K_03 Understanding of Inception - EN

The Inception network was an important milestone in the development of CNN classifiers. Prior to its inception (pun intended), most popular CNNs just stacked convolution layers deeper and deeper, hoping to get better performance.

Designing CNNs in a nutshell. Fun fact, this meme was referenced in the first inception net paper.

The Inception network on the other hand, was complex (heavily engineered). It used a lot of tricks to push performance; both in terms of speed and accuracy. Its constant evolution lead to the creation of several versions of the network. The popular versions are as follows:

- Inception v1.
- Inception v2 and Inception v3.
- Inception v4 and Inception-ResNet.

Each version is an iterative improvement over the previous one. Understanding the upgrades can help us to build custom classifiers that are optimized both in speed and accuracy.

This article aims to elucidate the evolution of the inception network.

1. Motivation

Increasing the depth (number of layers) is not the only way to make a model bigger. What about increasing both the depth and width of the network while keeping computations to a constant level?

This time the inspiration comes from the human visual system, wherein information is processed at multiple scales and then aggregated locally. How to achieve this without a memory explosion?

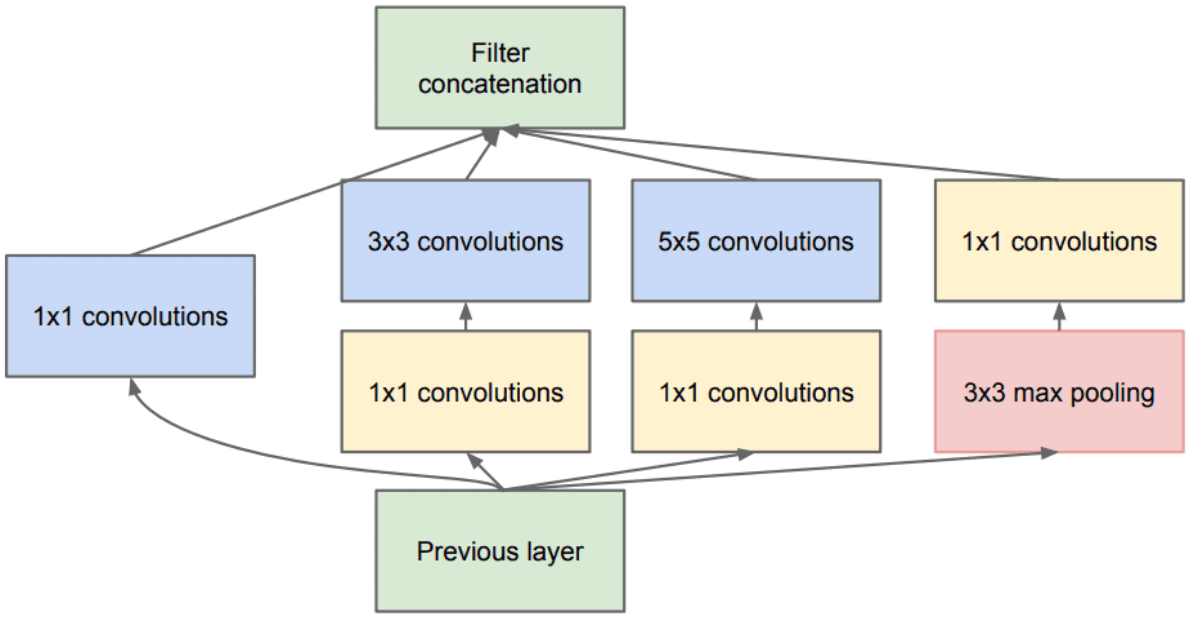
The answer is with 1×1 convolutions! The main purpose is dimension reduction, by reducing the output channels of each convolution block. Then we can process the input with different kernel sizes. As long as the output is padded, it is the same as in the input.

To find the appropriate padding with single stride convs without dilation, padding p and kernel k are defined so that $out = in$ (input and output spatial dims):

$out = in + 2\times p - k + 1$, which means that $p = (k-1)/2$. In Keras you simply specify padding='same'. This way, we can concatenate features convolved with different kernels.

Then we need the 1×1 convolutional layer to 'project' the features to fewer channels in order to win computational power. And with these extra resources, we can add more layers. Actually, the 1×1 convs work similar to a low dimensional embedding.

This in turn allows to not only increase the depth, but also the width of the famous GoogleNet by using Inception modules. The core building block, called the inception module, looks like this:



The whole architecture is called GoogLeNet or InceptionNet. In essence, the authors claim that they try to approximate a sparse convnet with normal dense layers (as shown in the figure).

Why? Because they believe that only a small number of neurons are effective. This comes in line with the Hebbian principle: “Neurons that fire together, wire together”.

Moreover, it uses convolutions of different kernel sizes (5×5, 3×3 ,1×1) to capture details at multiple scales.

In general, a larger kernel is preferred for information that resides globally, and a smaller kernel is preferred for information that is distributed locally.

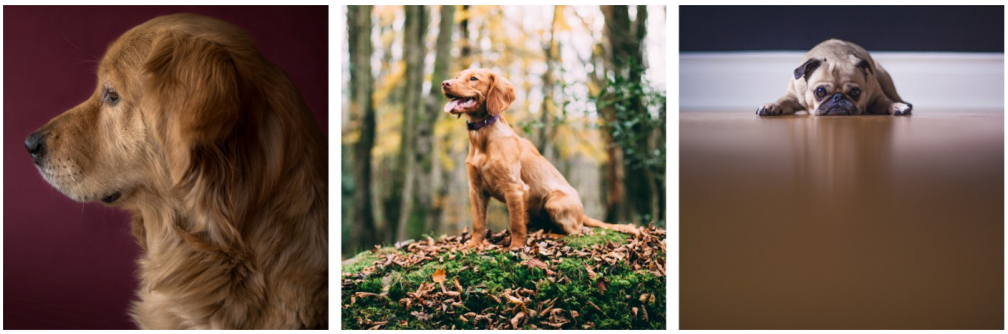
Besides, 1×1 *convolutions are used to compute reductions before the computationally expensive convolutions (*3×3 *and* 5×5*).*

The InceptionNet/GoogLeNet architecture consists of 9 inception modules stacked together, with max-pooling layers between (to halve the spatial dimensions). It consists of 22 layers (27 with the pooling layers). It uses global average pooling after the last inception module.

2. Inception v1

This is where it all started. Let us analyze what problem it was purported to solve, and how it solved it. (Paper (https://arxiv.org/pdf/1409.4842v1.pdf))

2.1 The Premise:



From left: A dog occupying most of the image, a dog occupying a part of it, and a dog occupying very little space.

- Salient parts in the image can have extremely large variation in size. For instance, an image with a dog can be either of the following, as shown below. The area occupied by the dog is different in each image.

- Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes tough. A larger kernel is preferred for information that is distributed more globally, and a smaller kernel is preferred for information that is distributed more locally.
- Very deep networks are prone to overfitting. It also hard to pass gradient updates through the entire network.
- Naively stacking large convolution operations is computationally expensive.

The most direct way to improve the performance of deep neural networks is to increase their size . This includes increasing the depth (number of network layers) and its width (the number of units in each layer is the number of channels in the convolutional network). This is an easy and safe way to train higher-quality models, especially when a large amount of labeled training data is available. But this simple program has two major drawbacks:

- Greater size usually means more arguments, which will make it easier to increase network overfitting , especially in the limited circumstances stated in the catalog of the training set (This is a major bottleneck because it is time-consuming, laborious and expensive to obtain a strongly labeled data set).
- A uniform increase in network size leads to a significant increase in the use of computing resources .

2.2 The Solution:

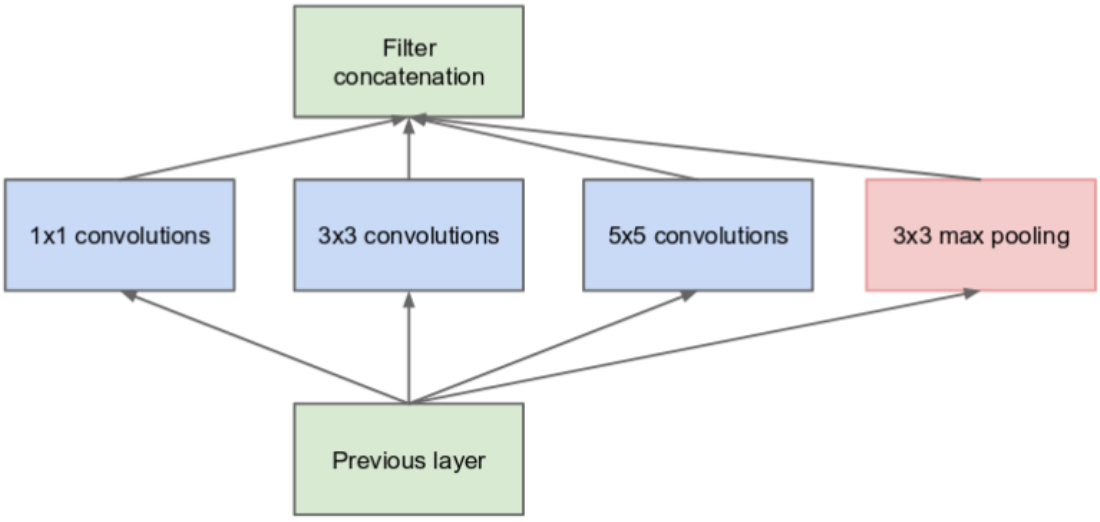
2.2.1. Introduce sparsity

Why not have filters with multiple sizes operate on the same level? The network essentially would get a bit “wider” rather than “deeper”. The authors designed the inception module to reflect the same. The below image is the “naive” inception module. It performs convolution on an input, with 3 different sizes of filters $(1 \times 1, 3 \times 3, 5 \times 5)$. Additionally, max pooling is also performed. The outputs are concatenated and sent to the next inception module.

Obviously, by using a sparse hierarchical structure instead of using a fully cascaded structure like FC/Conv, it helps to reduce computational overhead. Unfortunately, the current computing architecture is very inefficient when performing numerical calculations on sparse data structures. So what should we do? The answer is to build a dense block structure to approximate the optimal sparse structure, that is, use a sparse module composed of dense computing substructures to extract and express features.

The main idea of the Inception architecture is to approximately design a locally sparse structure in the computer vision convolutional network, and this structure uses the existing dense components.

The structure uses four branches, each of which is composed of 1x1 convolution, 3x3 convolution, 5x5 convolution, and 3x3max pooling, which not only increases the width of the network, but also increases the applicability of the network to different scales. After the four branches are output, they are superimposed on the channel dimension and used as the input of the next layer. The size of the feature map output by the four branches can be controlled by the padding size to ensure that their feature dimensions are the same (regardless of the number of channels). Why can this structure achieve sparsity? Personal understanding is that four parallel convolution kernels are used as dense modules, and this dense block combination is equivalent to sparse modules in effect. **Essentially, the highly correlated features are brought together by convolution and re-aggregation on multiple scales, and sparsity is used in the feature dimension**. The substructure of the design is shown in the figure below:



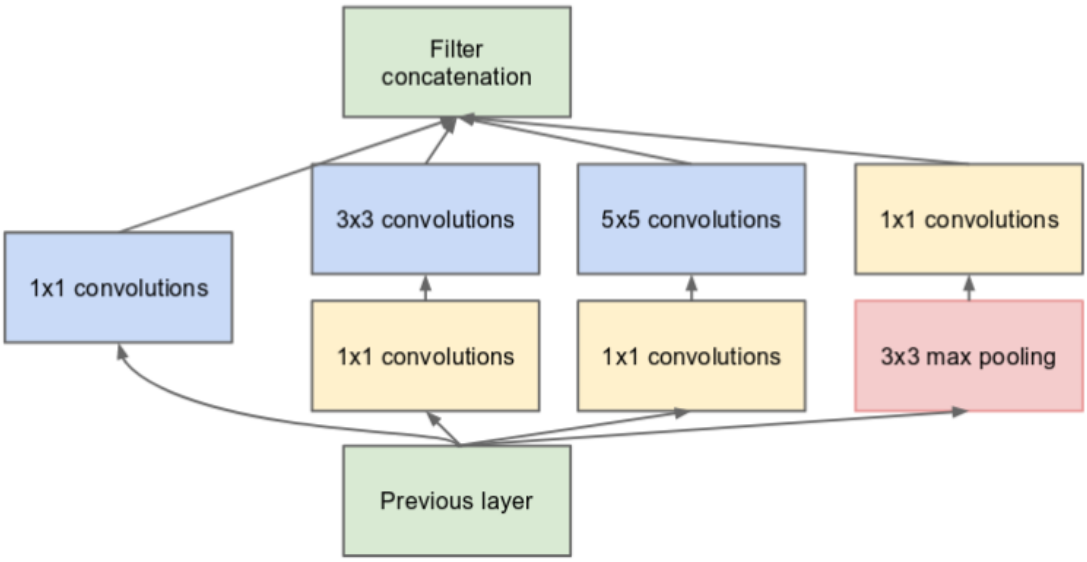
(a) Inception module, naïve version

The naive inception module.

2.2.2. Further dimensionality reduction

As stated before, deep neural networks are computationally expensive. To make it cheaper, the authors limit the number of input channels by adding an extra 1×1 convolution before the 3×3 and 5×5 convolutions. Though adding an extra operation may seem counterintuitive, 1×1 convolutions are far more cheaper than 5×5 convolutions, and the reduced number of input channels also help. Do note that however, the 1×1 convolution is introduced after the max pooling layer, rather than before.

All the convolution kernels in the next layer are built on the combined results of all the outputs of the previous layer, resulting in a large feature map thickness, and the amount of calculation required for the 5×5 convolution kernel is too large (5 The calculation cost of the ×5 convolution kernel itself is relatively high, and the proportion of the 5×5 convolution kernel will increase as the number of layers deepens). This led to the second idea of the Inception architecture: where computing requirements would increase too much, wisely reduce the dimensions. In other words, before the expensive 3×3 and 5×5 convolutions, 1×1 convolution is used to calculate dimensionality reduction (decomposition of convolution in the dimension of the depth of the feature map). In addition to dimensionality reduction , more activation functions are also used to improve nonlinearity and deepen the number of network layers . Most of the benefits of the GoogLeNet network are derived from the extensive use of dimensionality reduction (original text in Inception-v2) . The final result is shown in the figure below.



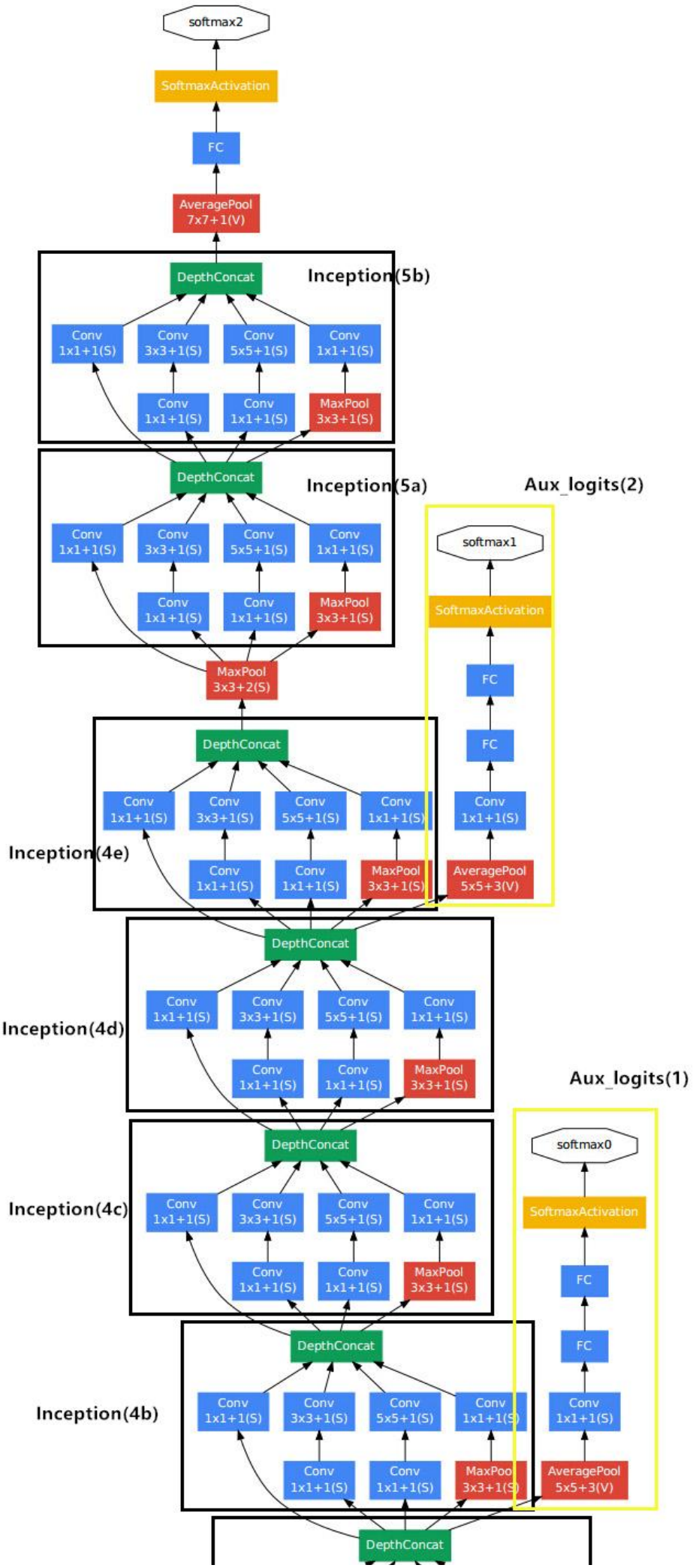
(b) Inception module with dimension reductions

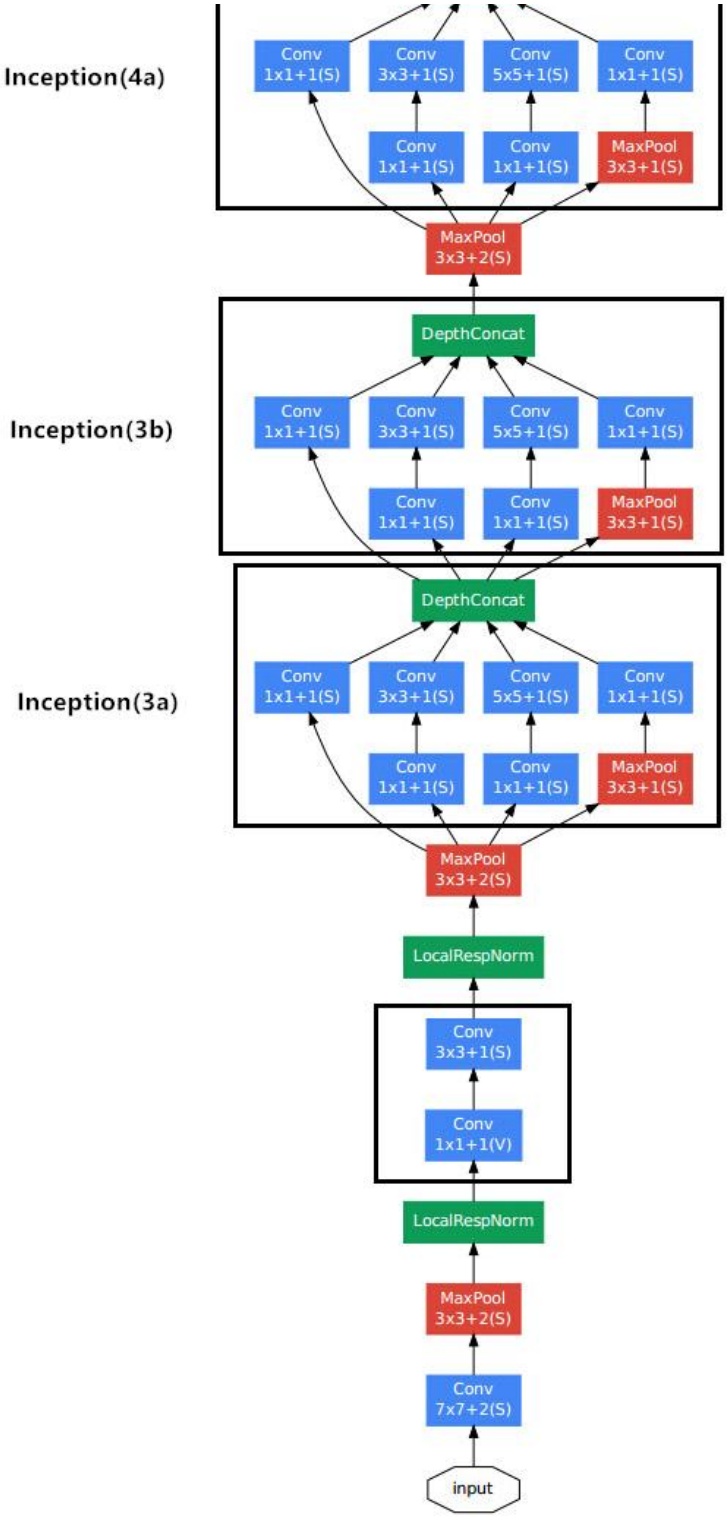
Inception module with dimension reduction.

2.2.3. Tips for the overall structure

- Due to technical reasons (memory efficiency during training), it is beneficial to only start using the Inception module at the higher layer while maintaining the traditional convolution form at the lower layer, and the GoogLeNet structure does exactly that.
- In order to avoid the problem of gradient disappearance, GoogLeNet cleverly adds two losses (adding auxiliary classifiers) at different depths to ensure the smooth return of the gradient.

Using the dimension reduced inception module, a neural network architecture was built. This was popularly known as GoogLeNet (Inception v1). The architecture is shown below:


softmax2
SoftmaxActivation
FC
AveragePool 7x7+1(V)
DepthConcat
Inception(5b)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Inception(5a)
Aux_logits(2)
softmax1
SoftmaxActivation
FC
FC
Conv 1x1+1(S)
AveragePool 5x5+3(V)
MaxPool 3x3+2(S)
DepthConcat
Inception(4e)
DepthConcat
Inception(4d)
Aux_logits(1)
softmax0
SoftmaxActivation
FC
FC
DepthConcat
Inception(4c)
DepthConcat
Inception(4b)
DepthConcat



GoogLeNet. The orange box is the stem, which has some preliminary convolutions. The purple boxes are auxiliary classifiers. The wide parts are the inception modules.

GoogLeNet has 9 such inception modules stacked linearly. It is 22 layers deep (27, including the pooling layers). It uses global average pooling at the end of the last inception module.

Needless to say, it is a pretty deep classifier. As with any very deep network, it is subject to the vanishing gradient problem.

To prevent the middle part of the network from “dying out”, the authors introduced two auxiliary classifiers (The purple boxes in the image). They essentially applied softmax to the outputs of two of the inception modules, and computed an auxiliary loss over the same labels. The total loss function is a weighted sum of the auxiliary loss and the real loss. Weight value used in the paper was 0.3 for each auxiliary loss.

```
# The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2
```

Needless to say, auxiliary loss is purely used for training purposes, and is ignored during inference.

2.3 Effects and advantages

- Fewer parameters save computing resources. (The parameter amount of GoogLeNet is only 1/12 of Alexnet, while the parameter amount of VGGNet is 3 times that of Alexnet)
- The deeper the number of layers increases the network performance and accuracy.
- Visual information is processed and then aggregated on different scales, so that the next stage can simultaneously abstract features from different scales.
- To some extent, the problem of gradient disappearance is alleviated.

3. Inception-v2, v3

Inception v2 and Inception v3 were presented in the same paper. The authors proposed a number of upgrades which increased the accuracy and reduced the computational complexity. Inception v2 explores the following:

3.1 The Premise

- Reduce representational bottleneck. The intuition was that, neural networks perform better when convolutions didn’t alter the dimensions of the input drastically. Reducing the dimensions too much may cause loss of information, known as a “representational bottleneck”
- Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.

The complexity of the Inception architecture makes it difficult to improve the network. Inception-v1 does not provide a clear description of the contributing factors that led to the various design decisions of the GoogLeNet architecture. This makes it more difficult to adapt to new use cases while maintaining its efficiency. For example, if it is deemed necessary to increase the capabilities of some Inception models, a simple transformation that doubles the number of filter bank sizes will result in a four-fold increase in computational cost and the number of parameters. This may prove to be undesirable in many practical situations. Generally speaking, although inception-v1 has achieved good results, it is difficult to improve due to the complexity of the structure. Therefore, some general design principles of inception are provided here (based on the design principles of large-scale experiments, inception is in In experiments with large-scale data, the advantages can be better utilized) to optimize inception-v1.

3.2 Principles of improvement

3.2.1. Avoid characterization bottlenecks

To avoid extreme compression of information content, it should be slowly reduced (compressed) from input to output. Theoretically, the information content cannot be evaluated only by the dimension of the feature, because it discards important factors such as the relevant structure, but the dimension can provide a rough estimate of the information content. So roughly speaking, the size of the feature map should be gradually and slowly reduced, to avoid sudden reduction, and not to reduce prematurely.

3.2.2. Enhanced high-dimensional representation

Higher-dimensional representations are easier to handle locally in the network. Increasing the activation of each block in the convolutional network allows better decoupling of features. The resulting network will train faster. (In the high-level Inception structure, 3×3 is split into 1×3 and 3×1 convolution kernels in parallel, becoming wider and more activation functions to enhance information representation)

3.2.3. Spatial aggregation in low dimensions

At low-dimensional feature maps, you can use a 1×1 convolution kernel to achieve dimensionality reduction before convolution (the author believes that the information in the same channel at low-dimensional features is highly correlated, so dimensionality reduction at the channel level is not too Many adverse effects, but it can speed up training).

3.2.4. Balance width and depth

Increasing the width and depth of the network is helpful to improve the accuracy of the network, but if you increase the width and depth at the same time, the amount of calculation will increase. Therefore, with limited computing resources, set the width and depth of the network in a balanced state In order to achieve better performance.

3.3 The Solution

3.3.1. Decompose to a smaller convolution

Convolutions with larger spatial filters (for example, 5×5 or 7×7) will increase quadratically in terms of computation (5×5 convolutions with n filters are in a grid with m filters). The calculation amount is $25/9 = 2.78$ times higher than that of 3×3 convolution with the same number of filters. At the same time, a larger spatial filter can capture the dependencies between further unit activations and signals (a wider field of view), so if the geometric size of the filter is directly reduced, the expressiveness will be lost.

How to balance the two factors of calculation and field of view? The answer is to replace the large convolution kernel by cascading small convolution kernels. For example, using a two-layer 3×3 convolution can get the same field of view as a 5×5 convolution, as shown in the figure below, but the amount of calculation only doubles, no longer a square increase $(25/(25 - 9\times2) = 0.28$, resulting in a relative gain of 28%). At the same time, it deepens the depth and nonlinearity of the network.

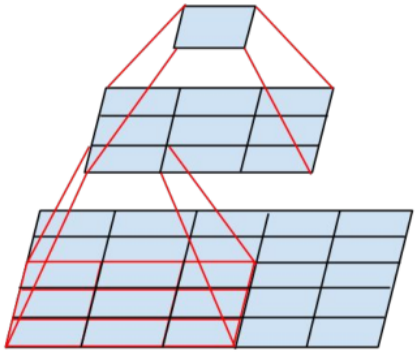


Figure 1. Mini-network replacing the 5×5 convolutions.

applied to the inception structure, there are the changes in the following two figures.

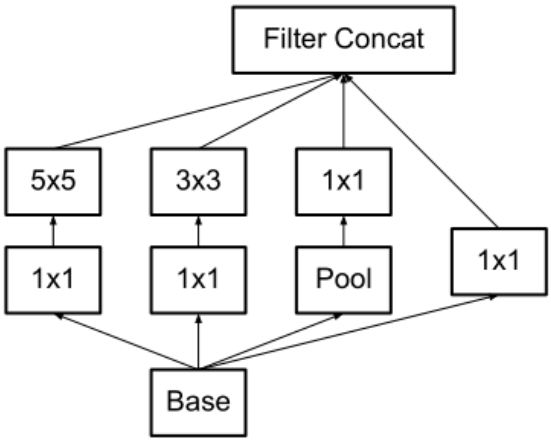


Figure 4. Original Inception module as described in [20].

- Factorize 5×5 convolution to two 3×3 convolution operations to improve computational speed. Although this may seem counterintuitive, a 5×5 convolution is 2.78 times more expensive than a 3×3 convolution. So stacking two 3×3 convolutions infact leads to a boost in performance. This is illustrated in the below image.

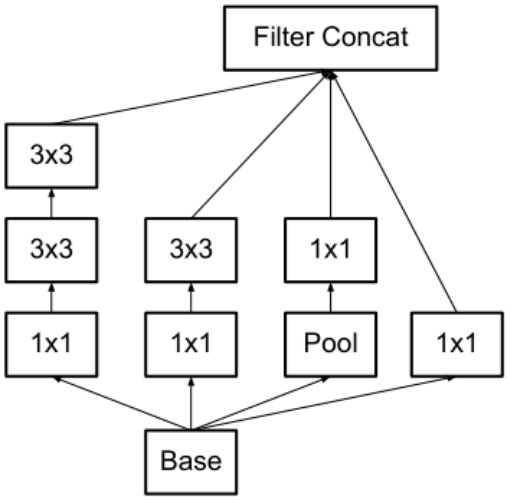
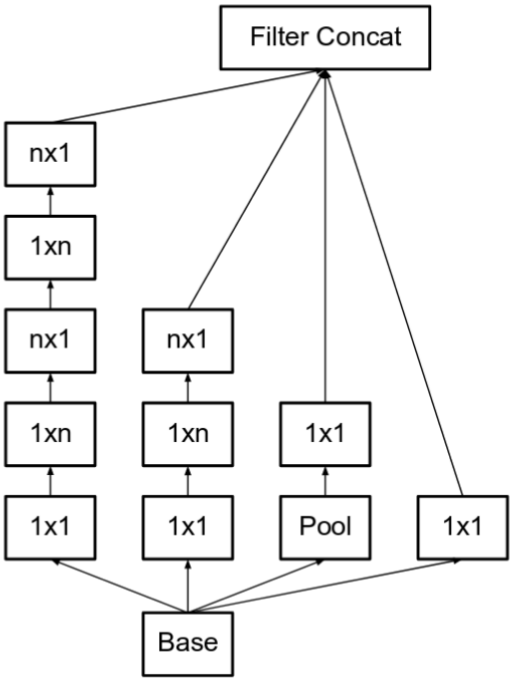


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

The left-most 5×5 convolution of the old inception module, is now represented as two 3×3 convolutions. (Source: Incpetion v2 (https://arxiv.org/pdf/1512.00567v3.pdf))

3.3.2. Decompose into asymmetric convolution

- Moreover, they factorize convolutions of filter size $n \times n$ to a combination of $1 \times n$ and $n \times 1$ convolutions. For example, a 3×3 convolution is equivalent to first performing a 1×3 convolution, and then performing a 3×1 convolution on its output. They found this method to be 33% more cheaper than the single 3×3 convolution. This is illustrated in the below image.



Here, put $n = 3$ to obtain the equivalent of the previous image. The left-most 5×5 convolution can be represented as two 3×3 convolutions, which inturn are represented as 1×3 and 3×1 in series. (Source: Incpetion v2)

The above results indicate that convolution filters larger than 3×3 may generally be useless, because they can always be reduced to a sequence of 3×3 convolutional layers. We can still ask the question whether they should be broken down into smaller ones, such as 2×2 convolutions. However, by decomposing into asymmetric convolution ($n\times1$ followed by $1\times n$), a better effect can be achieved than 2×2. For example, using a 3×1 convolution followed by a 1×3 convolution has the same expression effect as a 3×3 convolution, as shown in the figure below. The asymmetric convolution scheme saves 33% of the calculation, and the decomposition into two 2×2 convolutions means that only 11% is saved.

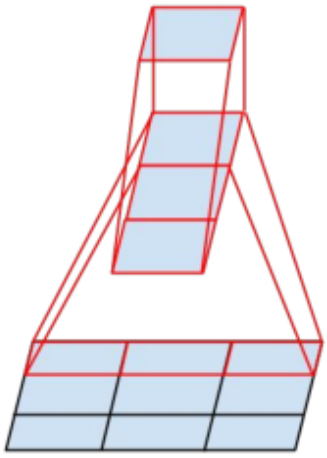


Figure 3. Mini-network replacing the 3×3 convolutions. The lower layer of this network consists of a 3×1 convolution with 3 output units.

can be further extended to the case of $n\times n$ convolution, and the $n\times n$ convolution factor is decomposed into $1\times n$ and $n\times 1$. However, experiments have shown that this method seems to perform poorly in the front part of the network (the layer with a larger feature map size), but it can work well in the layer with a medium-sized (12 ~ 20) feature map (this shows that The inception structure pays great attention to the study of the size of the feature map, and there are also inception blocks corresponding to different structures for different sizes of feature maps). Therefore, the inception structure is further improved as shown in the figure below.

- The filter banks in the module were expanded (made wider instead of deeper) to remove the representational bottleneck. If the module was made deeper instead, there would be excessive reduction in dimensions, and hence loss of information. This is illustrated in the below image.

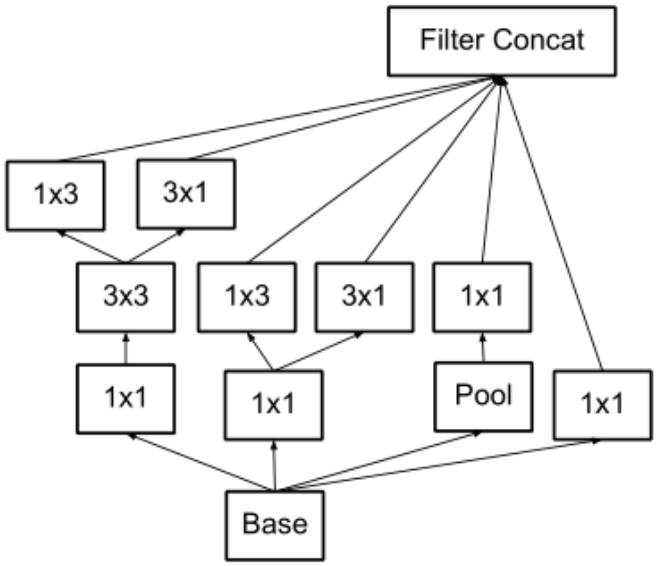


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing high dimensional sparse representation is the most critical as the ratio of local processing (by 1×1 convolutions) is increased compared to the spatial aggregation.

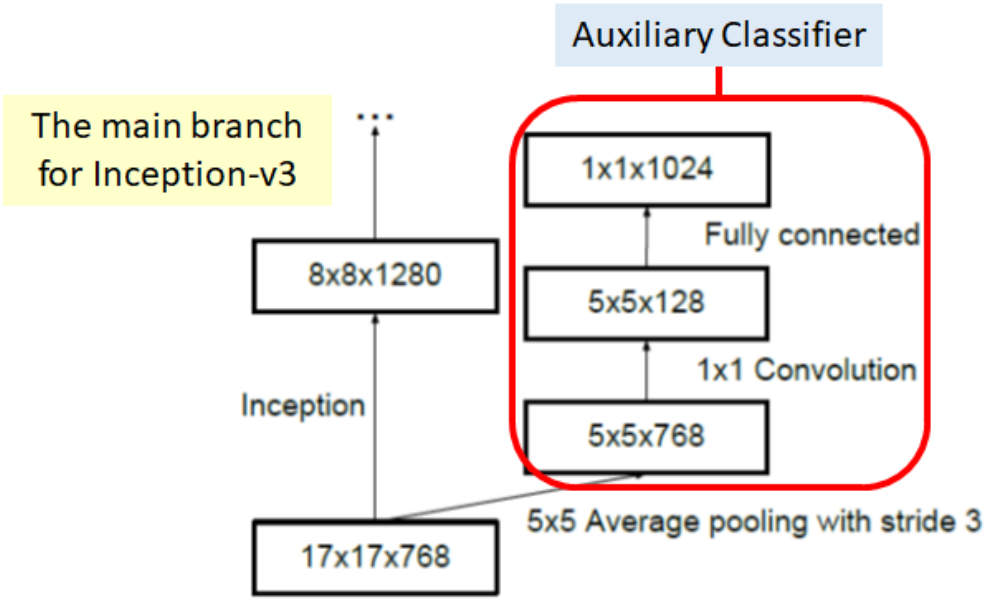
Making the inception module wider. This type is equivalent to the module shown above. (Source: Incpetion v2)

3.3.3. Use auxiliary classifiers

Inception-v1 introduces the concept of auxiliary classifiers to improve the convergence of very deep networks. The experiment found that the auxiliary classifier did not lead to improved convergence in the early training stage, but near the end of training, the network with auxiliary classifiers began to surpass the accuracy of the network without any branches, reaching a higher level of stability.

Auxiliary Classifiers were already suggested in GoogLeNet / Inception-v1. There are some modifications in Inception-v3.

Only 1 auxiliary classifier is used on the top of the last 17×17 layer, instead of using 2 auxiliary classifiers. (The overall architecture would be shown later.)



The purpose is also different. In GoogLeNet / Inception-v1, auxiliary classifiers are used for having deeper network. In Inception-v3, auxiliary classifier is used as regularizer. So, actually, in deep learning, the modules are still quite intuitive.

3.3.4. Effectively reduce the size of the feature map

The original inception block does not have the function of reducing the size of the feature map. If you want to reduce the feature map, you must implement the block plus a pooling layer . Suppose there is a feature map of $d\times d\times k$, in order to convert into a size of $d/2\times d/2\times 2k$, you can first use 1×1 convolution to become $d\times d\times 2k$, and then perform pooling, such a calculation The amount is very large (the right half of the figure below is also the method used in v1). However, first pooling and then adding channels will cause the problem of characterizing bottlenecks (the left half of the figure below).

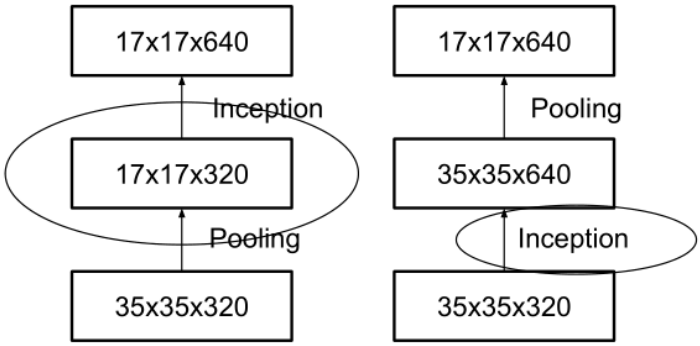


Figure 9. Two alternative ways of reducing the grid size. The solution on the left violates the principle 1 of not introducing an representational bottleneck from Section 2. The version on the right is 3 times more expensive computationally.

In order to reduce the amount of calculation and retain the feature expression, the author proposes a two-line structure, using the convolution part and the pooling part, both of which have a step length of 2. By increasing the step length of the two parts to 2, the improved inception block is given the function of reducing the size of the feature map , and the pooling outside the block is directly integrated into the block. It balances the relationship between the retention of information and the reduction of calculation. Combine the two parts by depth as shown below (the left half is from the perspective of operation, and the right half is from the perspective of the size of the feature map).

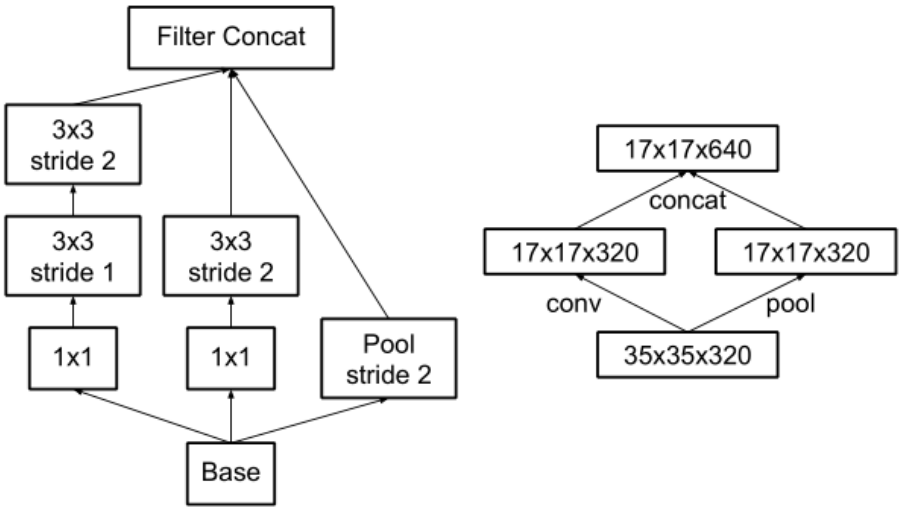


Figure 10. Inception module that reduces the grid-size while expands the filter banks. It is both cheap and avoids the representational bottleneck as is suggested by principle 1. The diagram on the right represents the same solution but from the perspective of grid sizes rather than the operations.

- The above three principles were used to build three different types of inception modules (Let's call them modules A, B and C in the order they were introduced. These names are introduced for clarity, and not the official names). The architecture is as follows:

Use the four methods in the previous section to improve inception-v1 to form the structure in the figure below (**the BN structure appeared before this, and v2 and v3 both use this structure**).

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Here, “figure 5” is module A, “figure 6” is module B and “figure 7” is module C. (Source: Incpetion v2)

3.3.5. Internal covariate shift

The change in the distribution of network activations due to the change in network parameters during training is defined as ICS. As inspired by the whitening technique for computer vision, we think the normalization to help training of the model. This is often performed to the input of the image. For example, the values of pixels of an image are normalized by subtracting the mean and being divided by the standard deviation.

Now, the question is can we do the same processing for all the internal input from layer to layer in a network.

- Possible solution 1: Suppose the normalization is independent to the optimization. But the experiments of this hypothesis shows that the model blows up, because the normalization is outside the gradient descent step.
- Possible solution 2: Take into account the normalization when doing gradient descent. However, the calculation cost is too high because of the computation of covariance matrix of all examples.

3.3.6. Batch Normalization

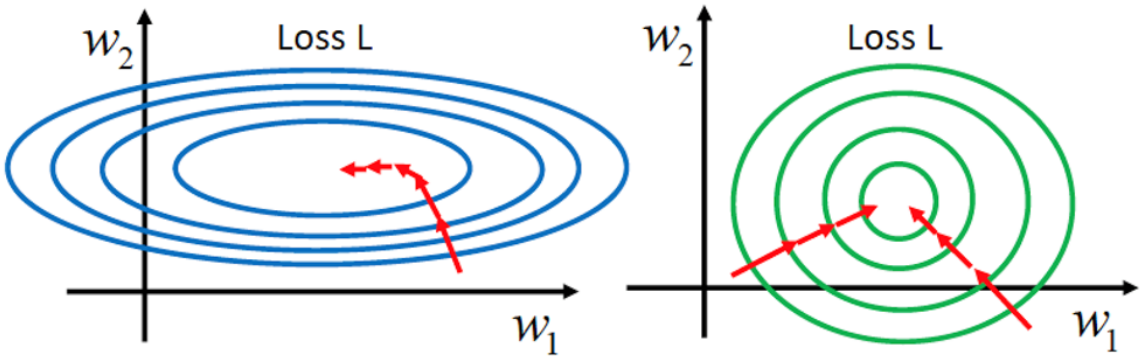
Batch normalization (BN) was introduced in Inception-v2 / BN-Inception. ReLU is used as activation function to address the saturation problem and the resulting vanishing gradients. But it also makes the output more irregular. It is advantageous for the distribution of X to remain fixed over time because a small change will be amplified when network goes deeper. Higher learning rate can be used.

As we should know, the input X is multiplied by weight W and added by bias b and become the output Y at the next layer after an activation function F:

$$Y = F(W \cdot X + b)$$

Previously, F is sigmoid function which is easily saturated at 1 which easily makes the gradient become zero. As the network depth increases, this effect is amplified, and thus slow down the training speed.

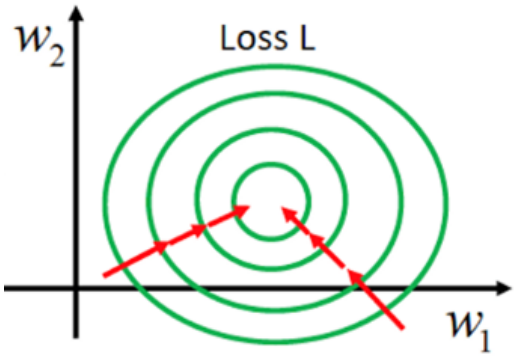
ReLU is then used as F, where $ReLU(x) = max(x, 0)$, to address the saturation problem and the resulting vanishing gradients. However, careful initialization, learning rate settings are required.



Without BN (Left), With BN (Right)

It is advantageous for the distribution of X to remain fixed over time because a small change will be amplified when network goes deeper.

BN can reduce the dependence of gradients on the scale of the parameters or their initial values. As a result, * Higher learning rate can be used. * The need for Dropout can be reduced.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch Normalization

During training, we estimate the mean μ and variance σ^2 of the mini-batch as shown above. And the input is normalized by subtracting the mean μ and dividing it by the standard deviation σ. (The epsilon ε is to prevent denominator from being zero) And additional learnable parameters γ and β are used for scale and shift to have a better shape and position after normalization. And output Y becomes as follows:

$$Y = F(BN(W \cdot X + b))$$

To have a more precise mean and variance, moving average is used to calculate the mean and variance.

During testing, the mean and variance are calculated using the population.

The full training and inference algorithm is as follows:

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

4. Inception v3

With version 1 and version 2, Inception have introduced sparse representation to reduce the calculation and batch normalization to speed up and stabilize the training. In version 3, the authors want to explore ways to scale up networks in ways that aim at utilizing the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization. They give some general design principles according to their large scale experimentation with various architectural.

- Avoid representational bottlenecks, especially early in the network.
- Higher dimensional representations are easier to pro- cess locally within a network.
- Spatial aggregation can be done over lower dimensional embeddings without much or any loss in representational power.
- Balance the width and depth of the network.

4.1 The Premise

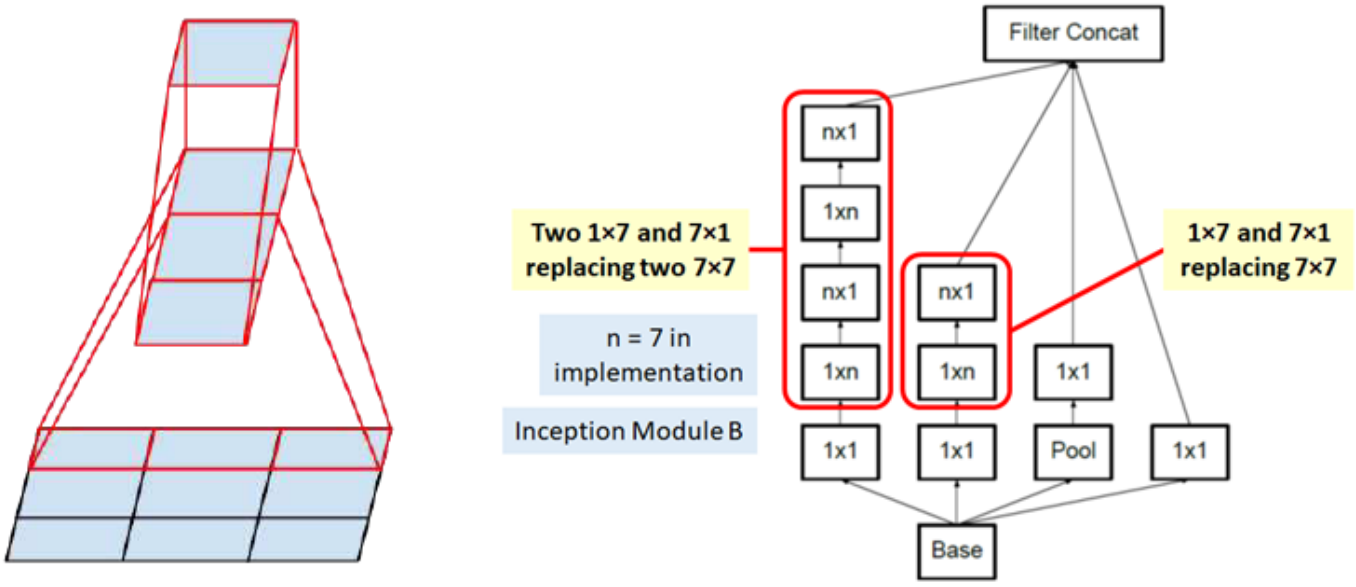
- The authors noted that the auxiliary classifiers didn't contribute much until near the end of the training process, when accuracies were nearing saturation. They argued that they function as regularizes, especially if they have BatchNorm or Dropout operations.
- Possibilities to improve on the Inception v2 without drastically changing the modules were to be investigated.

4.2 The Solution

Inception Net v3 incorporated all of the above upgrades stated for Inception v2, and in addition used the following:

- RMSProp Optimizer.
- Factorized 7x7 convolutions.
- BatchNorm in the Auxillary Classifiers.
- Label Smoothing (A type of regularizing component added to the loss formula that prevents the network from becoming too confident about a class. Prevents over fitting).

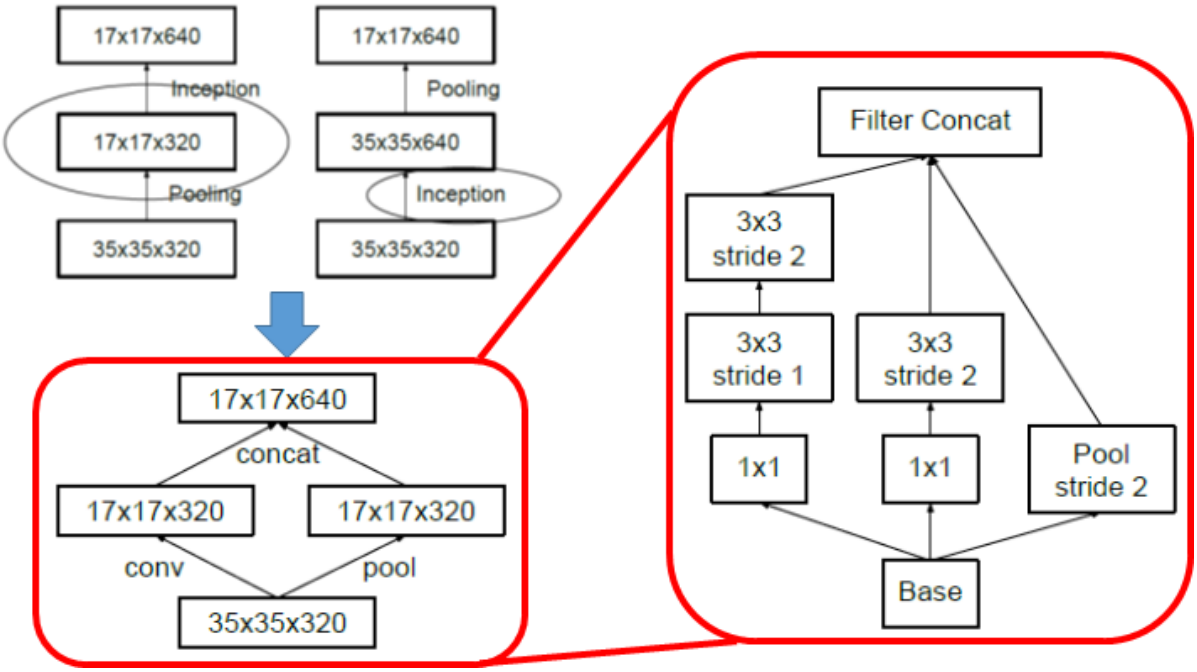
4.2.1. Factorization



3×3 conv becomes 1×3 and 3×1 convs (Left), 7×7 conv becomes 1×7 and 7×1 convs (Right)

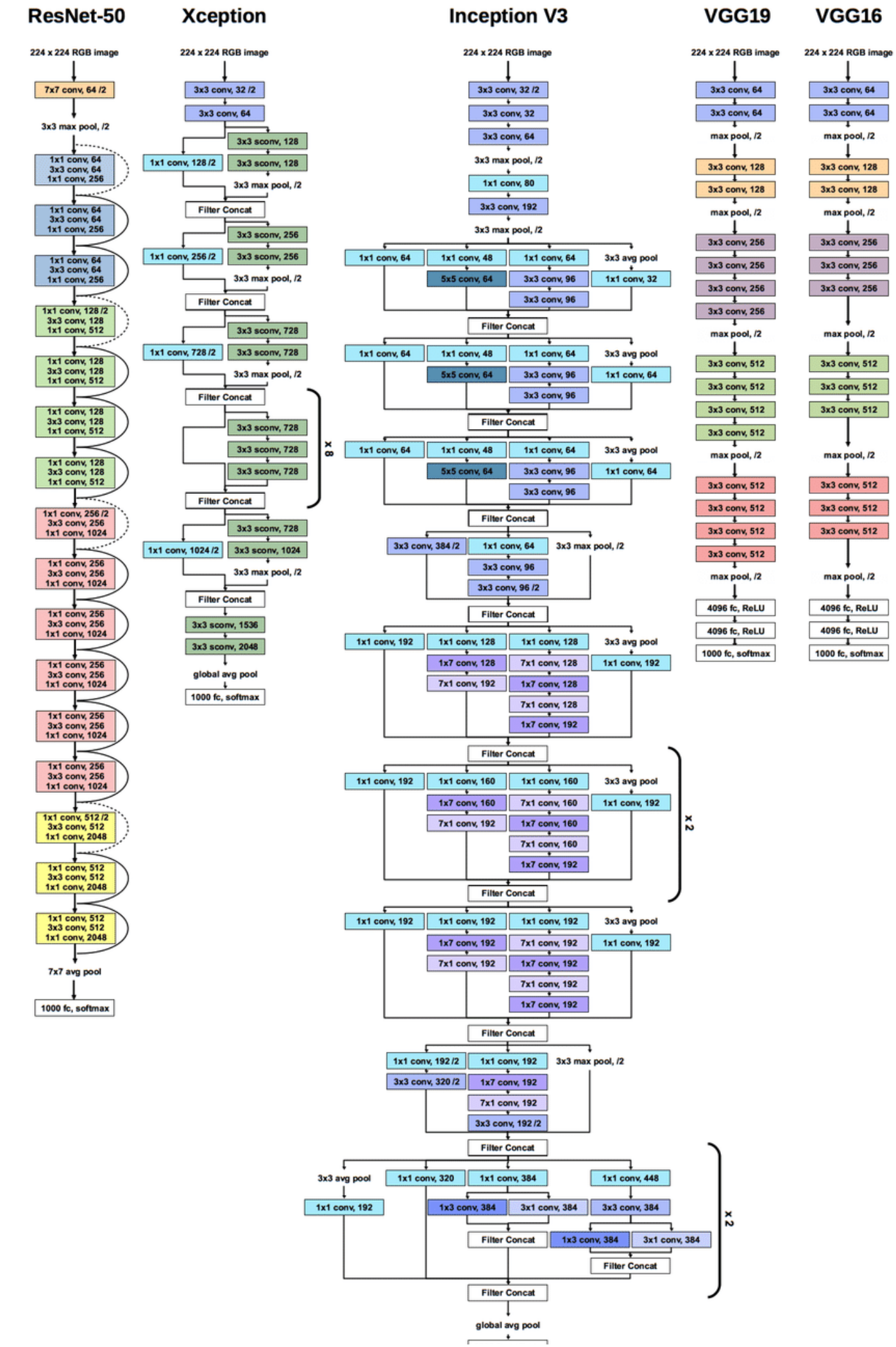
Factorization was introduced in convolution layer as shown above to further reduce the dimensionality, so as to reduce the overfitting problem. For example:

By using 3×3 filter, number of parameters = 3×3=9 By using 3×1 and 1×3 filters, number of parameters = 3×1+1×3=6 Number of parameters is reduced by 33%



Conventional downsizing (Top Left), Efficient Grid Size Reduction (Bottom Left), Detailed Architecture of Efficient Grid Size Reduction (Right)

And an efficient grid size reduction module was also introduced which is less expensive and still efficient network. With the efficient grid size reduction, say for example in the figure, 320 feature maps are done by conv with stride 2. 320 feature maps are obtained by max pooling. And these 2 sets of feature maps are concatenated as 640 feature maps and go to the next level of inception module.


ResNet-50
224 x 224 RGB image
7x7 conv, 64 /2
3x3 max pool, /2
1x1 conv, 64
3x3 conv, 64
1x1 conv, 256
1x1 conv, 128 /2
3x3 conv, 128
1x1 conv, 512
1x1 conv, 256 /2
3x3 conv, 256
1x1 conv, 1024
1x1 conv, 512 /2
3x3 conv, 512
1x1 conv, 2048
7x7 avg pool
1000 fc, softmax
Xception
224 x 224 RGB image
3x3 conv, 32 /2
3x3 conv, 64
3x3 sconv, 128
1x1 conv, 128 /2
3x3 max pool, /2
Filter Concat
3x3 sconv, 256
1x1 conv, 256 /2
3x3 sconv, 728
1x1 conv, 728 /2
x 8
3x3 sconv, 1024
1x1 conv, 1024 /2
3x3 sconv, 1536
3x3 sconv, 2048
global avg pool
1000 fc, softmax
Inception V3
224 x 224 RGB image
3x3 conv, 32 /2
3x3 conv, 32
3x3 conv, 64
3x3 max pool, /2
1x1 conv, 80
3x3 conv, 192
1x1 conv, 64
1x1 conv, 48
5x5 conv, 64
3x3 conv, 96
3x3 avg pool
1x1 conv, 32
1x1 conv, 384 /2
3x3 max pool, /2
1x1 conv, 128
1x7 conv, 128
7x1 conv, 128
7x1 conv, 192
1x7 conv, 192
1x1 conv, 160
7x1 conv, 160
1x7 conv, 160
x 2
3x3 conv, 320 /2
1x3 conv, 384
3x1 conv, 384
1x1 conv, 448
global avg pool
VGG19
224 x 224 RGB image
3x3 conv, 64
max pool, /2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
4096 fc, ReLU
1000 fc, softmax
VGG16
224 x 224 RGB image
3x3 conv, 64
max pool, /2
3x3 conv, 128
3x3 conv, 256
3x3 conv, 512
4096 fc, ReLU
1000 fc, softmax

Comparison with VGG

4.2.2. Label Smoothing

In cross entropy, the direct use the ground true to indicate the probability of certain class to 1 can cause 2 problems:

- causes overfitting
- encourages the differences between the largest logit and all others to become large, and this reduces the ability of the model to adapt.

The label smoothing can be simply understood as lower the probability of ground true from 1 to certain value, 0.9 for example.

```
new_labels = (1 − ε) * one_hot_labels + ε / K
```

where ε is 0.1 which is a hyperparameter and K is 1000 which is the number of classes. A kind of dropout effect observed in classifier layer.

5. Inception v4

Inception v4 and Inception-ResNet were introduced in the same paper. For clarity, let us discuss them in separate sections.

Investigate whether Inception itself can become more efficient by becoming deeper and wider. Because Tensorflow has begun to be widely used, it is no longer necessary to manually optimize the use of memory, and the module specifications are unified under this premise.

5.1 The Premise

Make the modules more uniform. The authors also noticed that some of the modules were more complicated than necessary. This can enable us to boost performance by adding more of these uniform modules.

5.2 The Solution

The “stem” of Inception v4 was modified. The stem here, refers to the initial set of operations performed before introducing the Inception blocks.

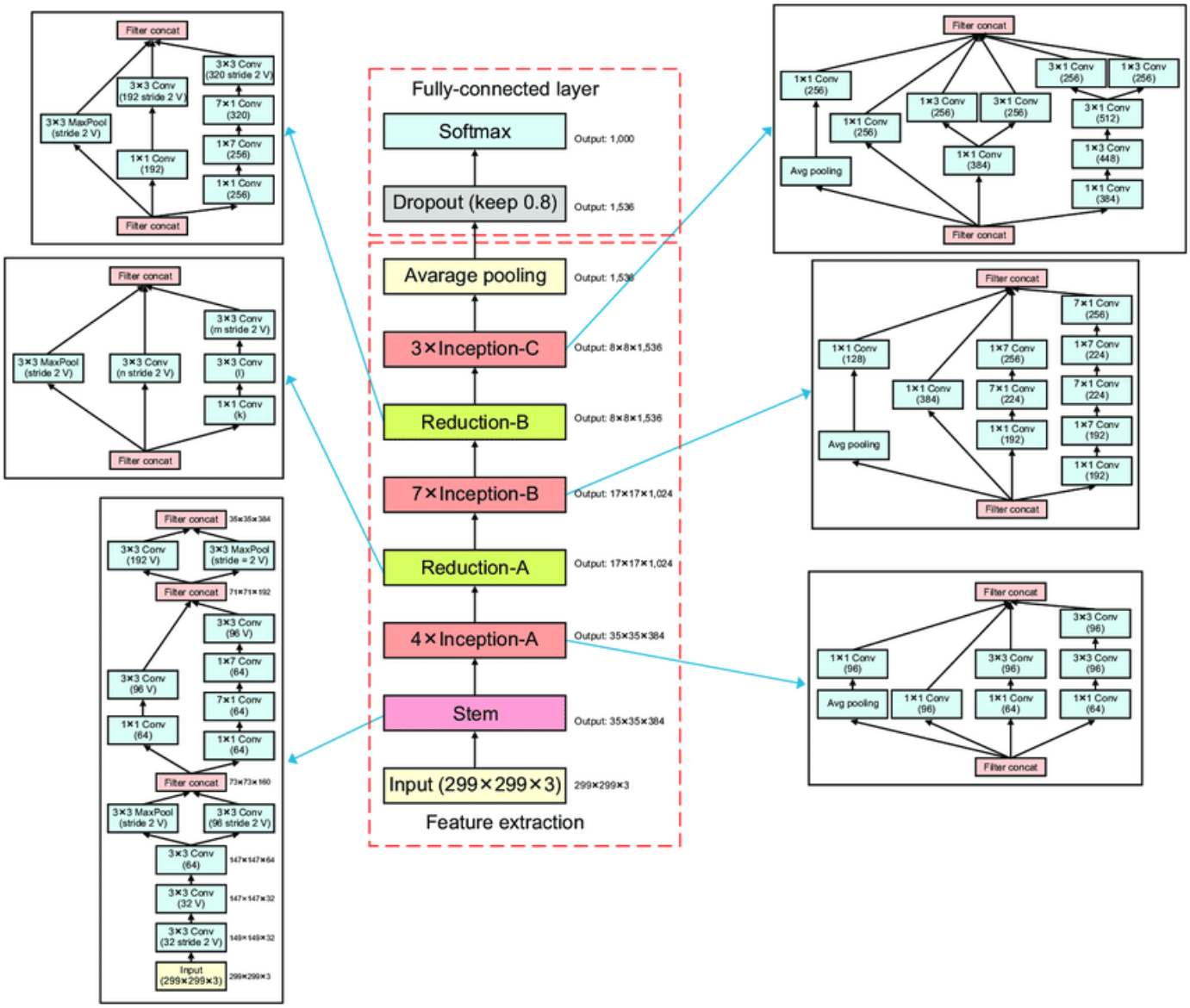
The initial set of layers which the paper refers “stem of the architecture” was modified to make it more uniform . These layers are used before Inception block in the architecture.

This model can be trained without partition of replicas unlike the previous versions of inceptions which required different replica in order to fit in memory. This architecture use memory optimization on back propagation to reduce the memory requirement.

Structure

The overall view is divided into three parts: 1. Stem block 2. 3.main Inception modules 3. Two transition blocks for reducing the size of the feature map.

(1) The overall structure is as follows:



(2) Stem block structure:

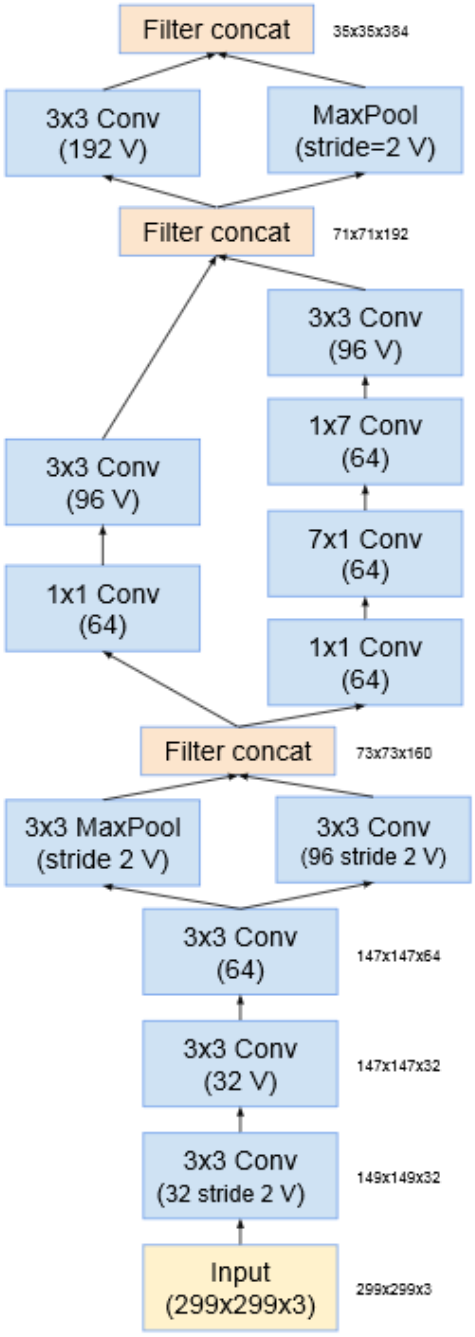


Figure 3. The schema for stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is the input part of those networks. Cf. Figures 9 and 15

Stem of Inception v4 and Inception-ResNet v2. (Source: [Inception v4]https://arxiv.org/pdf/1602.07261.pdf))

(3) 3.main Inception block structures

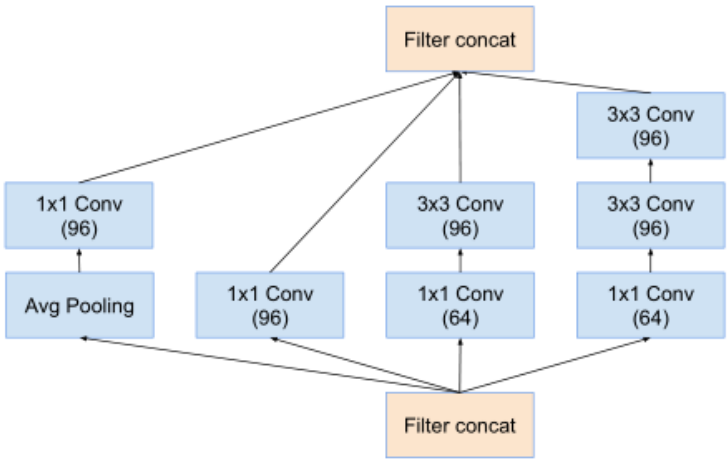


Figure 4. The schema for 35 × 35 grid modules of the pure Inception-v4 network. This is the Inception-A block of Figure 9.

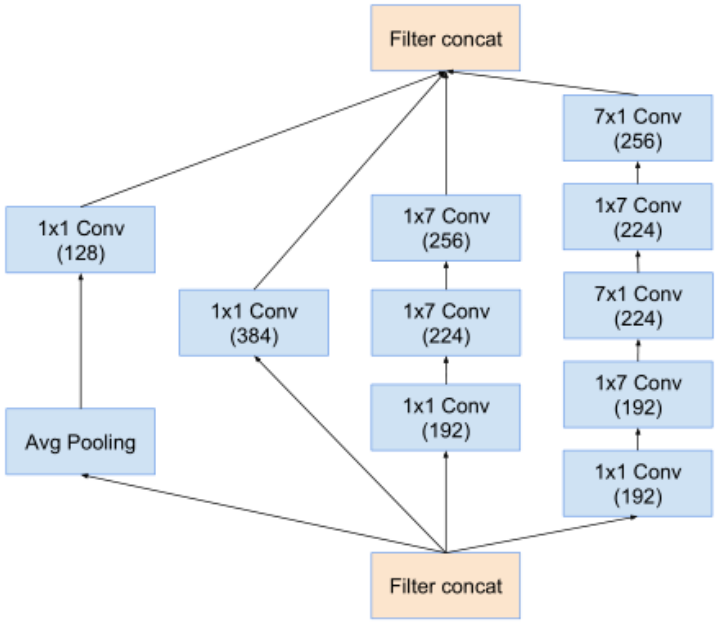


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

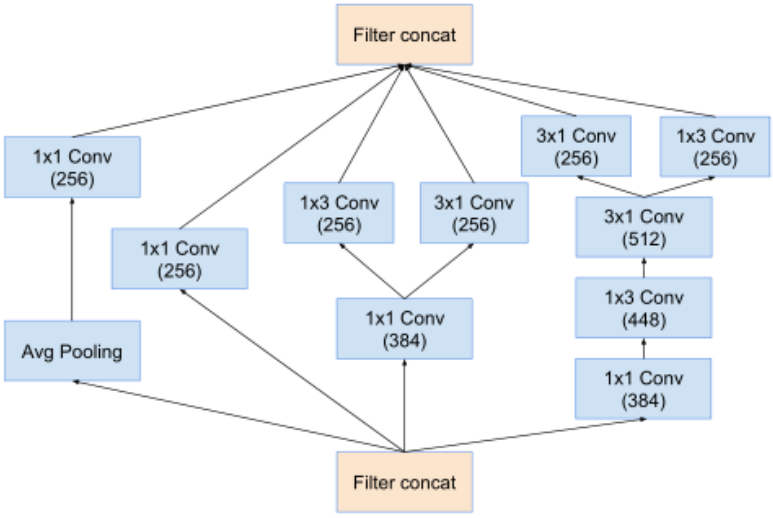
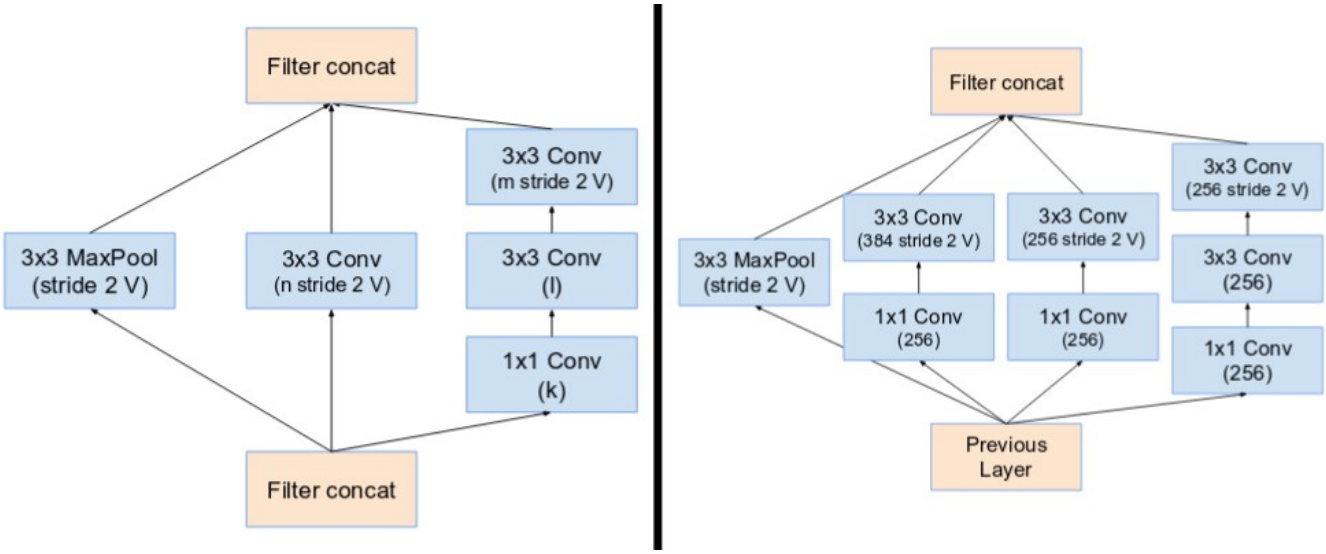


Figure 6. The schema for 8 × 8 grid modules of the pure Inception-v4 network. This is the Inception-C block of Figure 9.

(4) Two transition block structure Inception v4 introduced specialized “Reduction Blocks” which are used to change the width and height of the grid. The earlier versions didn’t explicitly have reduction blocks, but the functionality was implemented.

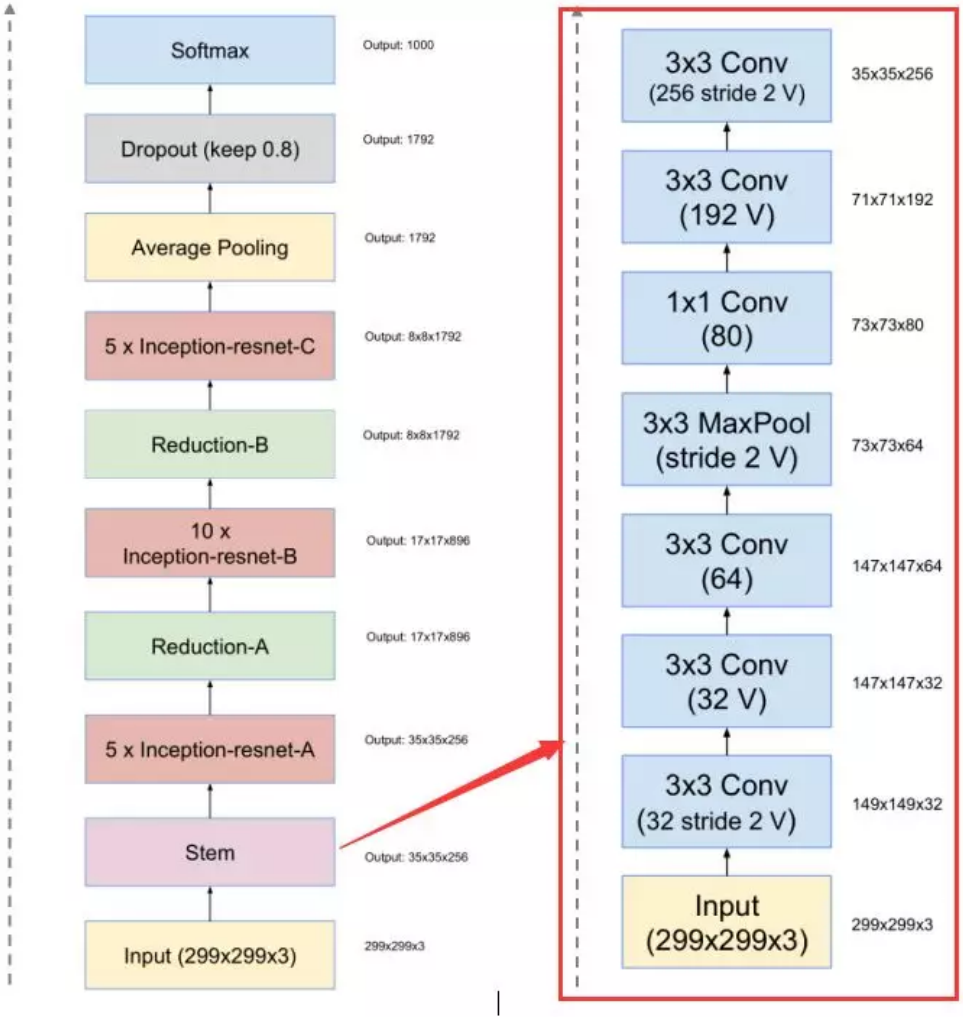


(From Left) Reduction Block A (35×35 to 17×17 size reduction) and Reduction Block B (17×17 to 8×8 size reduction). Refer to the paper for the exact hyper-parameter setting (V,l,k). (Source: Inception v4 (https://arxiv.org/pdf/1602.07261.pdf))

6. Inception-ResNet v1 and v2

Inspired by the performance of the ResNet, a hybrid inception module was proposed. There are two sub-versions of Inception ResNet, namely v1 and v2. Before we checkout the salient features, let us look at the minor differences between these two sub-versions.

- Inception-ResNet v1 has a computational cost that is similar to that of Inception v3.
- Inception-ResNet v2 has a computational cost that is similar to that of Inception v4.
- They have different stems, as illustrated in the Inception v4 section.
- Both sub-versions have the same structure for the modules A, B, C and the reduction blocks. Only difference is the hyper-parameter settings. In this section, we’ll only focus on the structure. Refer to the paper for the exact hyper-parameter settings (The images are of Inception-Resnet v1).



Stem of Inception-ResNet v1. (Source: Inception v4 (https://arxiv.org/pdf/1602.07261.pdf))

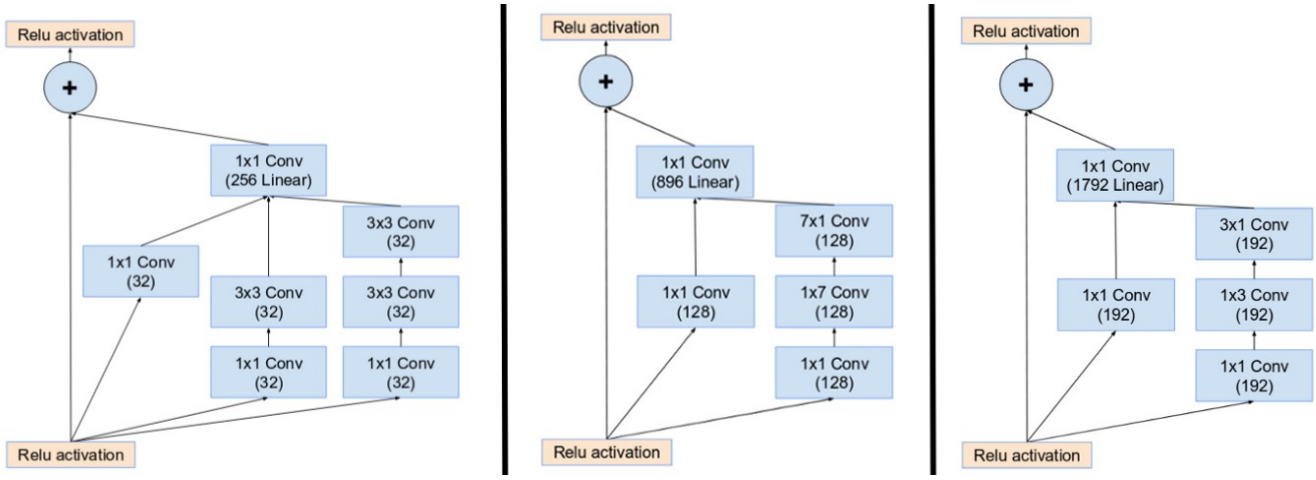
6.1 The Premise

Introduce residual connections that add the output of the convolution operation of the inception module, to the input.

In view of the fact that ResNet has achieved very good performance, and ResNet focuses on simplifying the optimization problem to speed up the convergence speed , while inception focuses on the processing of feature maps, and the two do not conflict. Naturally, I thought of combining the two to improve the convergence speed and accuracy of GoogleNet.

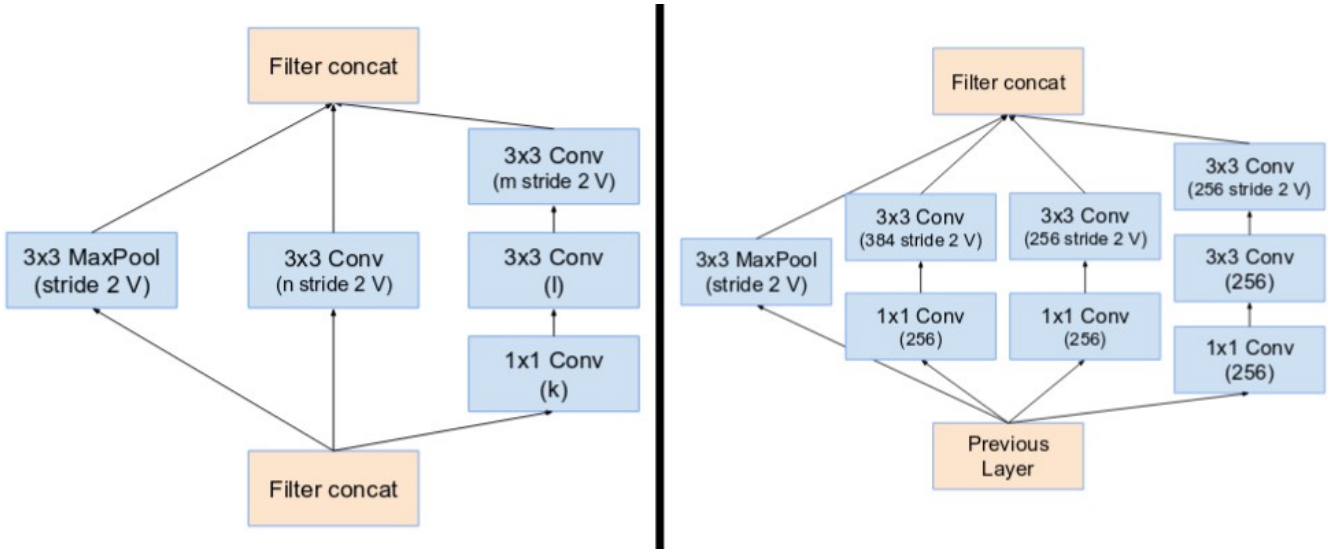
6.2 The Solution

For residual addition to work, the input and output after convolution must have the same dimensions. Hence, we use 1×1 convolutions after the original convolutions, to match the depth sizes (Depth is increased after convolution).



(From left) Inception modules A,B,C in an Inception ResNet. Note how the pooling layer was replaced by the residual connection, and also the additional 1×1 convolution before addition. (Source: Inception v4 (https://arxiv.org/pdf/1602.07261.pdf))

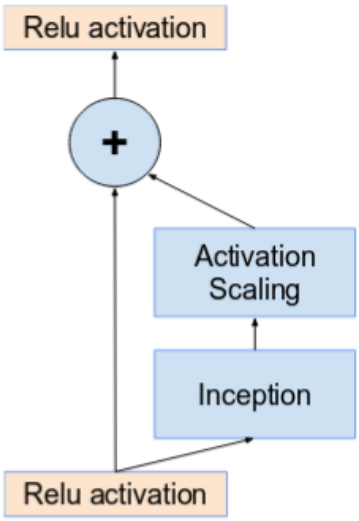
The pooling operation inside the main inception modules were replaced in favor of the residual connections. However, you can still find those operations in the reduction blocks. Reduction block A is same as that of Inception v4.



(From Left) Reduction Block A (35×35 to 17×17 size reduction) and Reduction Block B (17×17 to 8×8 size reduction). Refer to the paper for the exact hyper-parameter setting (V,l,k). (Source: Inception v4 (https://arxiv.org/pdf/1602.07261.pdf))

Inception architecture with residuals

Networks with residual units deeper in the architecture caused the network to “die” if the number of filters exceeded 1000. Hence, to increase stability, the authors scaled the residual activations by a value around 0.1 to 0.3.



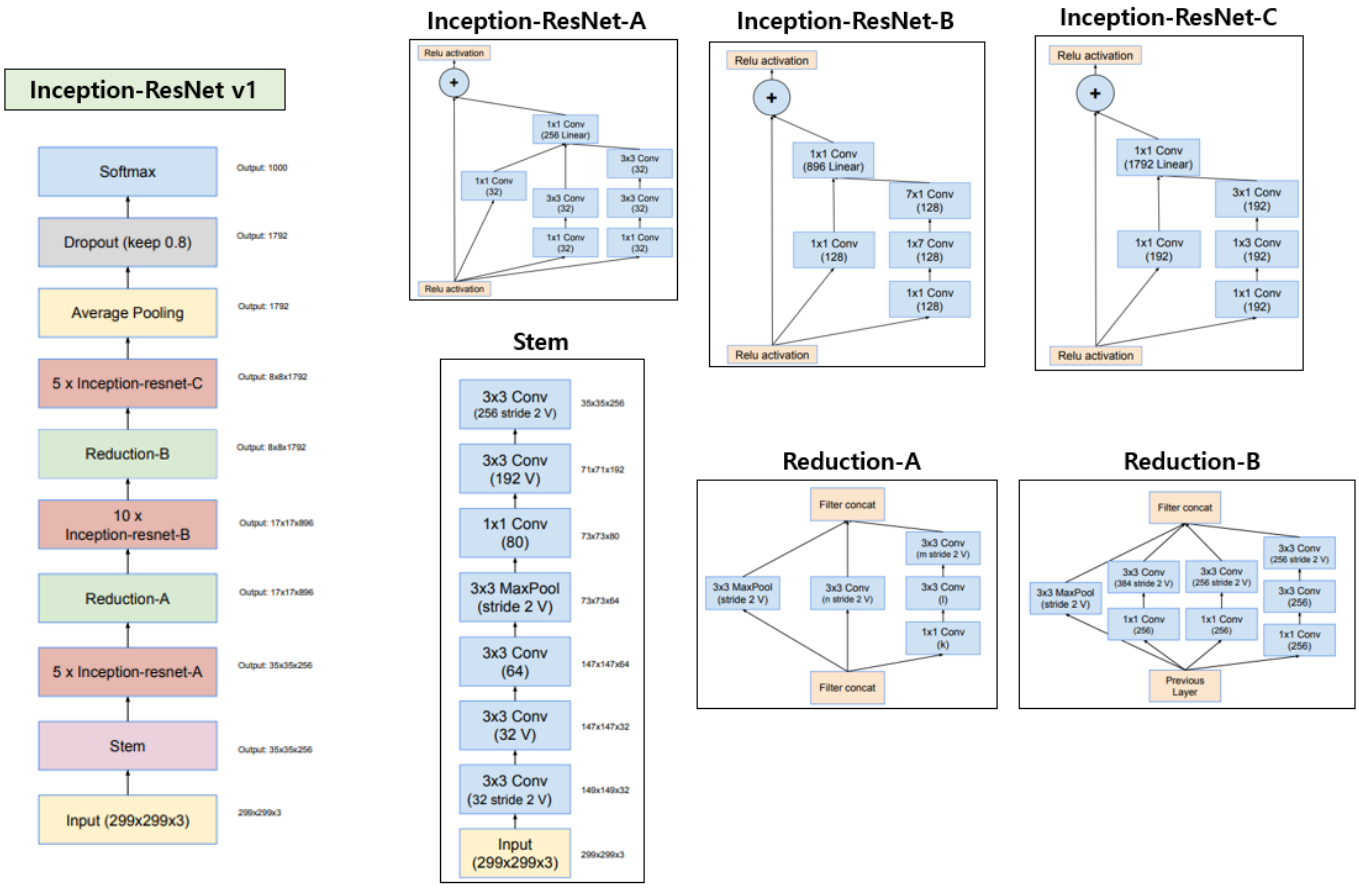
Activations are scaled by a constant to prevent the network from dying. (Source: Inception v4 (https://arxiv.org/pdf/1602.07261.pdf))

- The authors of the paper was inspired by the success of Residual Network. Therefore they explored the possibility of combining the Inception with ResNets. They proposed two Residual Network based Inception models: Inception ResNet V1 and Inception ResNet V2. Let's look at the key highlights of these architectures.
- The Inception block used in these architecture are computationally less expensive than original Inception blocks that we used in Inception V4.
- Each Inception block is followed by a 1×1 convolution without activation called filter expansion. This is done to scale up the dimensionality of filter bank to match the depth of input to next layer.
- The pooling operation inside the Inception blocks were replaced by residual connections. However, pooling operations can be found in reduction blocks.
- In Inception ResNets models, the batch normalization does not used after summations. This is done to reduce the model size to make it trainable on a single GPU.
- Both the Inception architectures have same architectures for Reduction Blocks, but have different stem of the architectures. They also have difference in their hyper parameters for training.
- It is found that Inception-ResNet V1 have similar computational cost as of Inception V3 and Inception-ResNet V2 have similar computational cost as of Inception V4. The original paper didn't use BatchNorm after summation to train the model on a single GPU (To fit the entire model on a single GPU).

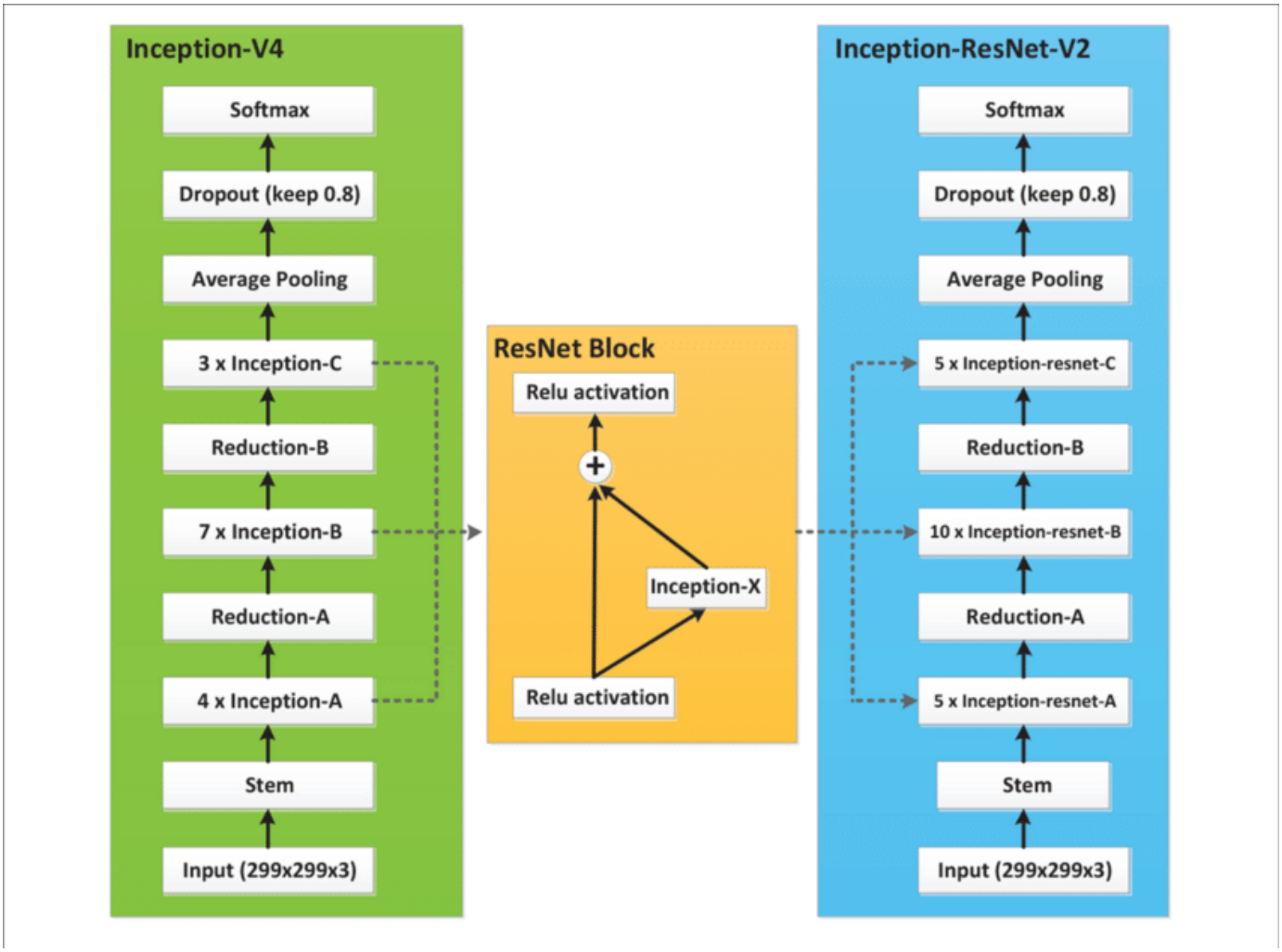
It was found that Inception-ResNet models were able to achieve higher accuracies at a lower epoch.

The final network layout for both Inception v4 and Inception-ResNet are as follows:

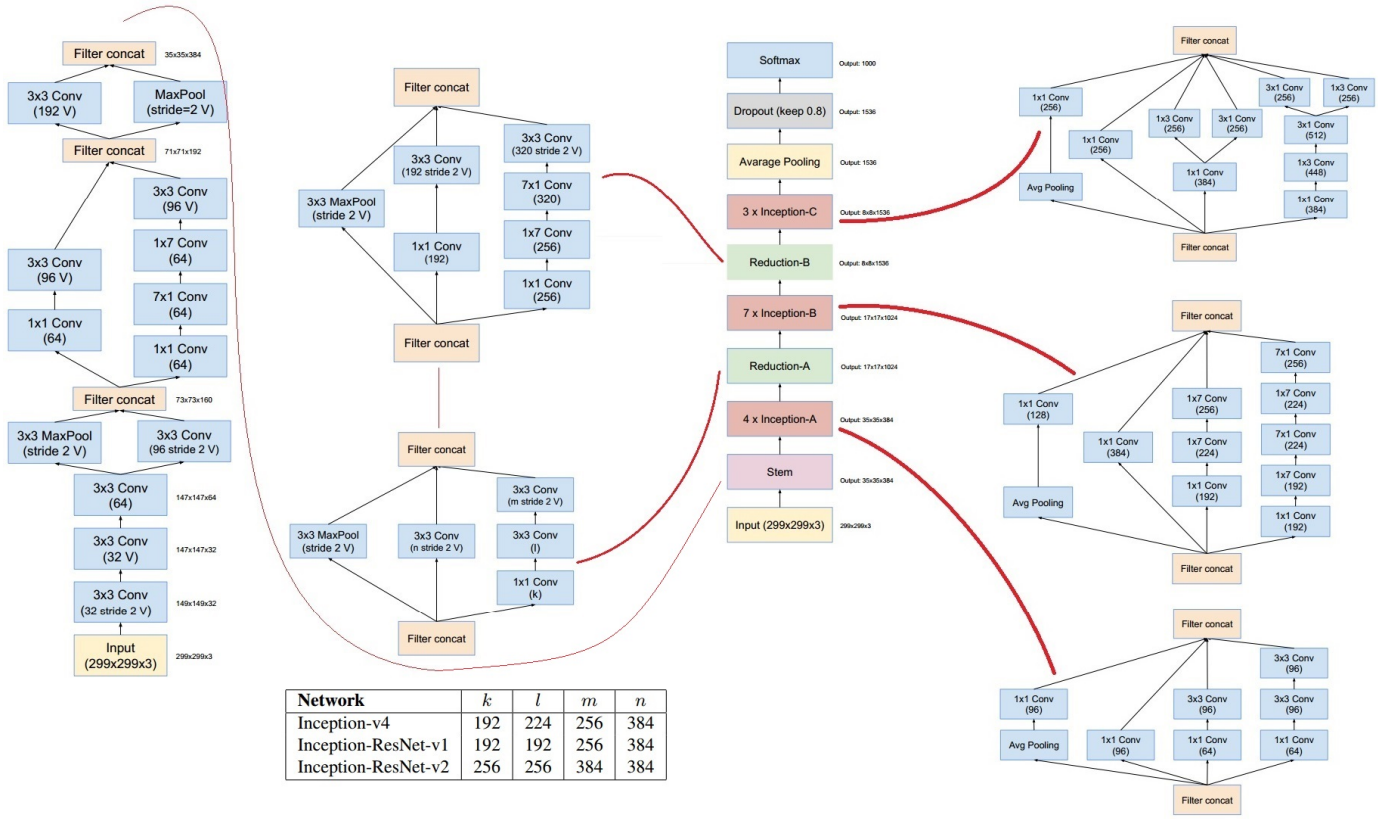
The overall structure of Inception-ResNet v1 is as follows.



Inception-ResNet v1

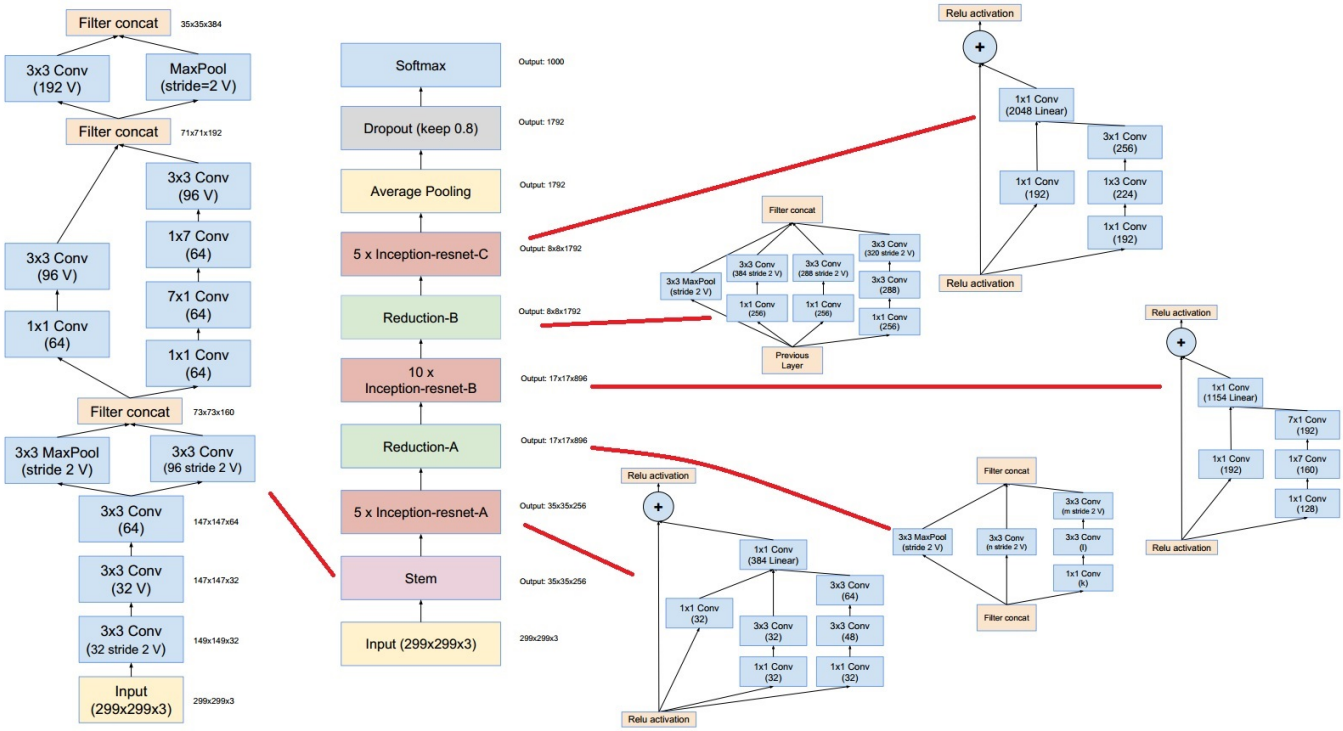


The left image is the layout of Inception v4. The right image is the layout of Inception-ResNet. (Source: Inception v4 (https://arxiv.org/pdf/1602.07261.pdf))



Network	k	l	m	n
Inception-v4	192	224	256	384
Inception-ResNet-v1	192	192	256	384
Inception-ResNet-v2	256	256	384	384

Inception-v4 Architecture



Inception-ResNet(v2) Architecture

6.3 Effect

Inception-ResNet v2, ResNet152 and Inception v4 models are similar in scale, v4 is slightly smaller, Inception-ResNet v1 and Inception v3 and ResNet50 models are of similar scale and similar in accuracy.

Last edited by : May 18, 2023, 10 a.m.

Comment 0 **Feedback**

- **Prev** : K_02 Understanding of VGG-16, VGG-19 - EN
- **Next** : K_04 Understanding of MobileNet - EN

TOP