Inception and ResNet

Inception v1 [4, 5]

- Premise
  - **Salient parts in the image can have extremely large variation in size.**

Inception v1 [4, 5]

- Premise
  - **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.

From left: A dog occupying most of the image, a dog occupying a part of it, and a dog occupying very little space (Images obtained from Unsplash).

Inception v1 [4, 5]

- **Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes tough.**

Inception v1 [4, 5]

- **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.**

Inception v1 [4, 5]

- **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.**

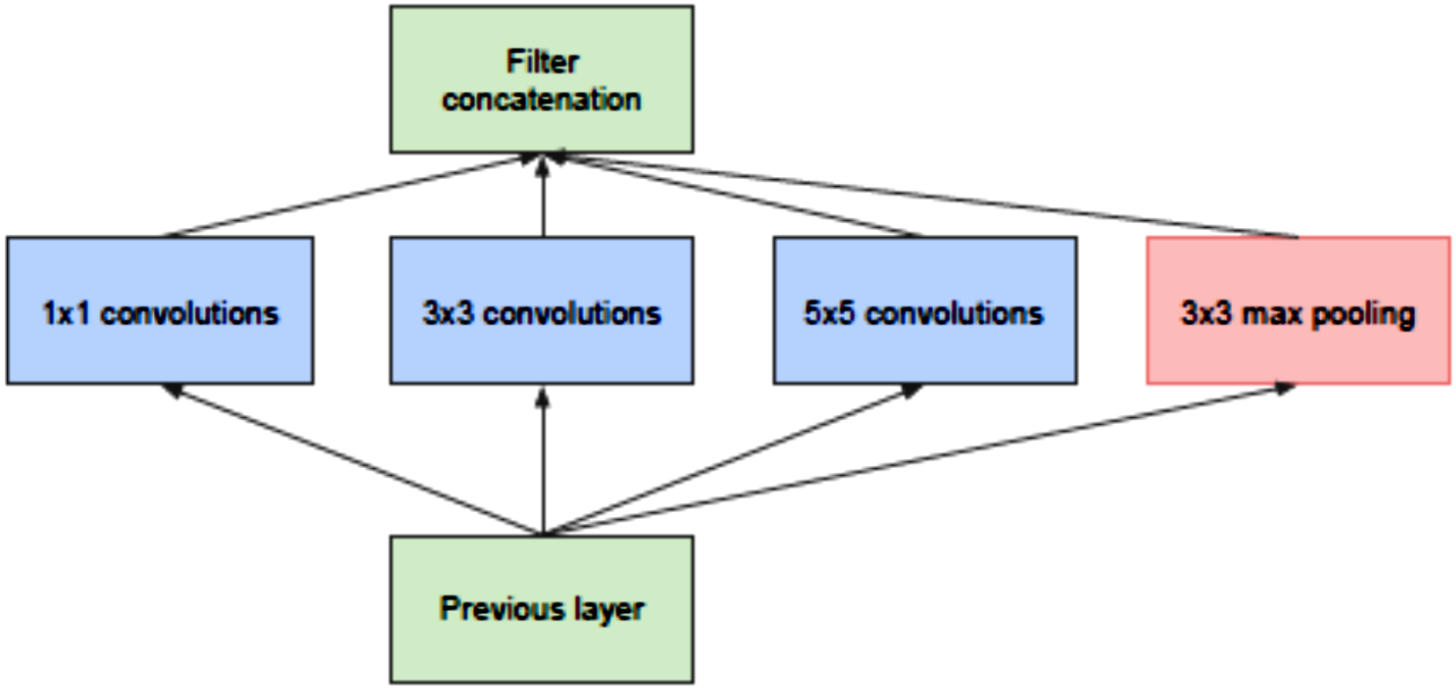
- Naively stacking large convolution operations is computationally expensive.

Inception v1 [4, 5]

- What is the solution?
  - **Why not have filters with multiple sizes operate on the same level?**

Inception v1 [4, 5]

- What is the solution?
  - **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.**

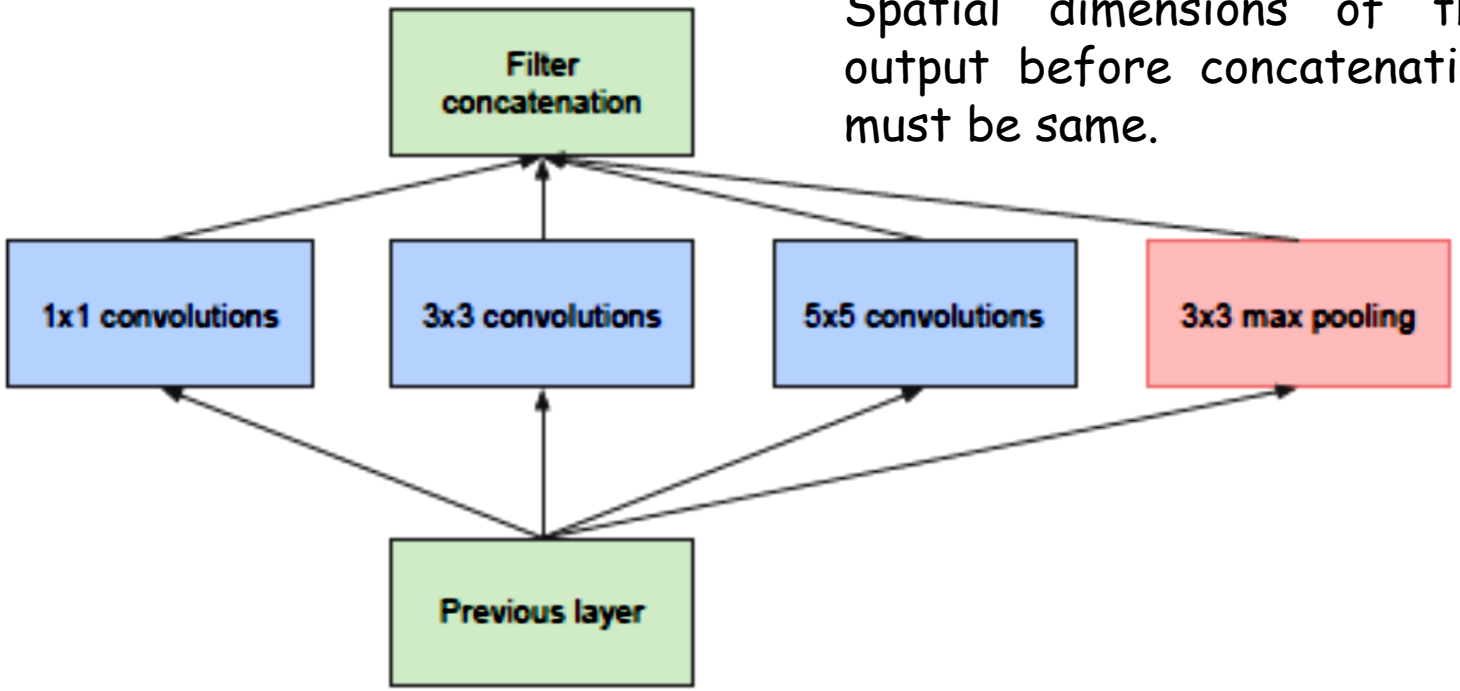


(a) Inception module, naïve version

Inception v1 [4, 5]

- What is the solution?
  - **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.**

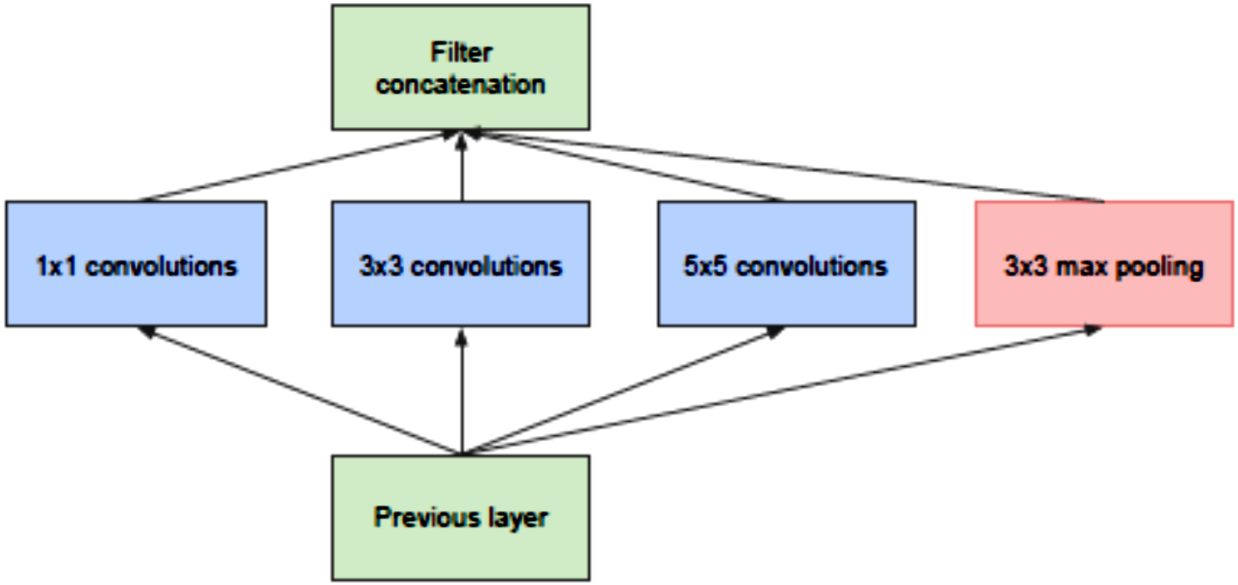
Spatial dimensions of the output before concatenation must be same.



(a) Inception module, naïve version

Inception v1 [4, 5]

- What is the solution?

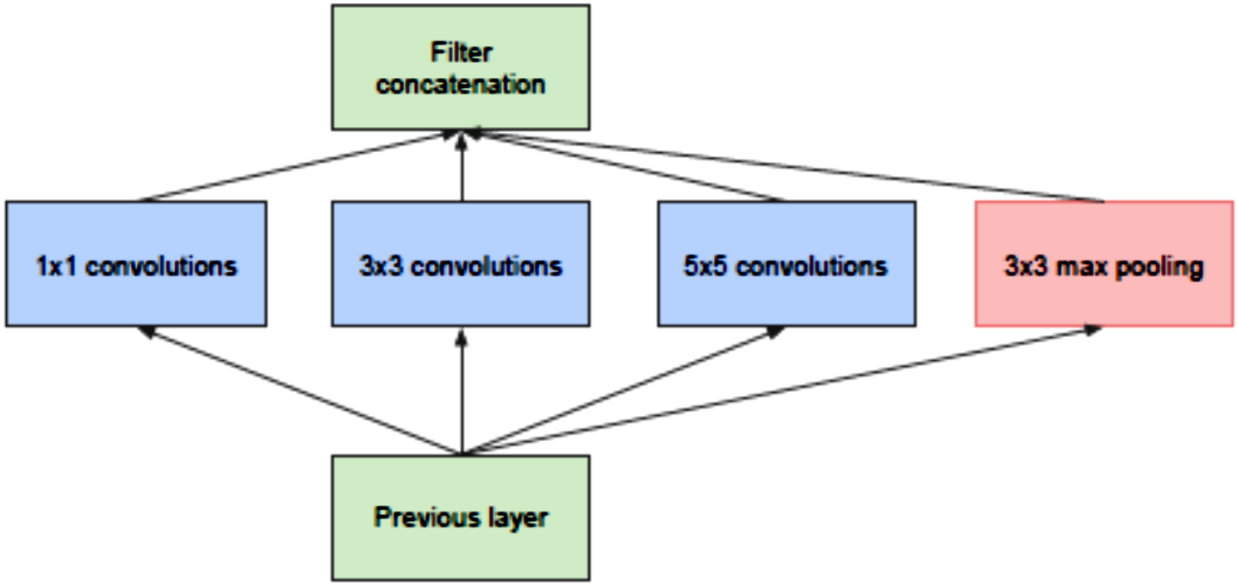


(a) Inception module, naïve version

- The above image is the "naive" inception module.

Inception v1 [4, 5]

- What is the solution?

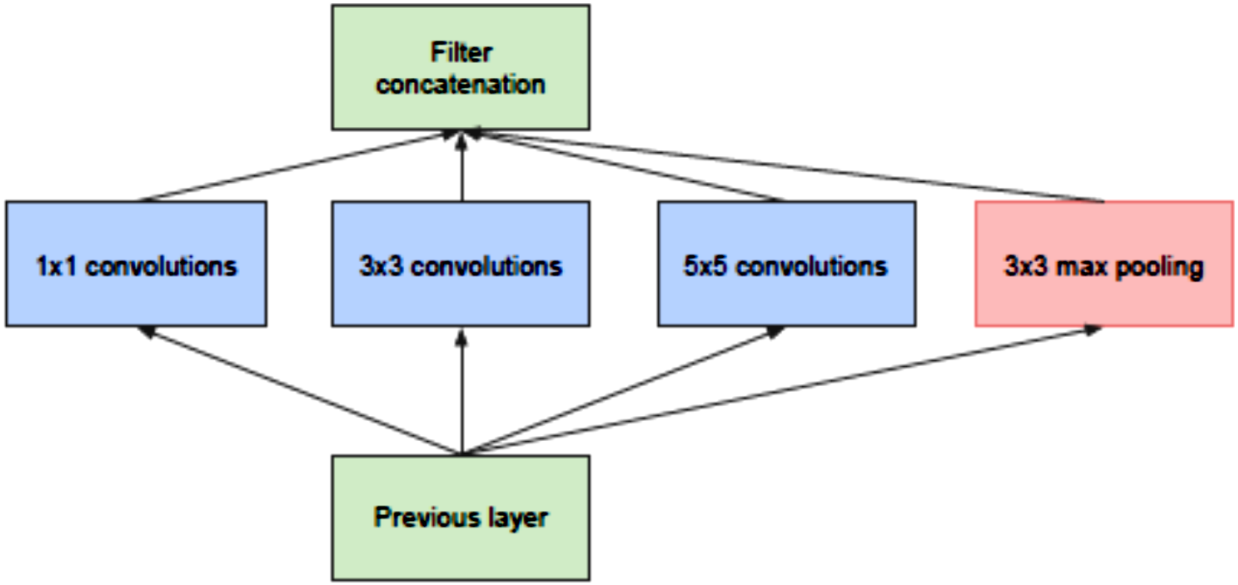


(a) Inception module, naïve version

- The above image is the "naive" inception module.
- It performs convolution on an input, with 3 different sizes of filters (1x1, 3x3, 5x5).

Inception v1 [4, 5]

- What is the solution?



(a) Inception module, naïve version

- The above image is the "naive" inception module.
- It performs convolution on an input, with 3 different sizes of filters (1x1, 3x3, 5x5).
- Additionally, max pooling is also performed. The outputs are concatenated and sent to the next inception module.

Inception v1 [6]

- Still a Problem - What is the solution?

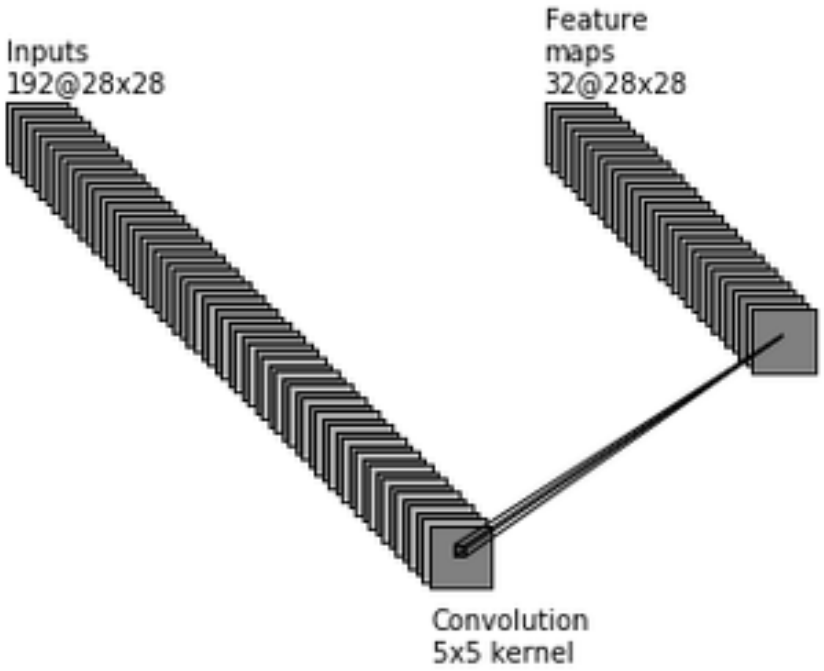


Figure 6. 5×5 convolutions inside the Inception module using the naive model

There would be

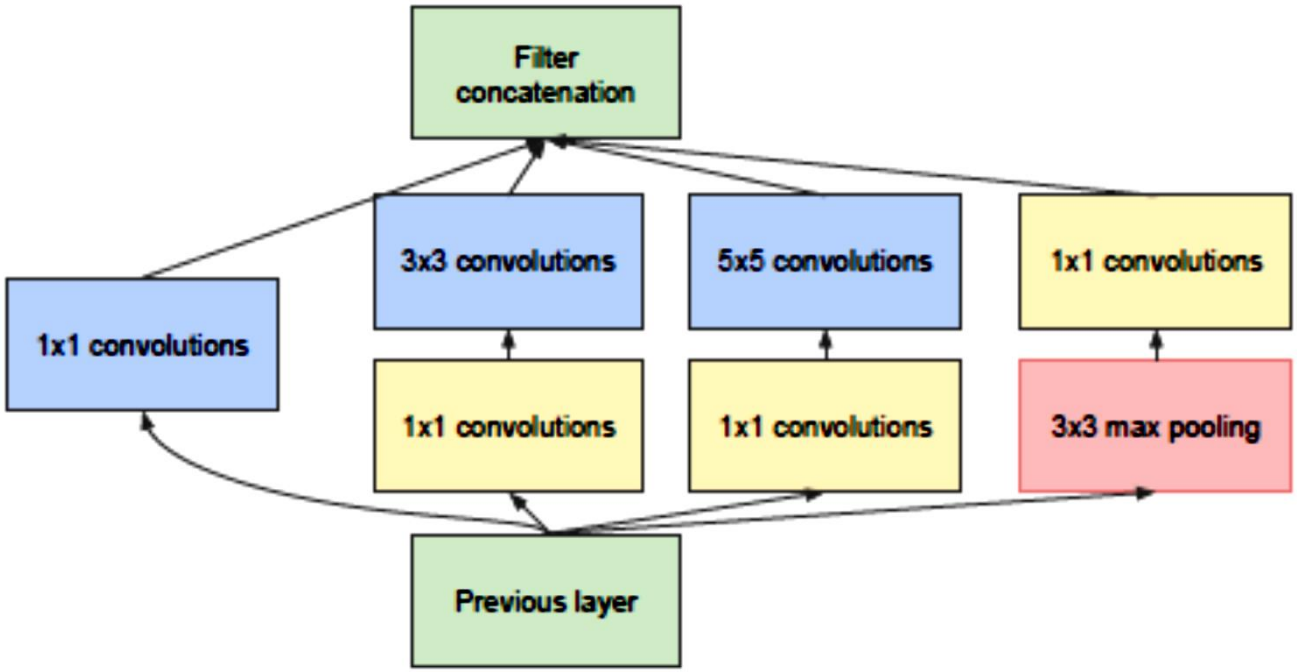
$$5^2(28)^2(192)(32) = 120,422,400 \text{ operations.}$$

Inception v1 [4, 5]

- What is the solution?
  - As stated before, deep neural networks are computationally expensive.

Inception v1 [4, 5]

- What is the solution?
  - 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 1x1 convolution before the 3x3 and 5x5 convolutions.



(b) Inception module with dimension reductions

Inception v1 [4, 5]

- What is the solution?
  - Though adding an extra operation may seem **counterintuitive**, 1x1 convolutions are far more cheaper than 5x5 convolutions, and the reduced number of input channels also help.

Inception v1 [6]

- What is the solution?

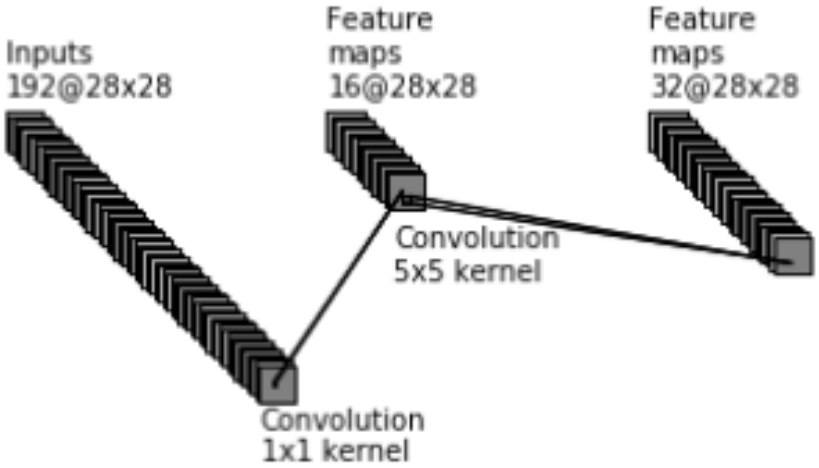


Figure 7. 1×1 convolutions serve as the dimensionality reducers that limit the number of expensive 5×5 convolutions that follow

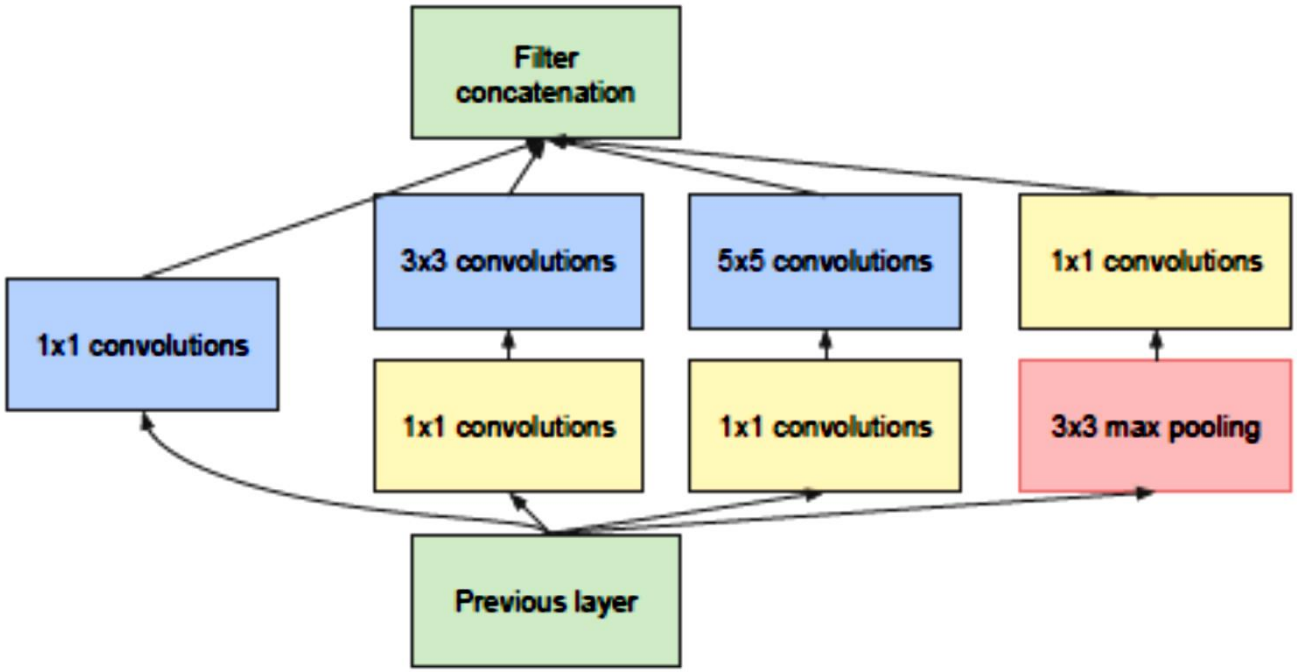
In this case, there would be

$$[(1^2)(28^2)(192)(16)]+[(5^2)(28^2)(16)(32)] = 2,408,448+10,035,200 = 12,443,648 \text{ operations.}$$

Inception v1 [4, 5]

- What is the solution?
  - Though adding an extra operation may seem **counterintuitive**, 1x1 convolutions are far more cheaper than 5x5 convolutions, and the reduced number of input channels also help.

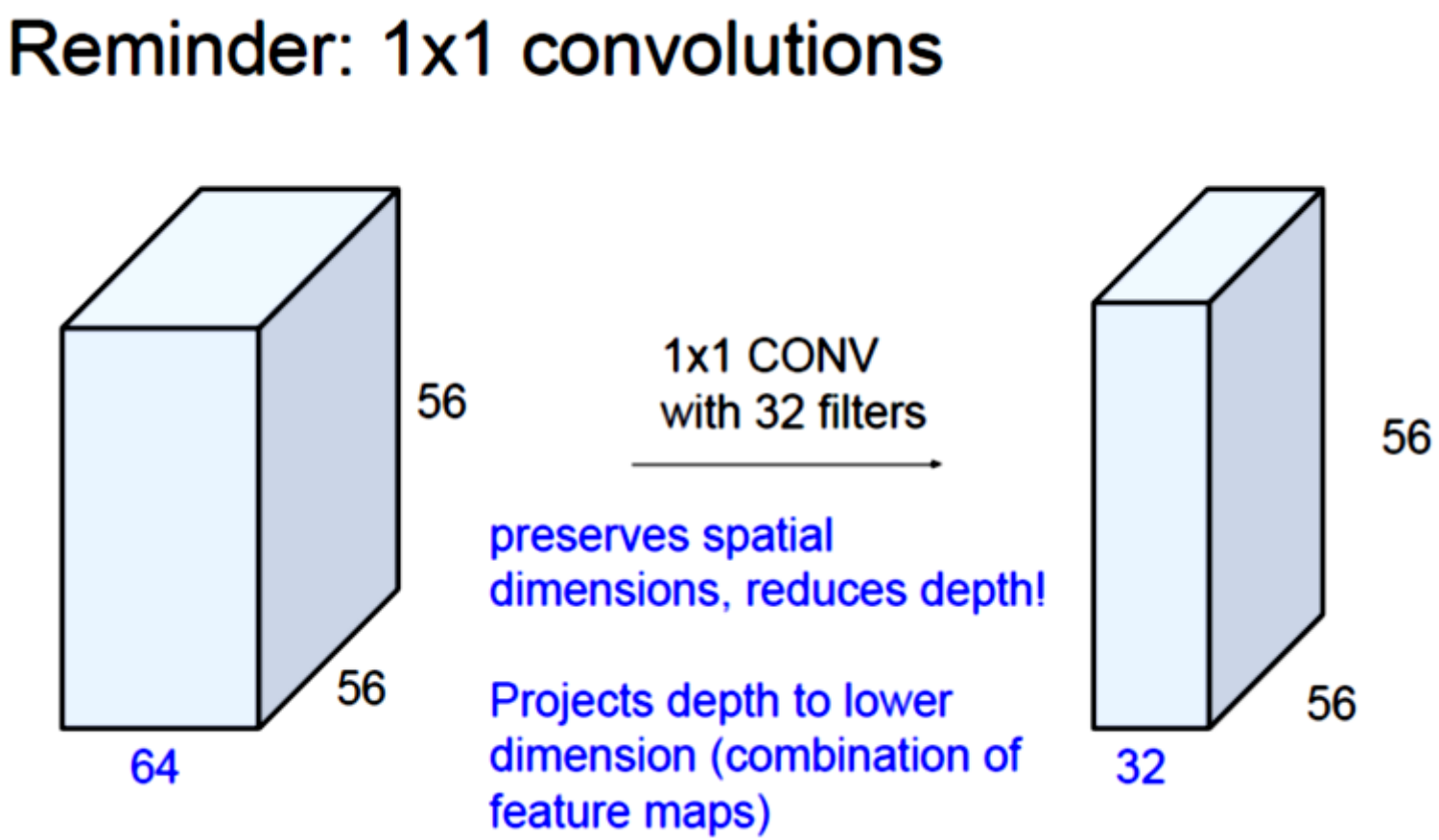
  - Do note that however, the 1x1 convolution is introduced after the max pooling layer, rather than before.



(b) Inception module with dimension reductions

Inception v1 [8]

- What is the solution?



Inception v1 [4, 5]

- 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:

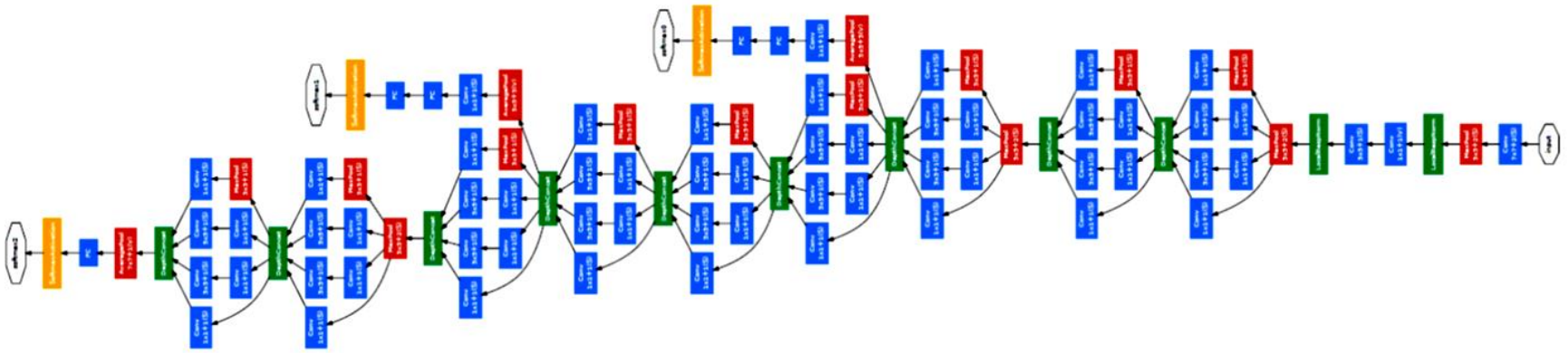


Figure 3: GoogLeNet network with all the bells and whistles

Inception v1 [4, 5]

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								

Table 1: GoogLeNet incarnation of the Inception architecture

Inception v1

- **All the convolutions, including those inside the Inception modules, use rectified linear activation.**

Inception v1

- **All the convolutions, including those inside the Inception modules, use rectified linear activation.**

- The size of the receptive field in their network is 224x224 taking RGB color channels with **mean subtraction**.

Inception v1

- **All the convolutions, including those inside the Inception modules, use rectified linear activation.**

- The size of the receptive field in their network is 224x224 taking RGB color channels with **mean subtraction**.

- One can see the number of 1x1 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.**

Inception v1

- **It was found that a move from fully connected layers to average pooling improved the top-1 accuracy by about 0.6%.**

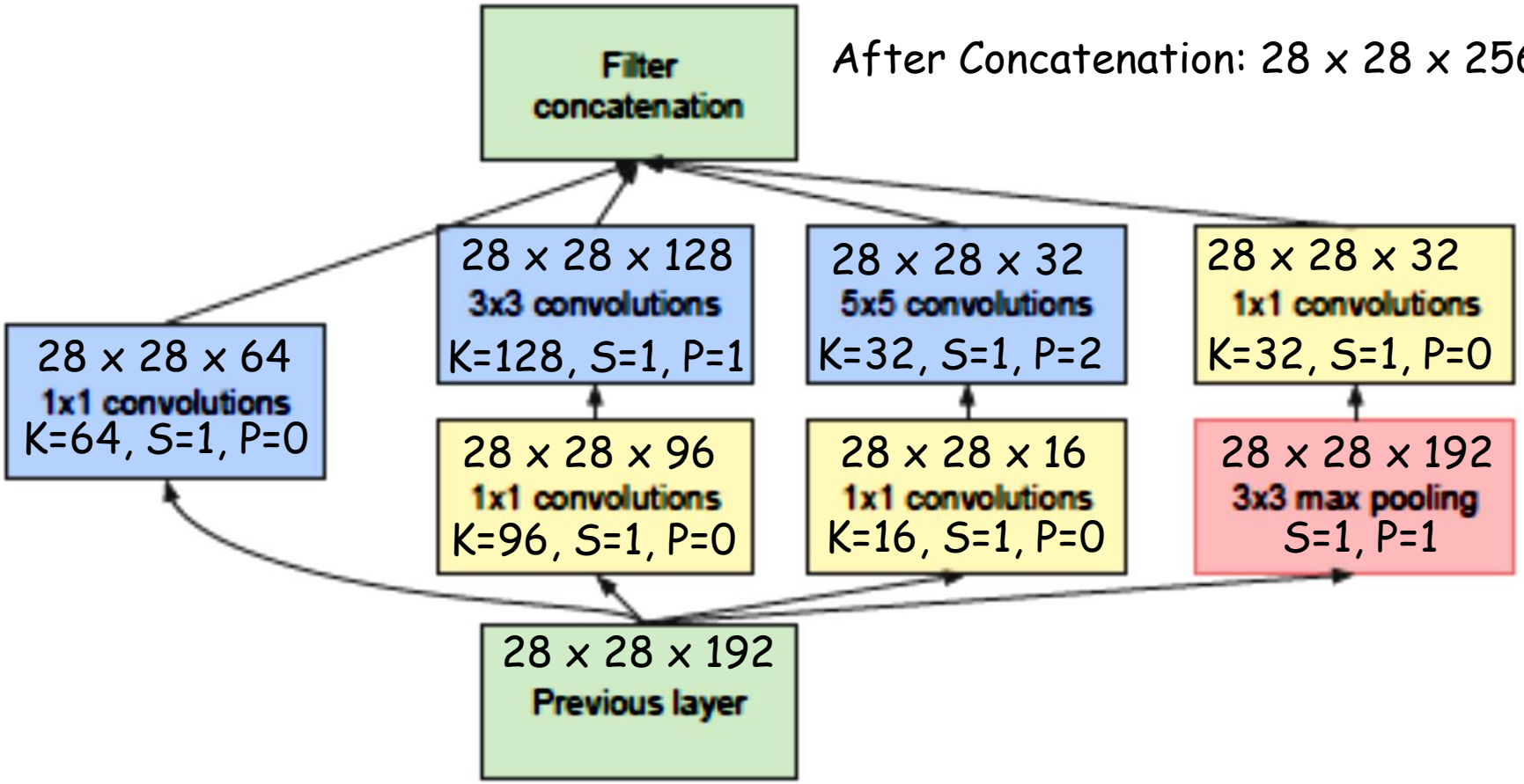
Inception v1

- **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.**

Inception v1 – Size Details Inception (3a)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M



(b) Inception module with dimension reductions

Inception v1 [4, 5]

- GoogLeNet has **9 such inception modules** stacked linearly.

Inception v1 [4, 5]

- 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.

Inception v1 [4, 5]

- 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**.

Inception v1 [4, 5]

- **To prevent the middle part of the network from "dying out", the authors introduced two auxiliary classifiers** (The purple boxes in the image). (**4a and 4d**)

Inception v1 [4, 5]

- **To prevent the middle part of the network from "dying out", the authors introduced two auxiliary classifiers** (The purple boxes in the image). (**4a and 4d**)

- **They essentially applied softmax to the outputs of two of the inception modules, and computed an auxiliary loss over the same labels.**

Inception v1 [4, 5]

- **To prevent the middle part of the network from "dying out", the authors introduced two auxiliary classifiers** (The purple boxes in the image). (**4a and 4d**)

- **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.

Inception v1 [4, 5]

- **To prevent the middle part of the network from "dying out", the authors introduced two auxiliary classifiers** (The purple boxes in the image). (**4a and 4d**)

- **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.

- **total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2.**

Inception v1 [4, 5]

- **To prevent the middle part of the network from "dying out", the authors introduced two auxiliary classifiers** (The purple boxes in the image). (**4a and 4d**)

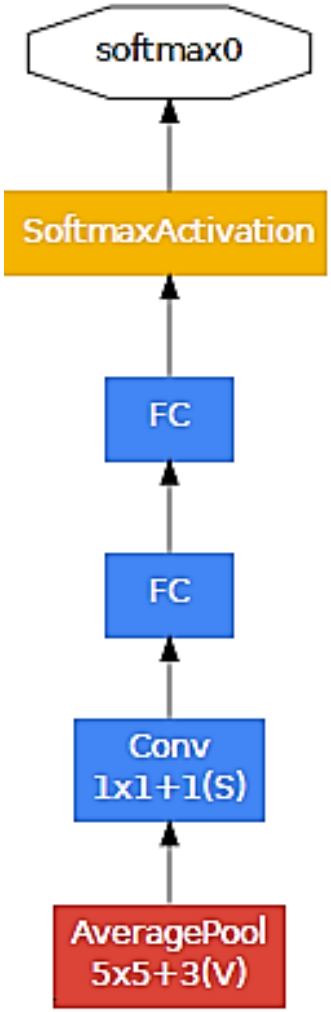
- **They essentially applied softmax to the outputs of two of the inception modules, and computed an auxiliary loss over the same labels.**

- 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.

- **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.

Inception v1 [5]



- An average pooling layer with 5x5 filter size and stride 3, resulting in an 4x4x512 output for the **(4a)**, and 4x4x528 for the **(4d)** stage.
- A 1x1 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 activation as the classifier (predicting the same 1000 classes as the main classifier, but removed at inference time).

Inception v1 [5]

- **Training Methodology**
  - Their image sampling methods were 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 was hard to give a definitive guidance to the most effective single way to train these networks.

Inception v1 [5]

- **ILSVRC 2014 Classification Challenge Setup and Results**
  - 1.
    - They independently trained 7 versions of the same GoogLeNet model (including one wider version), and performed ensemble prediction with them.

Inception v1 [5]

- **ILSVRC 2014 Classification Challenge Setup and Results**
  - 2.

    - **During testing, they adopted a more aggressive cropping approach than that of Krizhevsky et al.. Specifically, they resized the image to 4 scales where the shorter dimension (height or width) was 256, 288, 320 and 352 respectively, took the left, center and right square of these resized images (in the case of portrait images, they took the top, center and bottom squares).**

    - **For each square, they then took the 4 corners and the center 224x224 crop as well as the square resized to 224x224, and their mirrored versions.**

    - **This results in 4x3x6x2 = 144 crops per image.**

Inception v1 [5]

- **ILSVRC 2014 Classification Challenge Setup and Results**
  - 3.
    - **The softmax probabilities are averaged over multiple crops and over all the individual classifiers to obtain the final prediction.**

Number of models	Number of Crops	Cost	Top-5 error	compared to base
1	1	1	10.07%	base
1	10	10	9.15%	-0.92%
1	144	144	7.89%	-2.18%
7	1	7	8.09%	-1.98%
7	10	70	7.62%	-2.45%
7	144	1008	6.67%	-3.45%

Table 3: GoogLeNet classification performance break down

Inception v2 [4, 7]

- **Premise:**
  - **Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.**

Inception v2 [7]

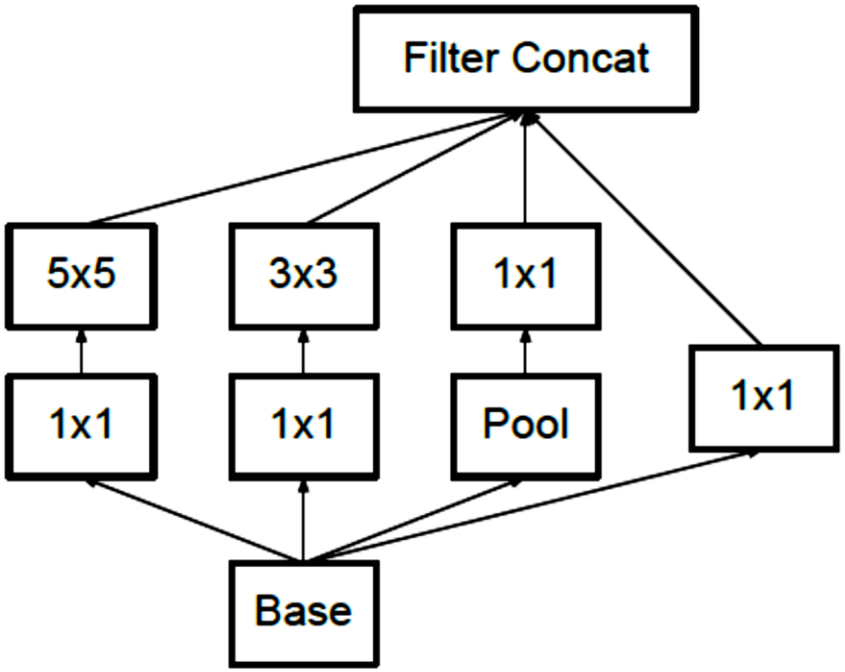


Figure 4. Original Inception module as described in [20].

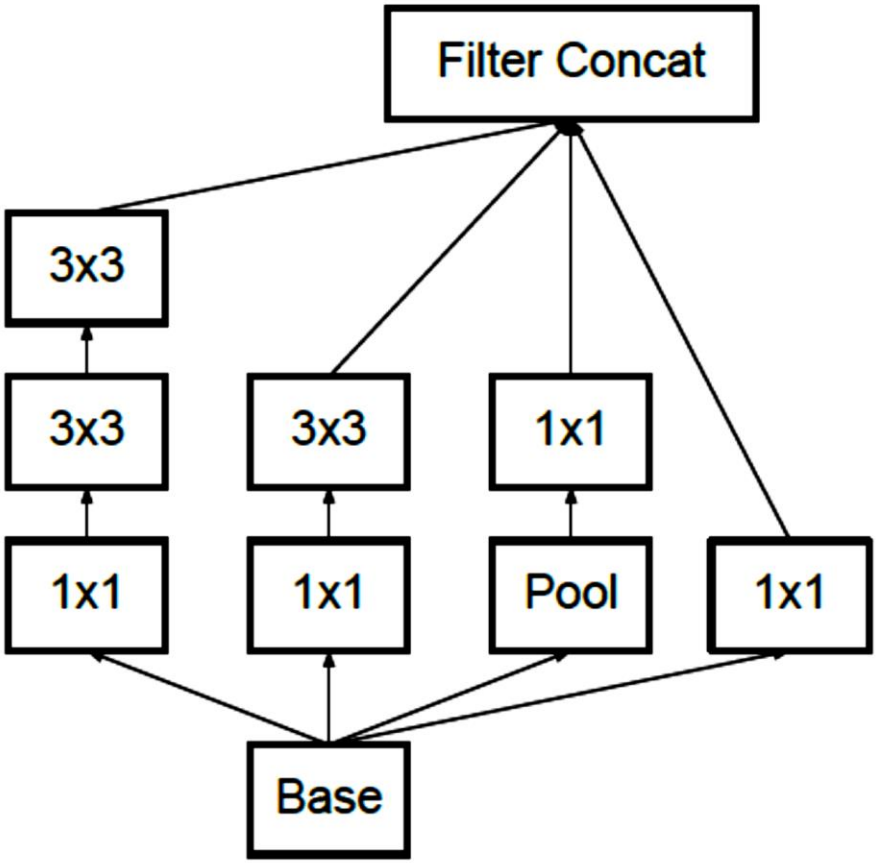


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

Input: 35 x 35 x 288, 3 such modules

Inception v2 [4, 7, 12]

- **Premise:**
  - Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.
    - Let's say we have a 5x5 input image and after a convolution, we want to produce a 5x5 output image (we will have to use padding).

Inception v2 [4, 7, 12]

- **Premise:**
  - Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.
    - Let's say we have a 5x5 input image and after a convolution, we want to produce a 5x5 output image (we will need to use padding).
    - The goal is to compare the number of operations between a 5x5 convolution and 2 stacked 3x3 convolutions.

Inception v2 [4, 7, 12]

- **Premise:**
  - Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.
    - Let's say we have a 5x5 input image and after a convolution, we want to produce a 5x5 output image (we will need to use padding).
    - The goal is to compare the number of operations between a 5x5 convolution and 2 stacked 3x3 convolutions.
    - With a 5x5 convolution, we need a padding of 2.
      - # of operations = 5x5x5x5 = 25x25.

Inception v2 [4, 7, 12]

- **Premise:**
  - Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.
    - Let's say we have a 5x5 input image and after a convolution, we want to produce a 5x5 output image (we will need to use padding).
    - The goal is to compare the number of operations between a 5x5 convolution and 2 stacked 3x3 convolutions.
    - With a 5x5 convolution, we need a padding of 2.
      - # of operations = 5x5x5x5 = 25x25.
    - With 2 stacked 3x3 convolution, we need a padding of 1 for each convolution.
      - # of operations = 5x5x3x3 + 5x5x3x3 = 25x9 + 25x9 = 25x18.

Inception v2 [4, 7, 12]

- **Premise:**
  - Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.
    - Let's say we have a 5x5 input image and after a convolution, we want to produce a 5x5 output image (we will need to use padding).
    - The goal is to compare the number of operations between a 5x5 convolution and 2 stacked 3x3 convolutions.
    - With a 5x5 convolution, we need a padding of 2.
      - # of operations = 5x5x5x5 = 25x25.
    - With 2 stacked 3x3 convolution, we need a padding of 1 for each convolution.
      - # of operations = 5x5x3x3 + 5x5x3x3 = 25x9 + 25x9 = 25x18.
    - Thus, 2 stacked 3x3 convolutions reduce the computation load by
      - (9+9)/25 that is ((25x18)/(25x25) = 18/25 = (9+9)/25) which result in a computation saving of 28% by this factorization ((1 - 18/25)*100 = 28%).

Inception v2 [4, 7, 12]

- **Premise:**
  - Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.
    - Let's say we have a 5x5 input image and after a convolution, we want to produce a 5x5 output image (we will need to use padding).
    - The goal is to compare the number of operations between a 5x5 convolution and 2 stacked 3x3 convolutions.
    - With a 5x5 convolution, we need a padding of 2.
      - # of operations = 5x5x5x5 = 25x25.
    - With 2 stacked 3x3 convolution, we need a padding of 1 for each convolution.
      - # of operations = 5x5x3x3 + 5x5x3x3 = 25x9 + 25x9 = 25x18.
    - Thus, 2 stacked 3x3 convolutions reduce the computation load by
      - (9+9)/25 that is ((25x18)/(25x25) = 18/25 = (9+9)/25) which result in a computation saving of 28% by this factorization ((1 - 18/25)*100 = 28%).
    - Moreover, it is worth noticing that 2 stacked 3x3 filters give the same receptive field as as a 5x5 filter.

Inception v2 [7]

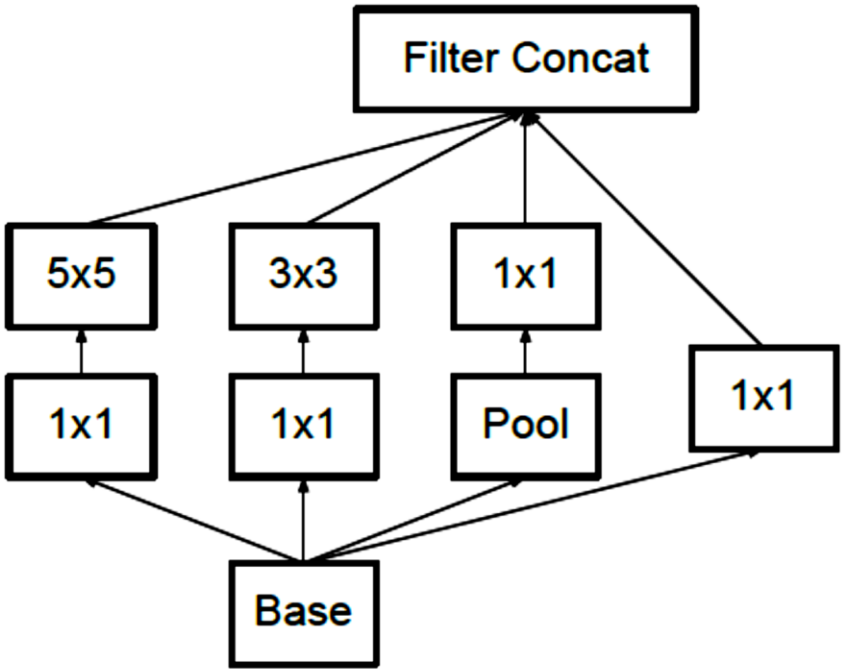


Figure 4. Original Inception module as described in [20].

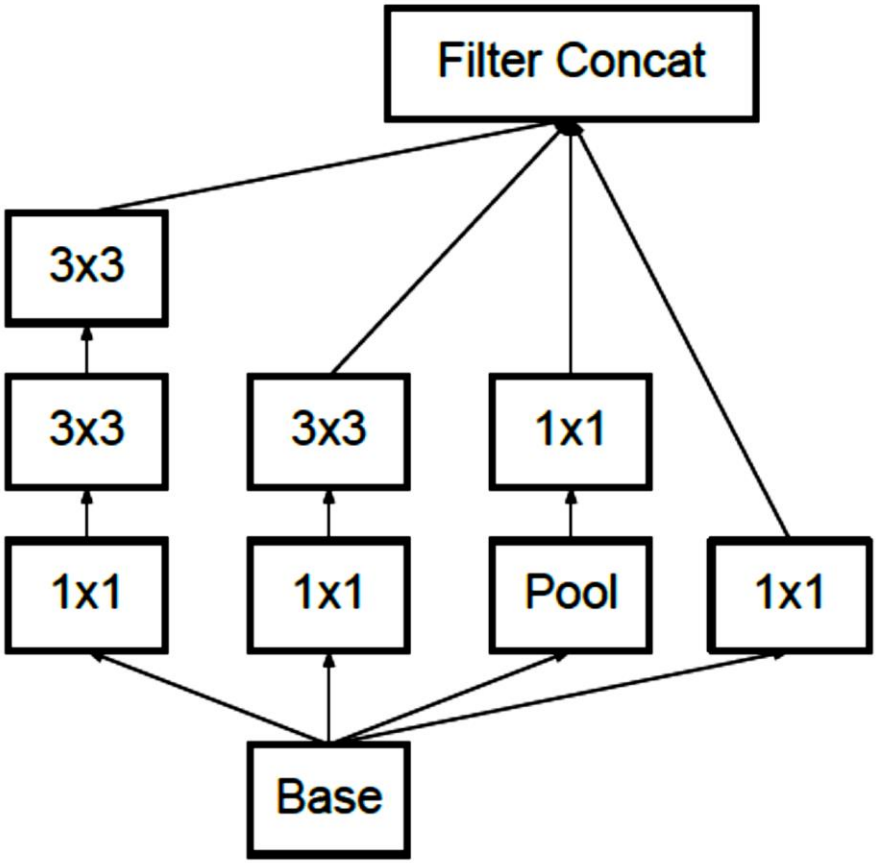


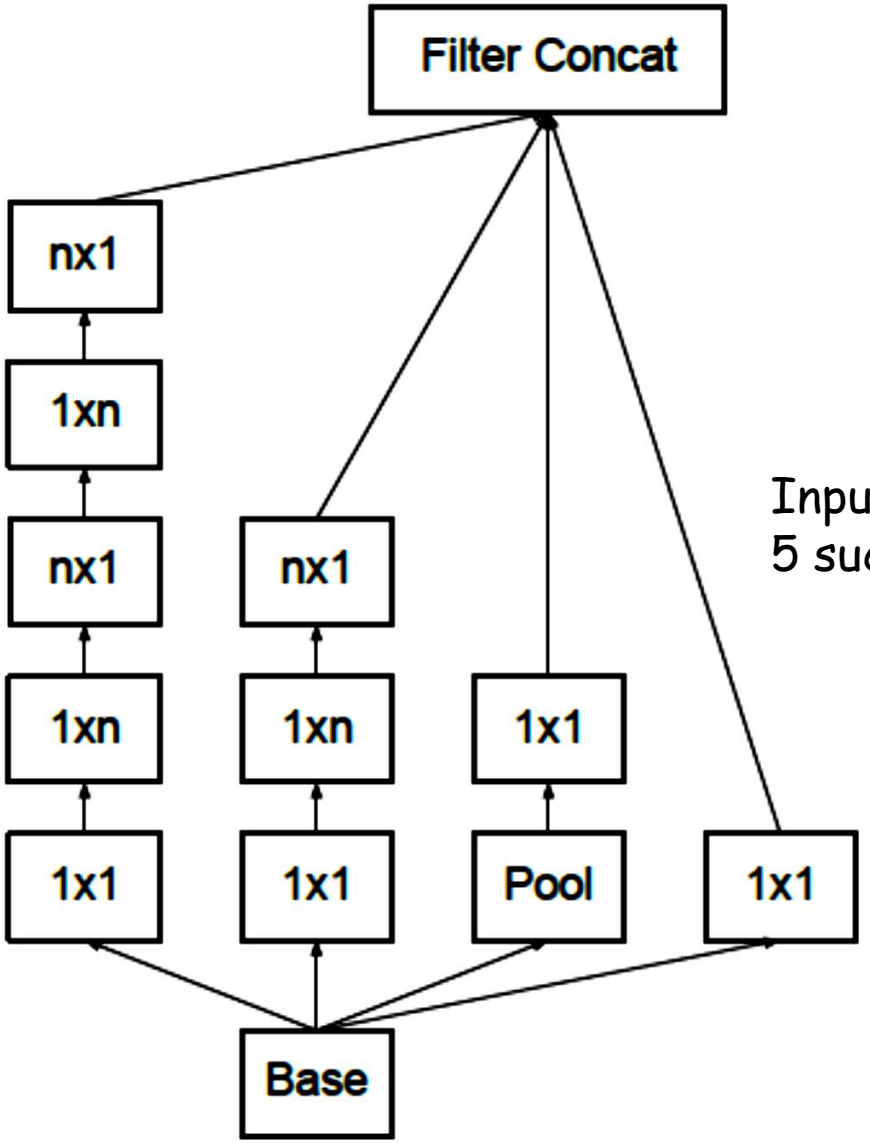
Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

Input: 35 x 35 x 288, 3 such modules

Inception v2 [4, 7]

- **Premise:**
  - **Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.**

Inception v2 [7]



Input: 17 x 17 x 768,
5 such modules

Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

Inception v2 [4, 7, 12]

- **Premise:**
  - **Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.**
    - They factorize n x n convolution into a combination of 1 x n convolution and n x 1 convolution.

Inception v2 [4, 7, 12]

- **Premise:**
  - **Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.**
    - They factorize n x n convolution into a combination of 1 x n convolution and n x 1 convolution.
    - They call it "asymmetric convolution". For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution, and then performing a 3x1 convolution on its output.

Inception v2 [4, 7, 12]

- **Premise:**
  - **Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.**
    - They factorize n x n convolution into a combination of 1 x n convolution and n x 1 convolution.
    - They call it "asymmetric convolution". For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution, and then performing a 3x1 convolution on its output.
    - They found it to be 33% cheaper than a single 3x3 convolution.

Inception v2 [4, 7, 12]

- **Premise:**
  - **Using smart factorization methods, convolutions can be made more efficient in terms of computational complexity.**
    - They factorize n x n convolution into a combination of 1 x n convolution and n x 1 convolution.
    - They call it "asymmetric convolution". For example, a 3x3 convolution is equivalent to first performing a 1x3 convolution, and then performing a 3x1 convolution on its output.
    - They found it to be 33% cheaper than a single 3x3 convolution.
    - Same experience as above but this time with a 3x3 input/output image:
      - with a 3x3 convolution, we need a padding of 1:
        - # of operations = 3x3x3x3 = 81.
      - with combination of 1x3 / 3x1 convolutions, we need a padding of 1 for each convolution:
        - # of operations = 3x3x1x3 + 3x3x3x1 = 54.
      - Thus, a combination of 1x3 and 3x1 convolutions result in a computation saving of 33%. That is ((1 - (54/81))*100 = 33%) by this factorization.

Inception v2 [7]

Input: 8 x 8 x 1280,
2 such modules

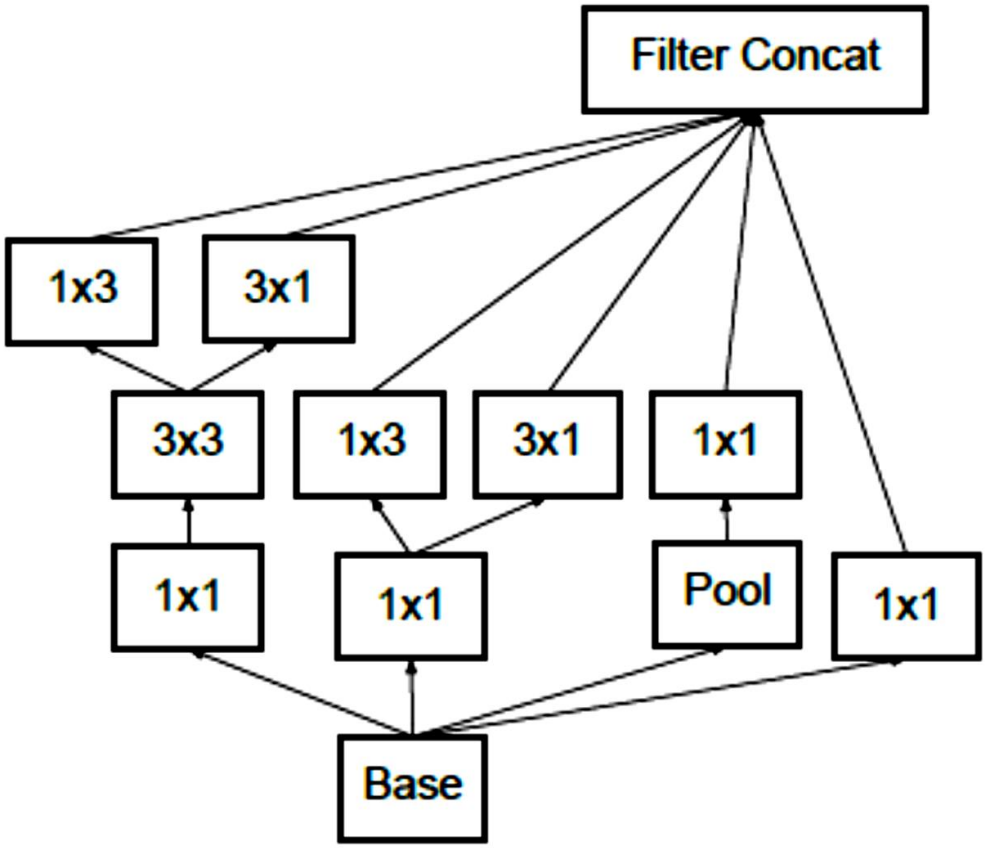


Figure 7. Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations, as suggested by principle 2 of Section 2. We are using this solution only on the coarsest grid, since that is the place where producing 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.

Inception v2 [7]

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

Note: In Inception v2, instead of first three layers as shown above, they had 1 layer with F=7 and S=2 just like in Googlenet/Inception v1.

Inception v3 [4]

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.**

Inception v3 [4]

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.**

Inception v3 [4]

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).

Inception v2 & v3 [7]

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized 7×7	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8

Table 3. Single crop experimental results comparing the cumulative effects on the various contributing factors. We compare our numbers with the best published single-crop inference for Ioffe at al [7]. For the "Inception-v2" lines, the changes are cumulative and each subsequent line includes the new change in addition to the previous ones. The last line is referring to all the changes is what we refer to as "Inception-v3" below. Unfortunately, He et al [6] reports the only 10-crop evaluation results, but not single crop results, which is reported in the Table 4 below.

Inception v2 & v3 [7]

Network	Crops Evaluated	Top-5 Error	Top-1 Error
GoogLeNet [20]	10	-	9.15%
GoogLeNet [20]	144	-	7.89%
VGG [18]	-	24.4%	6.8%
BN-Inception [7]	144	22%	5.82%
PReLU [6]	10	24.27%	7.38%
PReLU [6]	-	21.59%	5.71%
Inception-v3	12	19.47%	4.48%
Inception-v3	144	**18.77%**	**4.2%**

Table 4. Single-model, multi-crop experimental results comparing the cumulative effects on the various contributing factors. We compare our numbers with the best published single-model inference results on the ILSVRC 2012 classification benchmark.

Inception v2 & v3 [7]

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	**17.2%**	**3.58%***

Table 5. Ensemble evaluation results comparing multi-model, multi-crop reported results. Our numbers are compared with the best published ensemble inference results on the ILSVRC 2012 classification benchmark. *All results, but the top-5 ensemble result reported are on the validation set. The ensemble yielded 3.46% top-5 error on the validation set.

ResNet [8, 10]

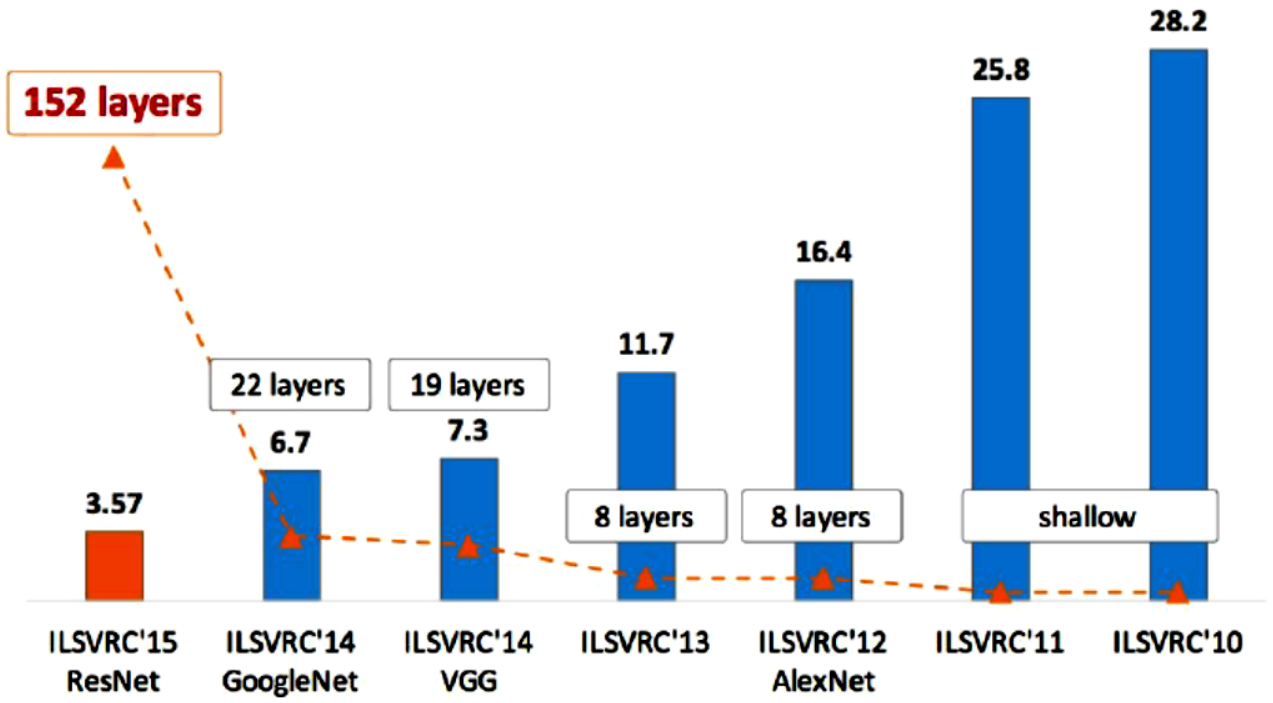


Figure copyright Kaiming He, 2016. Reproduced with permission.

ResNet [8, 10]

- **Is learning better networks as easy as stacking more layers?**

ResNet [8, 10]

- An obstacle to answering this question was the notorious problem of **vanishing/exploding gradients** "[1, 9]", which hamper convergence from the beginning.

ResNet [8, 10]

- An obstacle to answering this question was the notorious problem of **vanishing/exploding gradients** "[1, 9]", which hamper convergence from the beginning.

- This problem, however, has been largely addressed by **normalized initialization** "[23, 9, 37, 13]" and **intermediate normalization layers** "[16]", which enable networks with **tens of layers** to start converging for stochastic gradient descent (SGD) with backpropagation "[22]".

ResNet [8, 10]

- When deeper networks are able to start converging, a **degradation problem** has been exposed: **with the network depth increasing, accuracy gets saturated and then degrades rapidly**.

ResNet [8, 10]

- When deeper networks are able to start converging, a **degradation problem** has been exposed: **with the network depth increasing, accuracy gets saturated and then degrades rapidly**.

- Unexpectedly, such degradation **is not caused by overfitting**, and adding more layers to a suitably deep model leads to higher training error, as reported in "[11, 42]" and thoroughly verified by our experiments.

ResNet [8, 10]

- When deeper networks are able to start converging, a **degradation problem** has been exposed: **with the network depth increasing, accuracy gets saturated and then degrades rapidly**.

- Unexpectedly, such degradation **is not caused by overfitting**, and adding more layers to a suitably deep model leads to higher training error, as reported in "[11, 42]" and thoroughly verified by their experiments.

- Fig. 1 shows a typical example.

ResNet [8, 10]

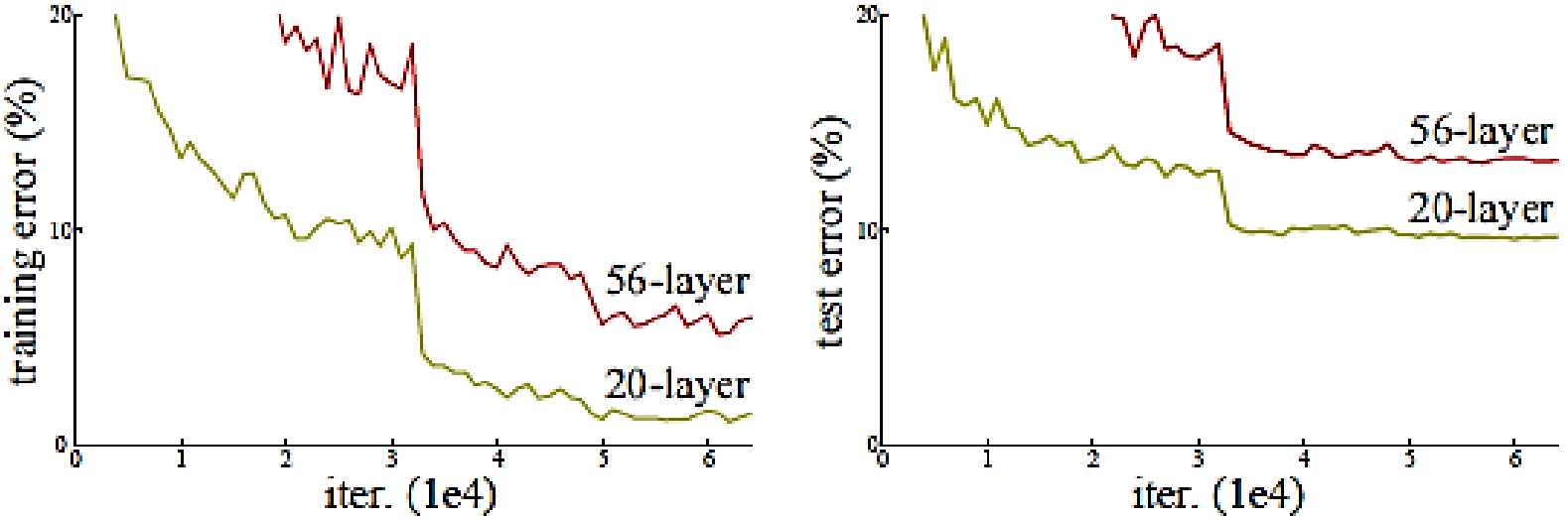
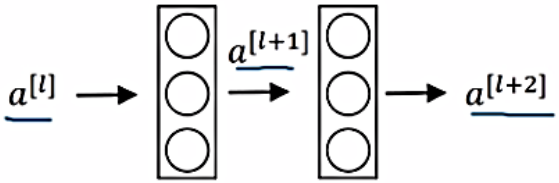
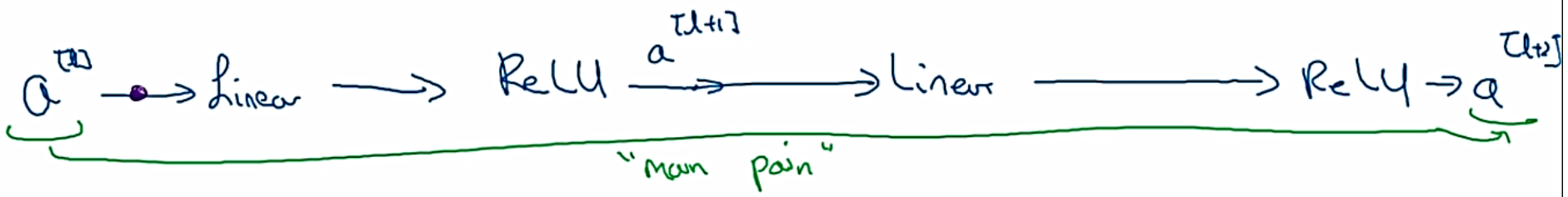


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error.

ResNet [8, 10]

Residual block



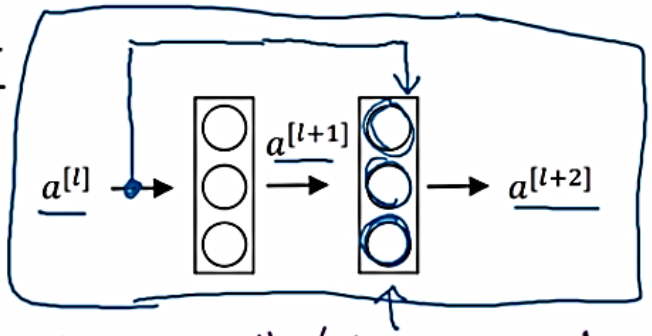


$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \qquad a^{[l+1]} = g(z^{[l+1]}) \qquad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]} \qquad a^{[l+2]} = g(z^{[l+2]})$$

Source: Andrew Ng's Lecture on ResNet

ResNet [8, 10]

Residual block



$a^{[l]}$ → Linear → ReLU → $a^{[l+1]}$ → Linear → ReLU → $a^{[l+2]}$

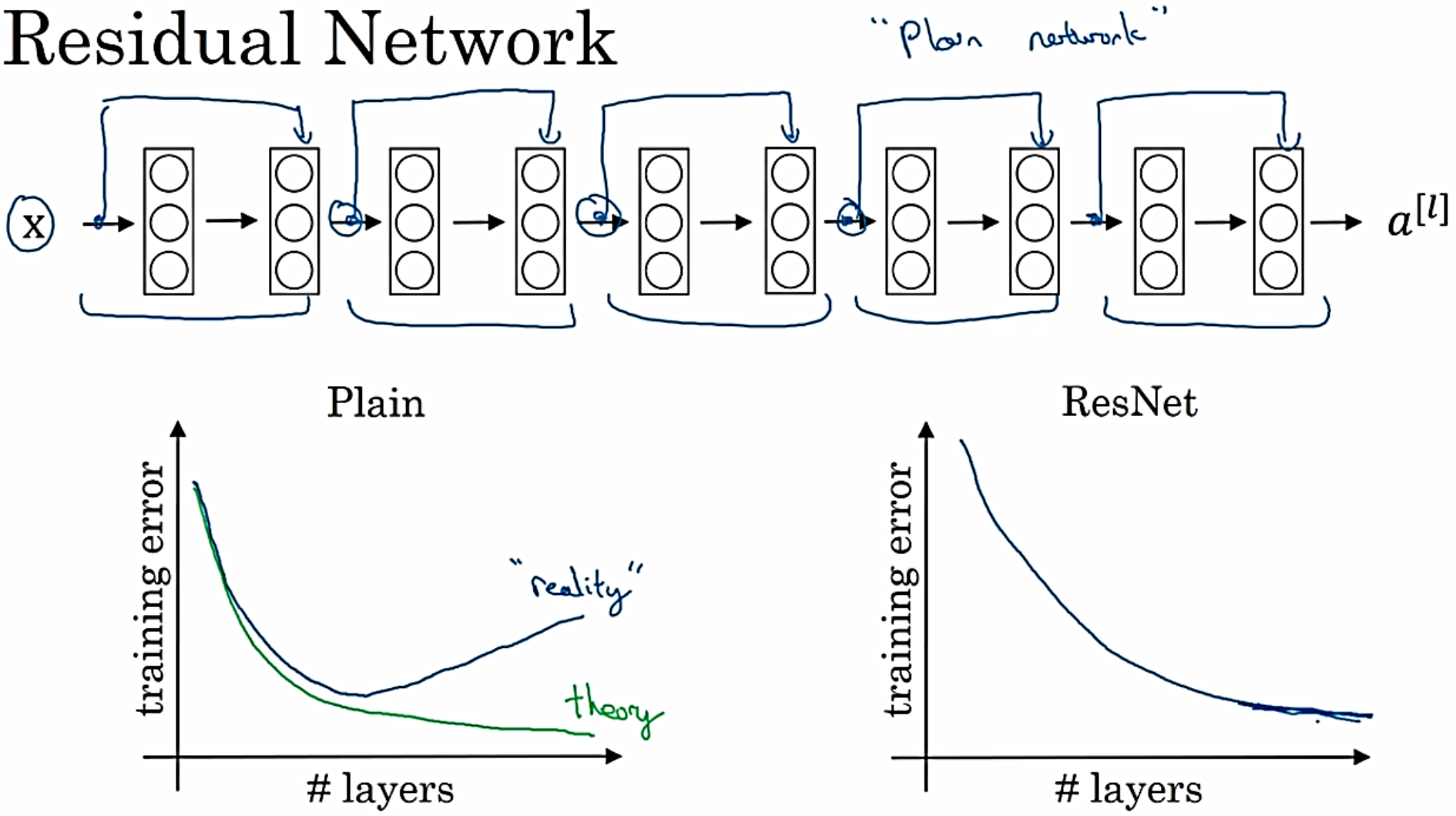
"main pain"

$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]} \qquad a^{[l+1]} = g(z^{[l+1]}) \qquad z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]} \qquad \cancel{a^{[l+2]} = g(z^{[l+2]})}$$

$$a^{[l+2]} = g\left(z^{[l+2]} + a^{[l]}\right)$$

Source: Andrew Ng's Lecture on ResNet

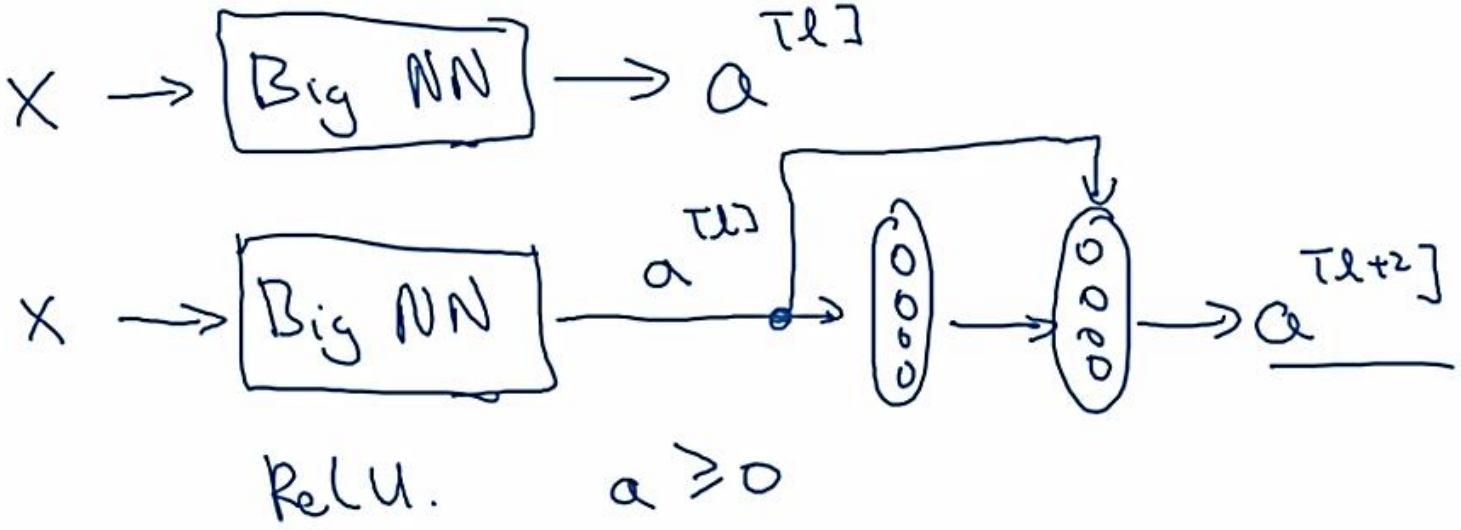
ResNet [8, 10]



Source: Andrew Ng's Lecture on ResNet

ResNet [8, 10]

Why do residual networks work?



ReLU. $a \geq 0$

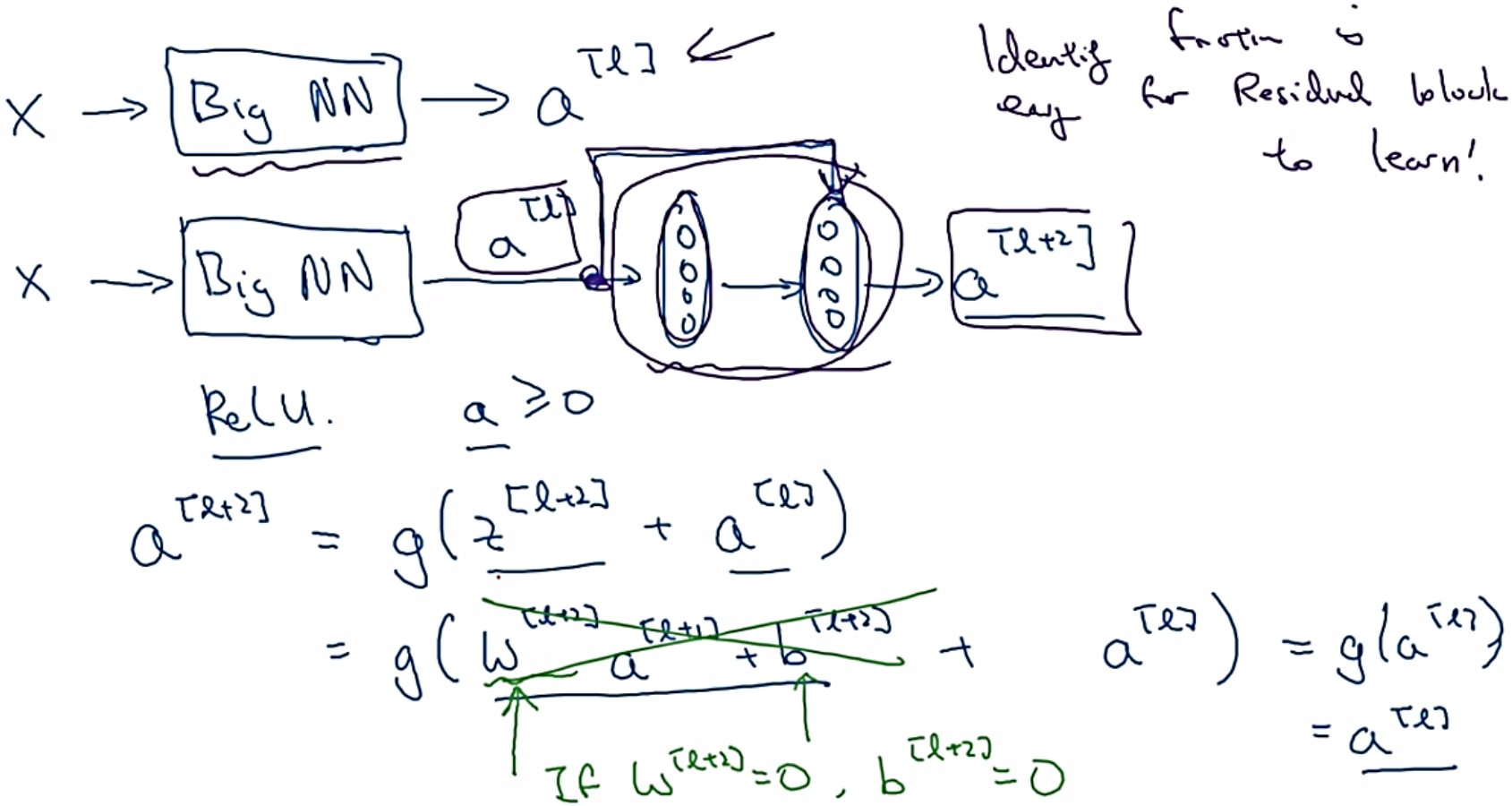
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

$$= g(w^{[l+2]} a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

Source: Andrew Ng's Lecture on ResNet

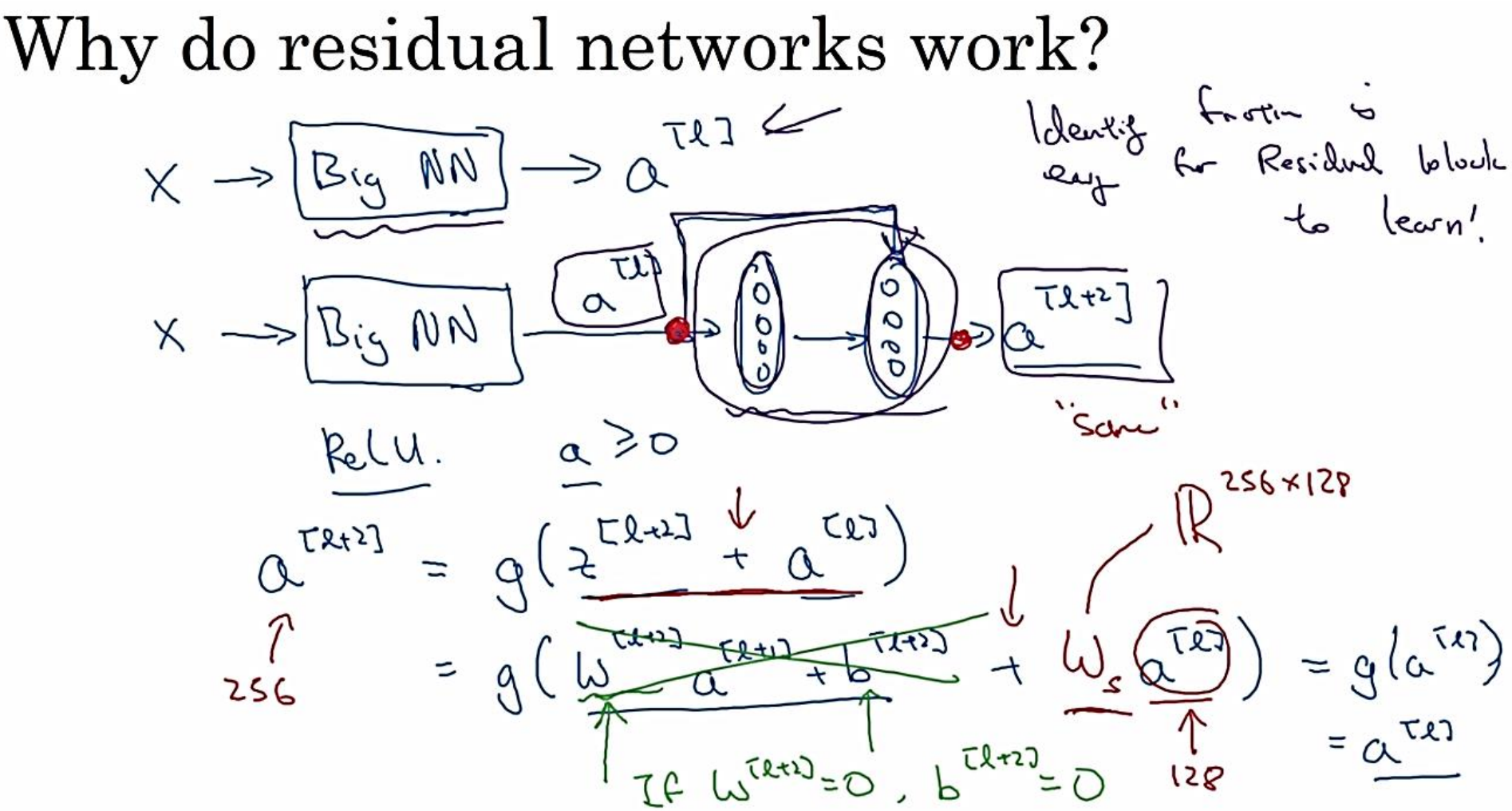
ResNet [8, 10]

Why do residual networks work?



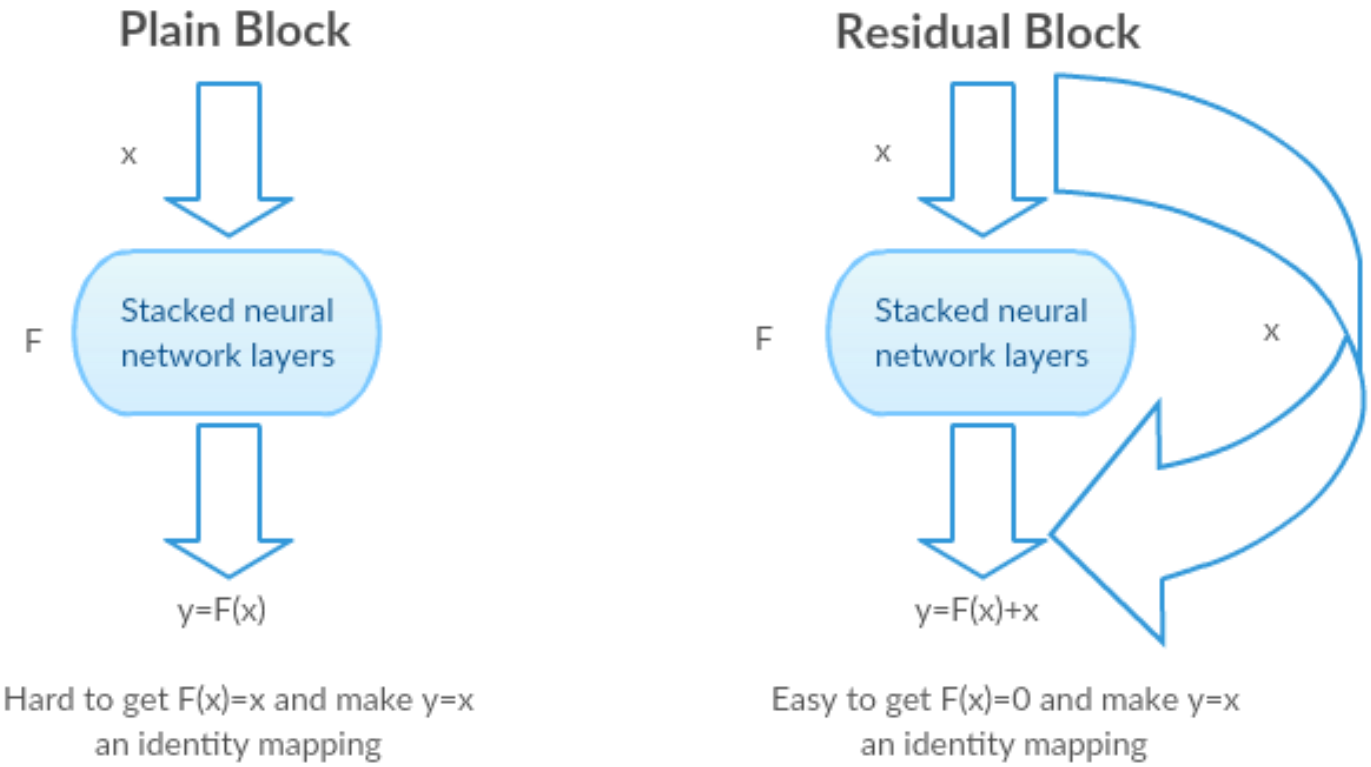
Source: Andrew Ng's Lecture on ResNet

ResNet [8, 10]



Source: Andrew Ng's Lecture on ResNet

ResNet [11]



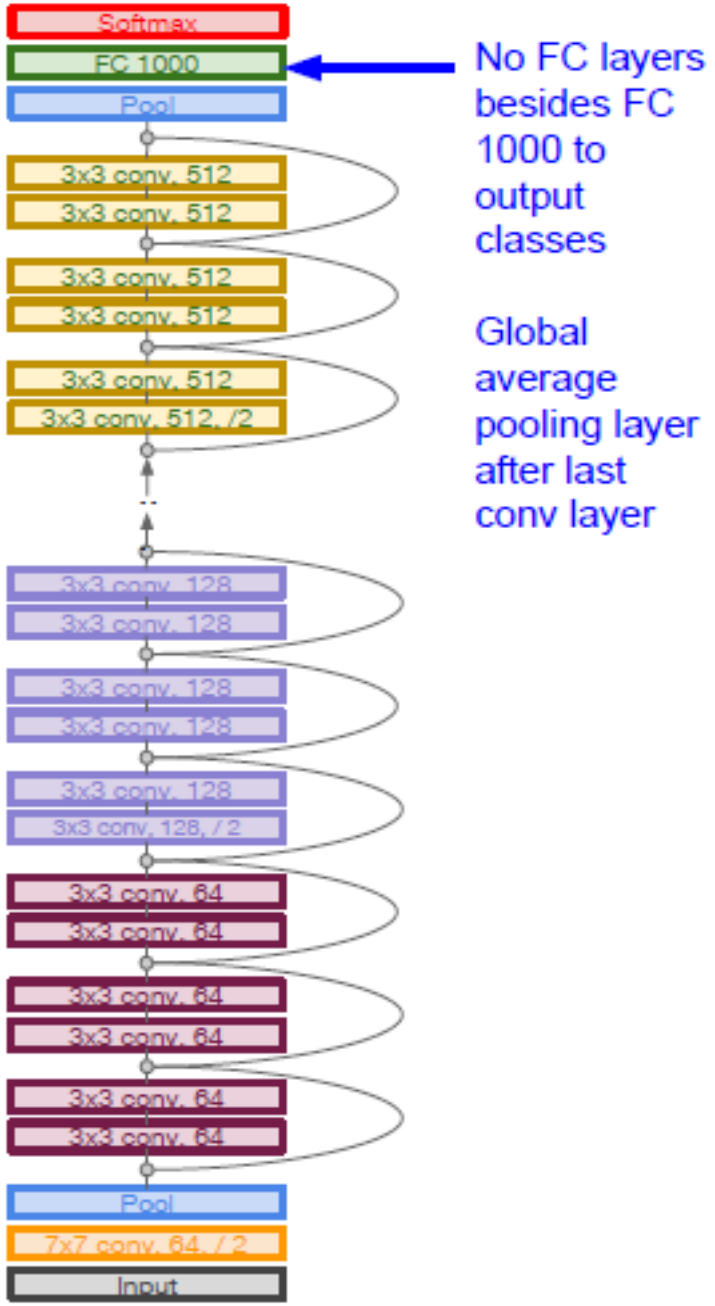
Hard to get F(x)=x and make y=x an identity mapping

Easy to get F(x)=0 and make y=x an identity mapping

ResNet [8, 10]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



ResNet [8, 10]

Identity Shortcut

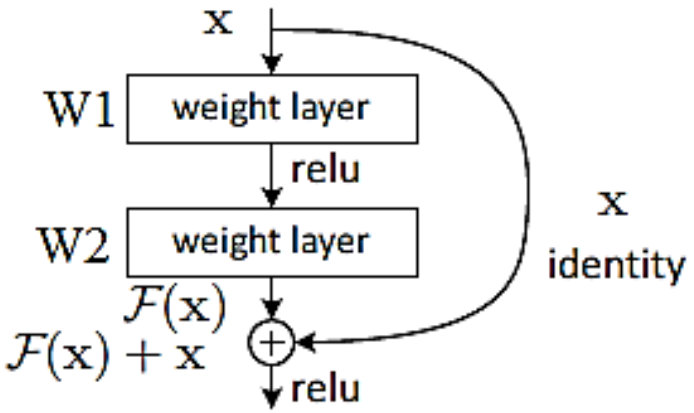


Figure 2. Residual learning: a building block.

$$\mathrm{y} = \mathcal{F}(\mathrm{x}, \{W_i\}) + \mathrm{x}. \quad (1)$$

$$\mathcal{F} = W_2\sigma(W_1\mathrm{x})$$

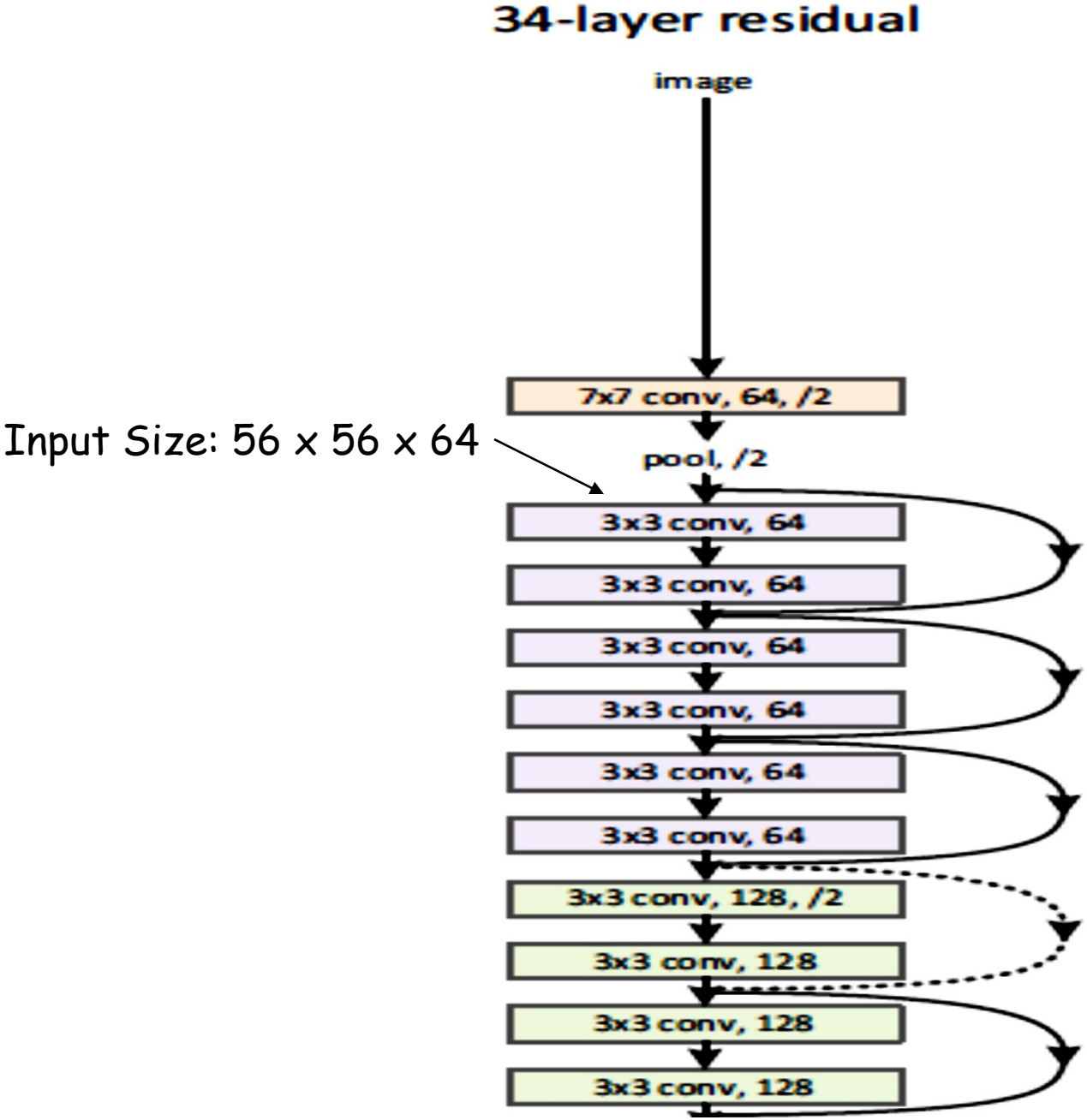
Sigma is Relu

y is the net input before the final ReLU

Projection Shortcut

$$\mathrm{y} = \mathcal{F}(\mathrm{x}, \{W_i\}) + W_s\mathrm{x}. \quad (2)$$

ResNet [8, 10]



Input Size: 56 x 56 x 64

ResNet [8, 10]

- When dimensionality of x and F(x) are same, identity shortcut is used (E.g. Eq. 1).

ResNet [8, 10]

- When dimensionality of x and F(x) are same, identity shortcut is used (E.g. Eq. 1).

- When they differ (dotted lines in the architecture), we have two options:
  - (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter

ResNet [8, 10]

- When dimensionality of x and F(x) are same, identity shortcut is used (E.g. Eq. 1).

- When they differ (dotted lines in the architecture), we have two options:
  - (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1x1 convolutions).

ResNet [8, 10]

- When dimensionality of x and F(x) are same, identity shortcut is used (E.g. Eq. 1).

- When they differ (dotted lines in the architecture), we have two options:
  - (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1x1 convolutions).

  - For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - First convert x from 56 x 56 x 64 to 28 x 28 x 64 using let's say pooling of 2 x 2 with stride 2. Now create 64 more channels of spatial extent 28 x 28 with all zeros and concatenate to have 28 x 28 x 128.

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - First convert x from 56 x 56 x 64 to 28 x 28 x 64 using let's say pooling of 1 x 1 with stride 2. Now create 64 more channels of spatial extent 28 x 28 with all zeros and concatenate to have 28 x 28 x 128.

- (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1x1 convolutions).

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - First convert x from 56 x 56 x 64 to 28 x 28 x 64 using let's say pooling of 1 x 1 with stride 2. Now create 64 more channels of spatial extent 28 x 28 with all zeros and concatenate to have 28 x 28 x 128.

- (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1x1 convolutions).
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - First convert x from 56 x 56 x 64 to 28 x 28 x 64 using let's say pooling of 1 x 1 with stride 2. Now create 64 more channels of spatial extent 28 x 28 with all zeros and concatenate to have 28 x 28 x 128.

- (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1x1 convolutions).
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - Use 128 convolution filters of size 1 x 1 x 64.

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - First convert x from 56 x 56 x 64 to 28 x 28 x 64 using let's say pooling of 1 x 1 with stride 2. Now create 64 more channels of spatial extent 28 x 28 with all zeros and concatenate to have 28 x 28 x 128.

- (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1x1 convolutions).
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - Use 128 convolution filters of size 1 x 1 x 64.
  - So these approach (e.g. (B)), introduces extra parameters.

ResNet [8, 10]

- (A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - First convert x from 56 x 56 x 64 to 28 x 28 x 64 using let's say pooling of 1 x 1 with stride 2. Now create 64 more channels of spatial extent 28 x 28 with all zeros and concatenate to have 28 x 28 x 128.

- (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1x1 convolutions).
  - E.g. X: 56 x 56 x 64 and F(X): 28 x 28 x 128 (4^{th} Residual Block)
  - Use 128 convolution filters of size 1 x 1 x 64.
  - So these approach (e.g. (B)), introduces extra parameters.

- For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

ResNet [8, 10]

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

ResNet [8, 10]

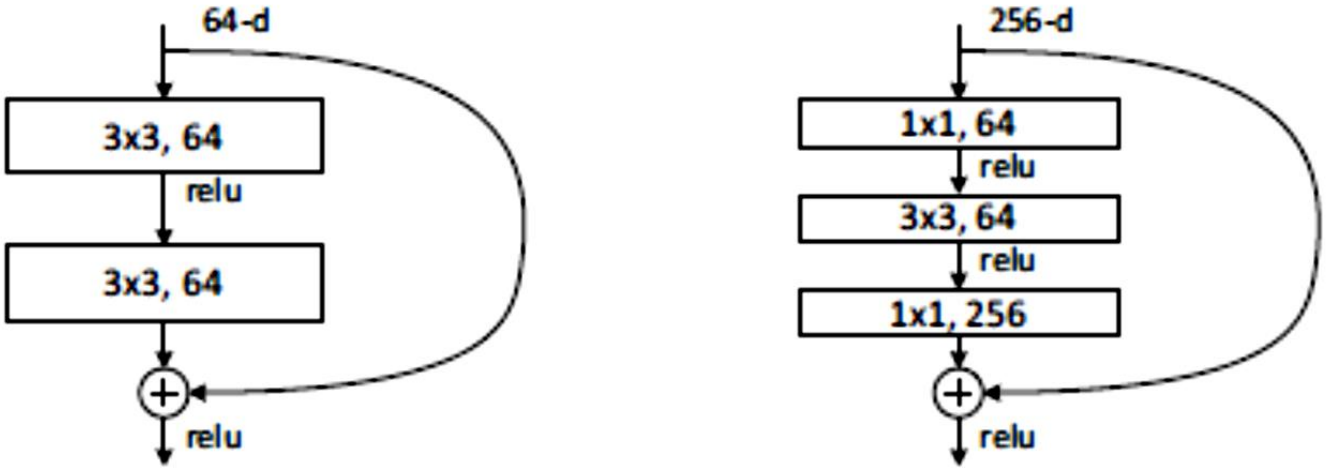


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

ResNet [8, 10]

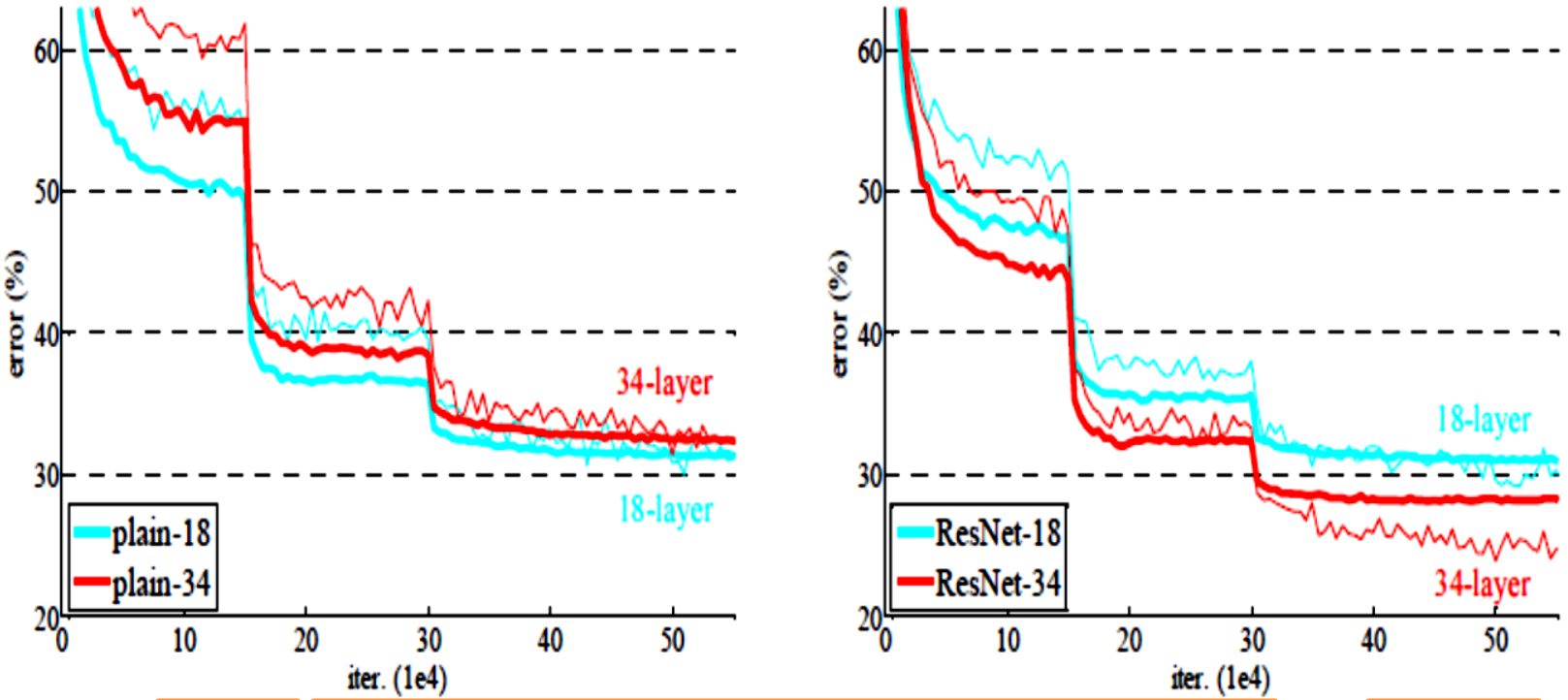


Figure 4. Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

ResNet [8, 10]

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

ResNet [8, 10]

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

In Table 3 we compare three options: (A) zero-padding shortcuts are used for increasing dimensions, and all shortcuts are parameter free (the same as Table 2 and Fig. 4 right); (B) projection shortcuts are used for increasing dimensions, and other shortcuts are identity; and (C) all shortcuts are projections.

ResNet [8, 10]

method	top-5 err. (test)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

(only with two 152-layer ones at the time of submitting)

ResNet [8, 10]

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Table 6. Classification error on the **CIFAR-10** test set. All methods are with data augmentation. For ResNet-110, we run it 5 times and show "best (mean±std)" as in [43].

References

1. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

2. cvml.ist.ac.at/courses/DLWT_W17/material/AlexNet.pdf

3. https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637

4. https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

5. Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

References

6. https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

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

8. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture9.pdf

9. https://stackoverflow.com/questions/43382499/reducing-filter-size-in-convolutional-neural-network

References

10. He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

11. https://medium.com/@14prakash/understanding-and-implementing-architectures-of-resnet-and-resnext-for-state-of-the-art-image-cf51669e1624

12. https://hackmd.io/@bouteille/SkD5Xd4DL

Disclaimer

- These slides are not original and have been prepared from various sources for teaching purpose.