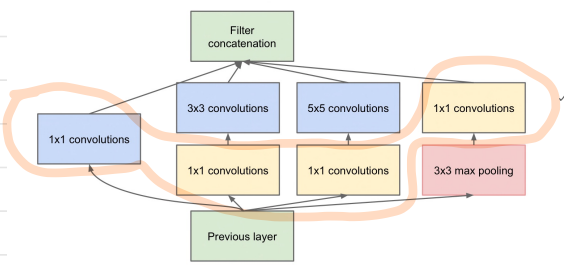
(a) Inception module, naïve version

This leads to the second idea of the proposed architecture: judiciously applying dimension reductions and projections wherever the computational requirements would increase too much otherwise. This is based on the success of embeddings: even low dimensional embeddings might contain a lot of information about a relatively large image patch. However, embeddings represent information in a dense, compressed form and compressed information is harder to model. We would like to keep our representation sparse at most places (as required by the conditions of [2]) and compress the signals only whenever they have to be aggregated en masse. That is, 1×1 convolutions are used to compute reductions before the expensive 3×3 and 5×5 convolutions. Besides being used as reductions, they also include the use of rectified linear activation which makes them dual-purpose. The final result is depicted in Figure 2(b).

In general, an 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. For technical reasons (memory efficiency during training), it seemed beneficial to start using Inception modules only at higher layers while keeping the lower layers in traditional convolutional fashion. This is not strictly necessary, simply reflecting some infrastructural inefficiencies in our current implementation.

One of the main beneficial aspects of this architecture is that it allows for increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity. The ubiquitous use of dimension reduction allows for shielding the large number of input filters of the last stage to the next layer, first reducing their dimension before convolving over them with a large patch size. Another practically useful aspect of this design is that it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously.

The improved use of computational resources allows for increasing both the width of each stage as well as the number of stages without getting into computational difficulties. Another way to utilize the inception architecture is to create slightly inferior, but computationally cheaper versions of it. We have found that all the included the knobs and levers allow for a controlled balancing of computational resources that can result in networks that are $2-3\times$ faster than similarly performing networks with non-Inception architecture, however this requires careful manual design at this point.



(b) Inception module with dimension reductions

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

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. "#3×3 reduce" and "#5×5 reduce" stands for the number of 1×1 filters in the reduction layer used before the 3×3 and 5×5 convolutions. One can see the number of 1×1 filters in the projection layer after the built-in max-pooling in the pool proj column. All these reduction/projection layers use rectified linear activation as well.

→ ReLU activation function을 사용한다. (모든 층)

The network was designed with computational efficiency and practicality in mind, so that inference can be run on individual devices including even those with limited computational resources, especially with low-memory footprint. 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.

layer 세는 방법에 따라 22layer, 또는 27layer 이라고 결정 지을 수 있다.

깊이 있는 네트워크에 대해서 모든 layer를 통해 gradient는 효과적으로 전달할 수 있어야 한다.

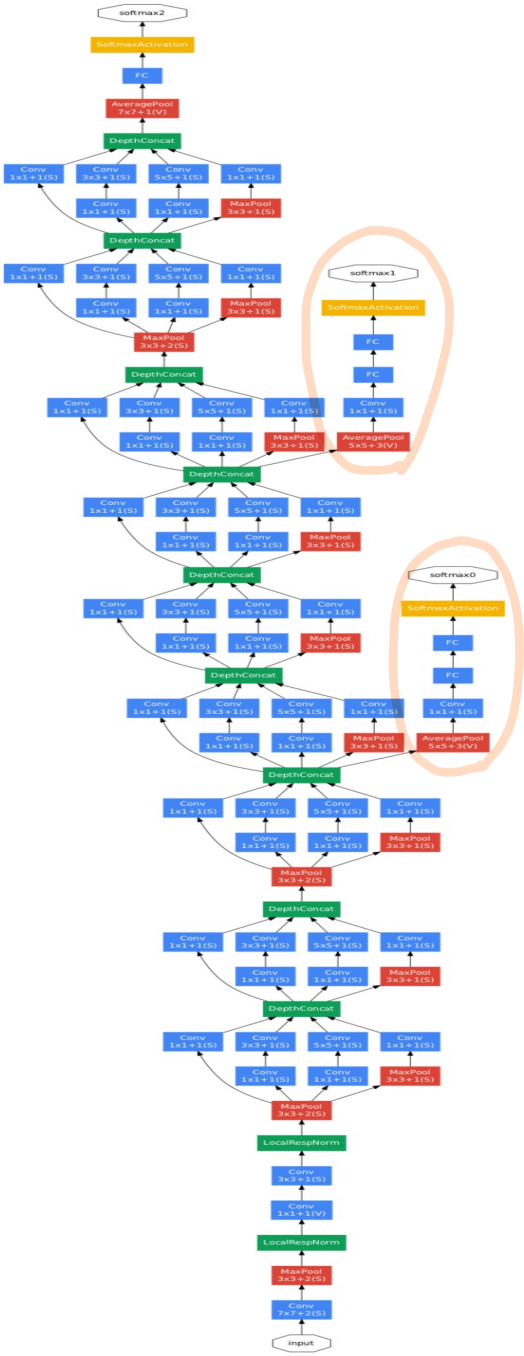
보조 분류기의 추가를 통해서 하위 layer에서 분류 식별성과 regulation 효과, gradient 손실 문제 완화 효과를 기대한다.

Given the relatively large depth of the network, the ability to propagate gradients back through all the layers in an effective manner was a concern. One interesting insight is that the strong performance of relatively shallower networks on this task suggests that the features produced by the layers in the middle of the network should be very discriminative. By adding 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.

→ 보조 분류기 loss는 30%만 고려.

실제로 성능 향상에는 0.5%만 기여.

Network의 중간 layer의 feature 식별이 중요함은 shallow Model의 좋은 성능을 통해 확인 가능하다.



Auxiliary Network

- An average pooling layer with 5×5 filter size and stride 3, resulting in an $4\times4\times512$ output for the (4a), and $4\times4\times528$ for the (4d) stage.
- A 1×1 convolution with 128 filters for dimension reduction and rectified linear activation.
- A fully connected layer with 1024 units and rectified linear activation.
- A dropout layer with 70% ratio of dropped outputs.
- A linear layer with softmax loss as the classifier (predicting the same 1000 classes as the main classifier, but removed at inference time).

6 Training Methodology

Our networks were trained using the DistBelief [4] 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 [17], fixed learning rate schedule (decreasing the learning rate by 4% every 8 epochs). Polyak averaging [13] was used to create the final model used at inference time.

Our image sampling methods have changed substantially over the months leading to the competition, and already converged models were trained on with other options, sometimes in conjunction with changed hyperparameters, like dropout and learning rate, so it is hard to give a definitive guidance to the most effective single way to train these networks. To complicate matters further, some of the models were mainly trained on smaller relative crops, others on larger ones, inspired by [8]. Still, one prescription that was verified to work very well after the competition includes 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, we found that the photometric distortions by Andrew Howard [8] were useful to combat overfitting to some extent. In addition, we started to use random interpolation methods (bilinear, area, nearest neighbor and cubic, with equal probability) for resizing relatively late and in conjunction with other hyperparameter changes, so we could not tell definitely whether the final results were affected positively by their use.

- 0.9 momentum을 가진 asynchronous SGD 사용
- Fixed Learning Rate Schedule (8번째마다 learning rate을 4% 감소하도록 (α=0.96))
- Polyak Averaging
  - instead of using the actual parameter vector, keep a moving average of the parameter vector and use that at test time.
- 이미지 데이터는 Random Interpolation Method을 사용해서 변경하였다. (=bilinear, area, nearest neighbor, cubic)
- 8%-100% 크기에서 균등분포로 patch sampling을 하는 것이 좋은 결과
- photometric distortions이 과적합방지에 유용

9 Conclusions

Our results seem to yield a solid evidence that approximating the expected optimal sparse structure by readily available dense building blocks is a viable method for improving neural networks for computer vision. The main advantage of this method is a significant quality gain at a modest increase of computational requirements compared to shallower and less wide networks. Also note that our detection work was competitive despite of neither utilizing context nor performing bounding box

regression and this fact provides further evidence of the strength of the Inception architecture. Although it is expected that similar quality of result can be achieved by much more expensive networks of similar depth and width, our approach yields solid evidence that moving to sparser architectures is feasible and useful idea in general. This suggest promising future work towards creating sparser and more refined structures in automated ways on the basis of [2].

- 현재 이용이 가능한 기존의 convolution layer을 이용해서 optimal sparse structure로 근사했다, 실제로 성능향상에 도움을 줬다.

- dense 하게 배치 shallow 한 net 보다 computational cost 의 증가도 적고 상당한 성능의 향상을 얻을 수 있다.

- object detection에서 context의 활용이나 bounding box regression 없이도 경쟁력 있었으며, 이는 Inception Architecture의 강점 또한 증명하였다.