Recap

What we have discussed last week …

How can neural networks classify images?

Recap

What we have discussed last week …

How can neural networks classify images?

→Initial idea: Let’s use vanilla fully connected neural networks (FCNN)

Steps: Vectorize the image -> Apply an FCNN

Recap

What we have discussed last week …

How can neural networks classify images?

→Initial idea: Let’s use vanilla fully connected neural networks (FCNN)

Steps: Vectorize the image -> Apply an FCNN

Why is this a bad idea?

Recap

What we have discussed last week …

How can neural networks classify images?

→Initial idea: Let's use vanilla fully connected neural networks (FCNN)

Steps: Vectorize the image -> Apply an FCNN

Why is this a bad idea?

Many parameters

-> Hard to fit on a GPU

-> High risk of overfitting

No translation invariance

Recap

What we have discussed last week …

Example:

Assume that we have an RGB image of size 500x500

→ Vectorization leads to an 750k dim-vector.

Let’s say we have Linear Layer with 1000 neurons.

<u>Question:</u>

- How many weight params does the layer have?
- How much memory does these weight params consume? (float32)

Recap

What we have discussed last week …

Example:

Assume that we have an RGB image of size 500x500 px

→ Vectorization leads to an 750k dim-vector.

Let’s say we have Linear Layer with 1000 neurons.

Only the first layer!

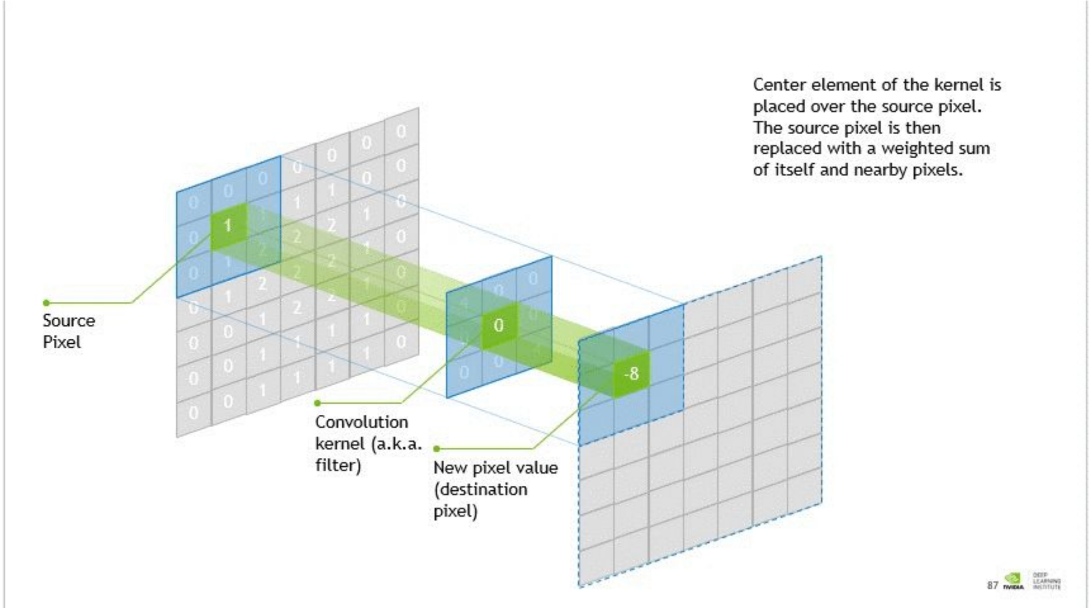
<u>Question:</u>

- How many weight params does the layer have? [750M weight params]
- How much memory does these weight params consume? [~2.79 GB memory]

Recap

What we have discussed last week …

We need something else …. **Convolutions**

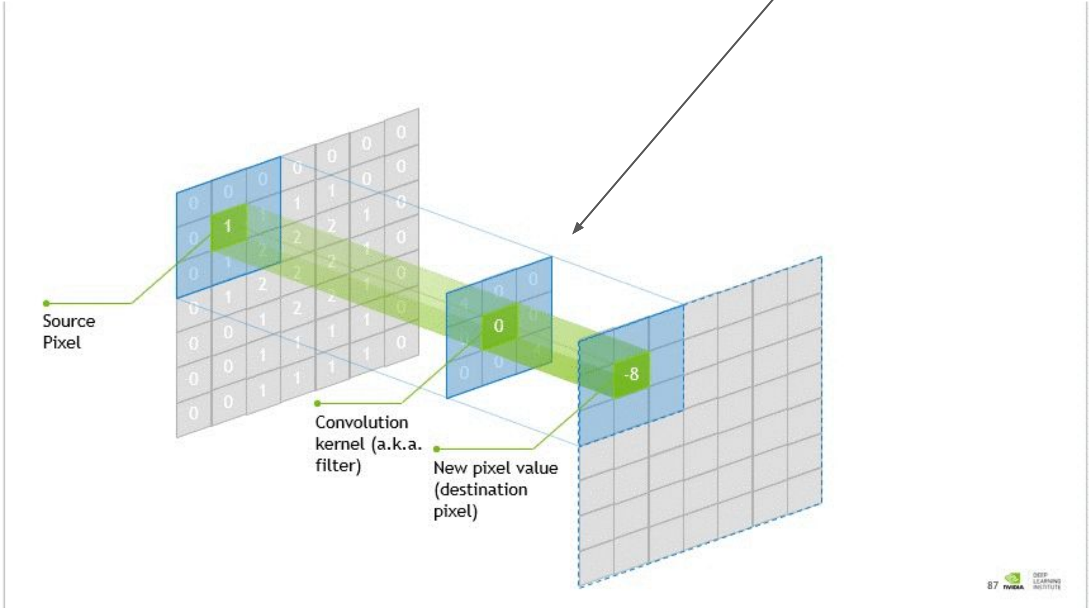


Recap

What we have discussed last week …

We need something else …. **Convolutions**

Weights are shared
(= reused at each positions)

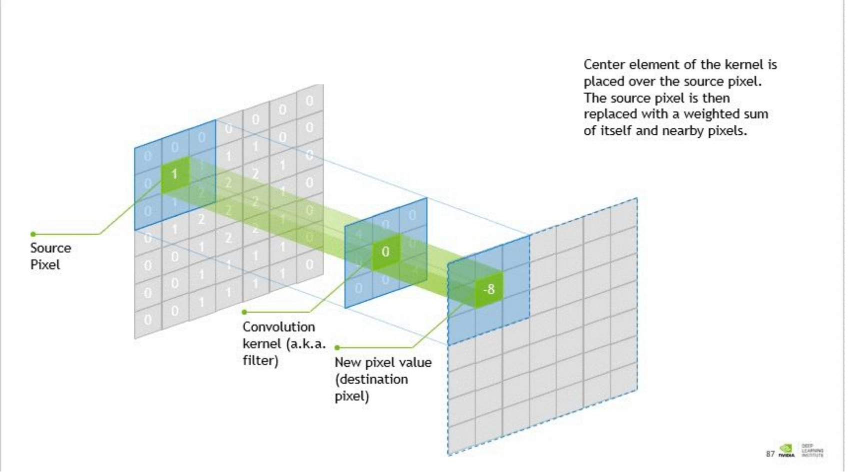


Recap

What we have discussed last week …

Again, assume that we have an RGB image of size 500x500 px.
We apply a convolution with a 3x3 kernel.

Question:
How many parameters (weights + bias) does the kernel have?



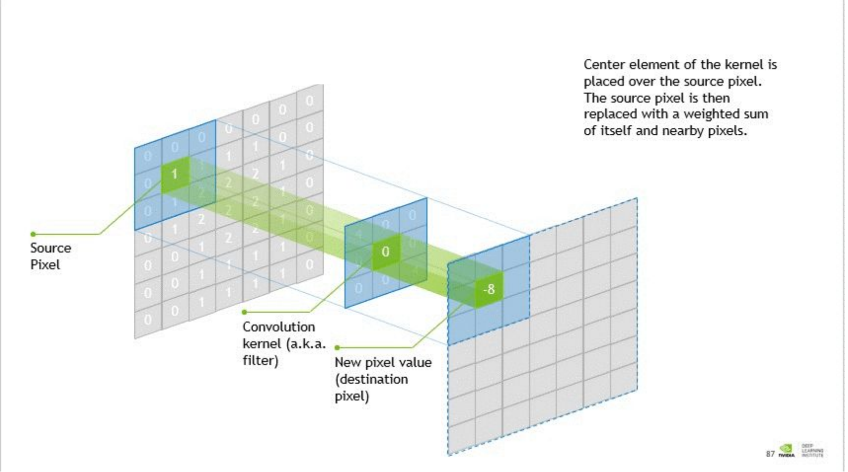
Recap

What we have discussed last week …

Again, assume that we have an RGB image of size 500x500 px.
We apply a convolution with a single 3x3 kernel.

Question:
How many parameters (weights + bias) does the kernel have?

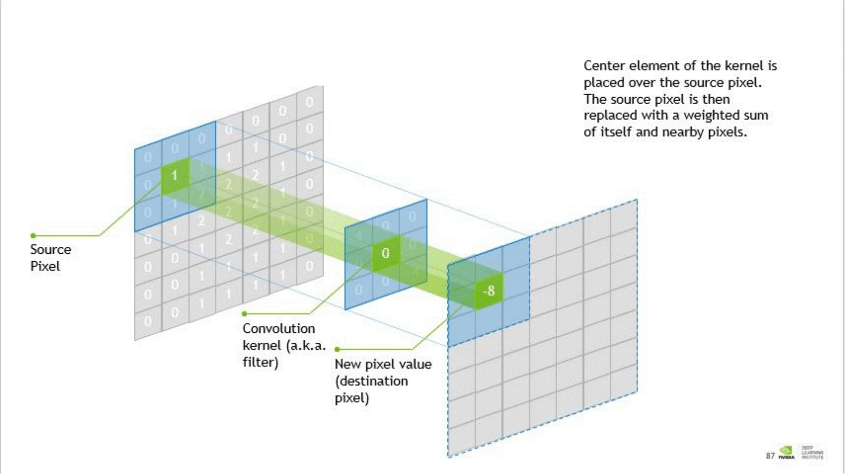
[27 weights + 1 bias = 28 learnable params]



Recap
What we have discussed last week …

Again, assume that we have an RGB image of size 500x500 px.
We apply a convolution with a single 3x3 kernel.

Question:
How many params has a layer with
100 kernels?

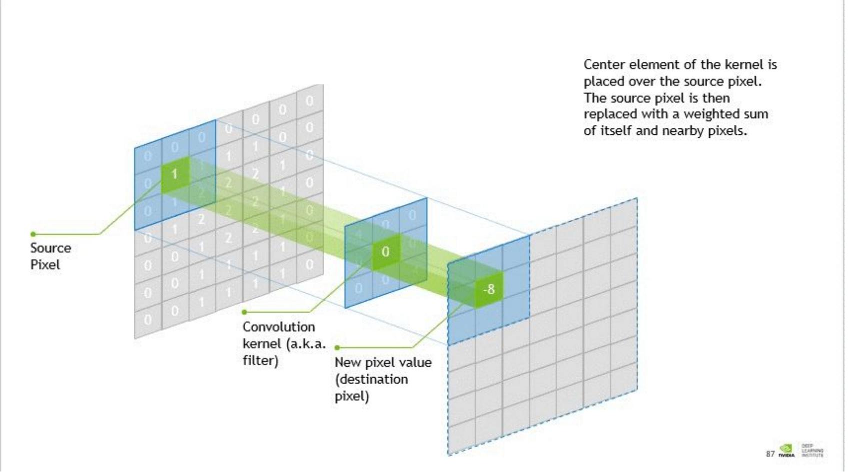


Recap
What we have discussed last week …

Again, assume that we have an RGB image of size 500x500 px.
We apply a convolution with a single 3x3 kernel.

Question:
How many params has a layer with
100 kernels?

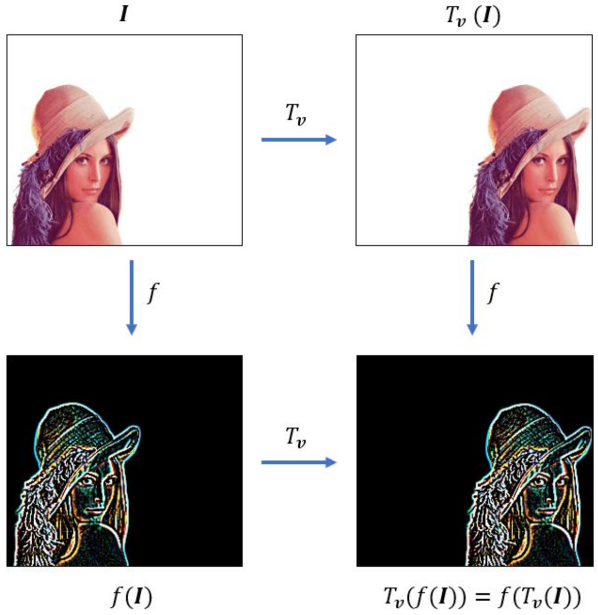
[28 params * 100 = 2800 learnable params]



Recap

What we have discussed last week …

However, by re-applying the same kernel at different image locations we also obtain **translation invariance.**



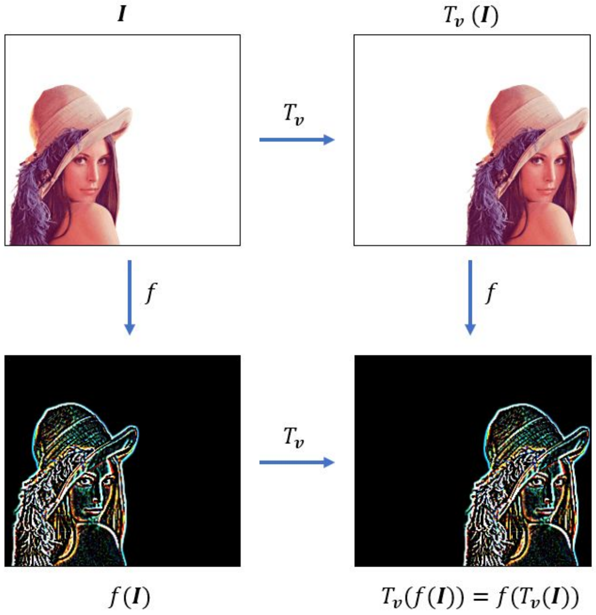
Recap

What we have discussed last week …

However, by re-applying the same kernel at different image locations we also obtain **translation invariance.**

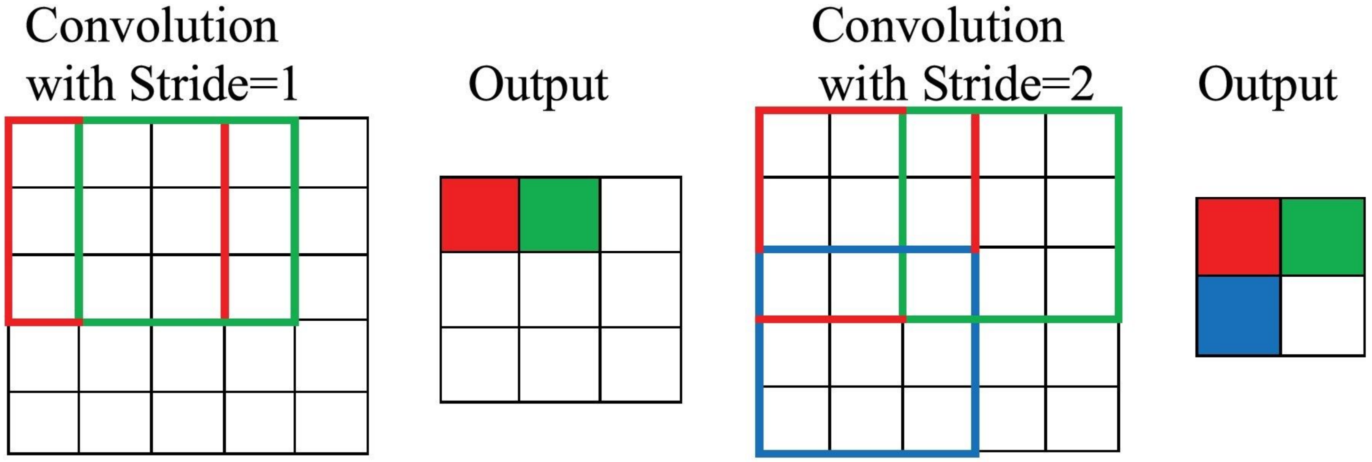
Question:

Let's assume we wanted to achieve translation invariance with a linear layer. How do we have to constraint the weights?



Recap
What we have discussed last week …

The stride controls the step size.
The larger the stride, the smaller the output feature map.

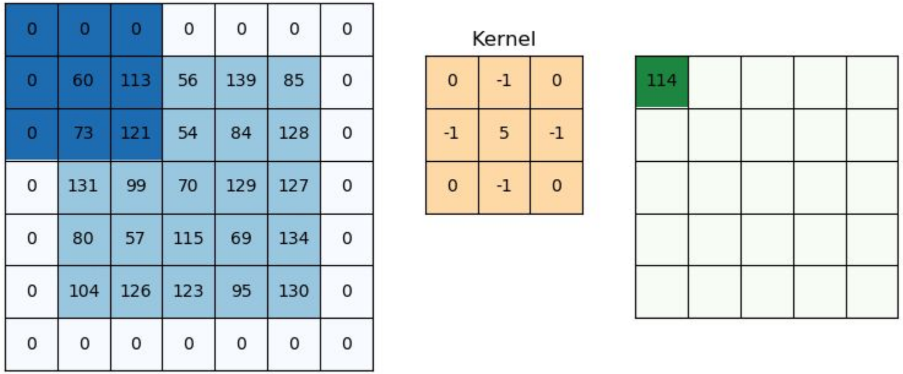


Recap

What we have discussed last week …

Padding changes the size of the input.
The larger the padding, the larger the output feature map.

Zero padding with P=1 implies that we add a row/column on **all four side** of the input feature map.



Recap

What we have discussed last week …

Assuming that the input is fixed, the convolutions output size can be influenced by modifying the following hyperparameters …

- **Kernel Size**
- **Stride**
- **Padding**

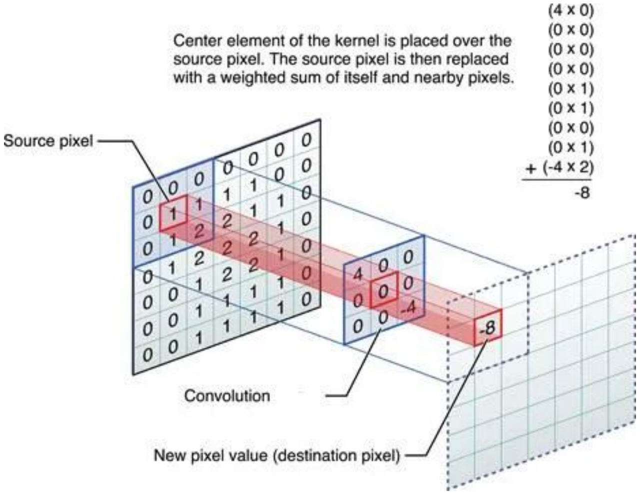
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

Disclaimer

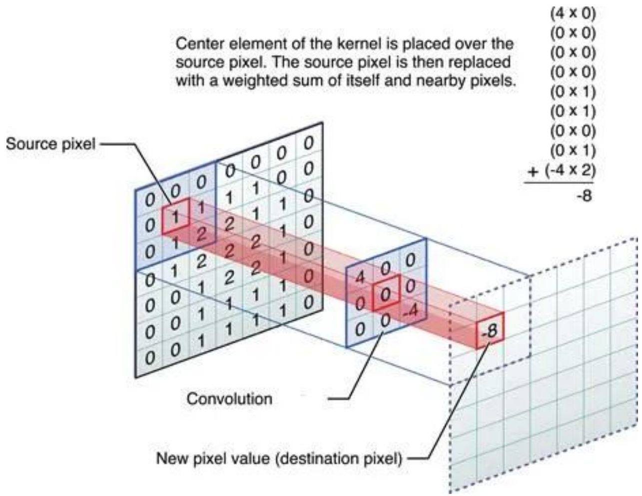
Note that the use of the term “convolution” in the context of deep learning is strongly misleading.

A google search for “2D convolution” yields visuals and formulas such as …



$$g(x, y) = f \star K = \sum_{u=-h}^{h} \sum_{v=-h}^{h} f(x+u, y+v)\mathbf{K}(u, v)$$

Disclaimer



$$\boldsymbol{g}(x, y) = \boldsymbol{f} \star \boldsymbol{K} = \sum_{u=-h}^{h} \sum_{v=-h}^{h} \boldsymbol{f}(x+u, y+v)\mathbf{K}(u, v)$$

However, signal processing exports will argue that the visuals/formulas shown a **correlation** (NOT a convolution).

Disclaimer

Note the difference in the definitions …

Correlation

$$g(x,y) = f \star K = \sum_{u=-h}^{h} \sum_{v=-h}^{h} f(x+u, y+v)\mathbf{K}(u,v)$$

h x h kernel K

Convolution

$$g(x,y) = f * K = \sum_{u=-h}^{h} \sum_{v=-h}^{h} f(x-u, y-v)\mathbf{K}(u,v)$$

A convolution is a correlation with the kernel matrix flipped horizontally/vertically.

Disclaimer

Note the difference in the definitions …

Correlation

$$g(x,y) = f \star K = \sum_{u=-h}^{h} \sum_{v=-h}^{h} f(x+u, y+v)\,\mathbf{K}(u,v)$$

Convolution

h x h kernel K

$$g(x,y) = f * K = \sum_{u=-h}^{h} \sum_{v=-h}^{h} f(x-u, y-v)\,\mathbf{K}(u,v)$$

So what does the PyTorch documentation say about the *nn.Conv2d* Layer?

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Disclaimer

Convolutional Layers compute a correlation!

So, why aren’t they called correlation layers?

Most like because math for convolutions turns out to be much nicer. Note for example that the convolution is commutative while the correlation isn’t.

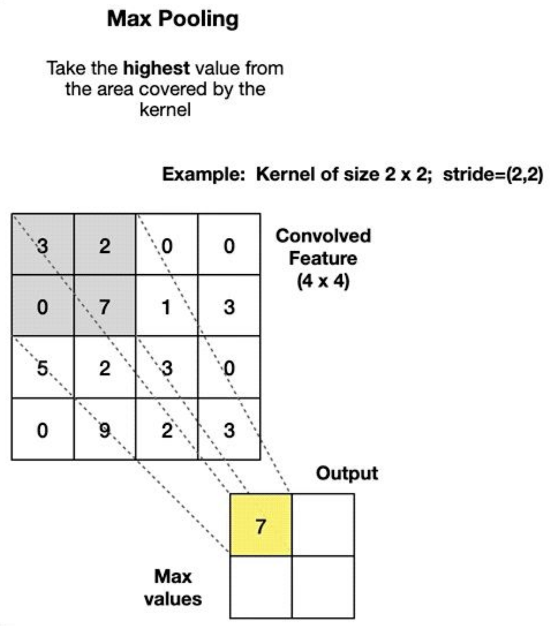
From a technical standpoint it doesn’t matter if we implement a correlation. Backpropagation will yield the reversed matrix.

Recap

What we have discussed last week …

Further NN layers which we encountered …

Pooling layers



Use to reduce the size of a feature map!

No learnable parameters

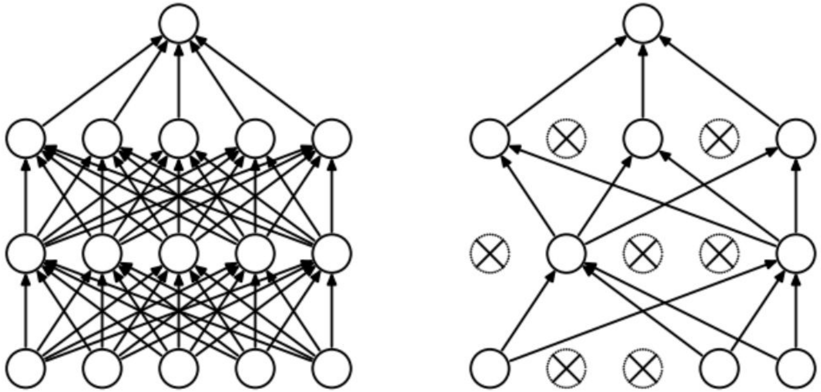
Recap
What we have discussed last week …

Further NN layers which we encountered …

Dropout Layers

Randomly disables the connection between two connected neurons (= randomly set weights to zero)

Purpose:
Regularization (= Prevent overfitting)



(a) Standard Neural Net (b) After applying dropout.

Recap

What we have discussed last week …

Pooling layers

Note that there exists various types of pooling layers such as …

- *Max Pool*
- *Average Pool*
- *Power Average Pool*
- *…*

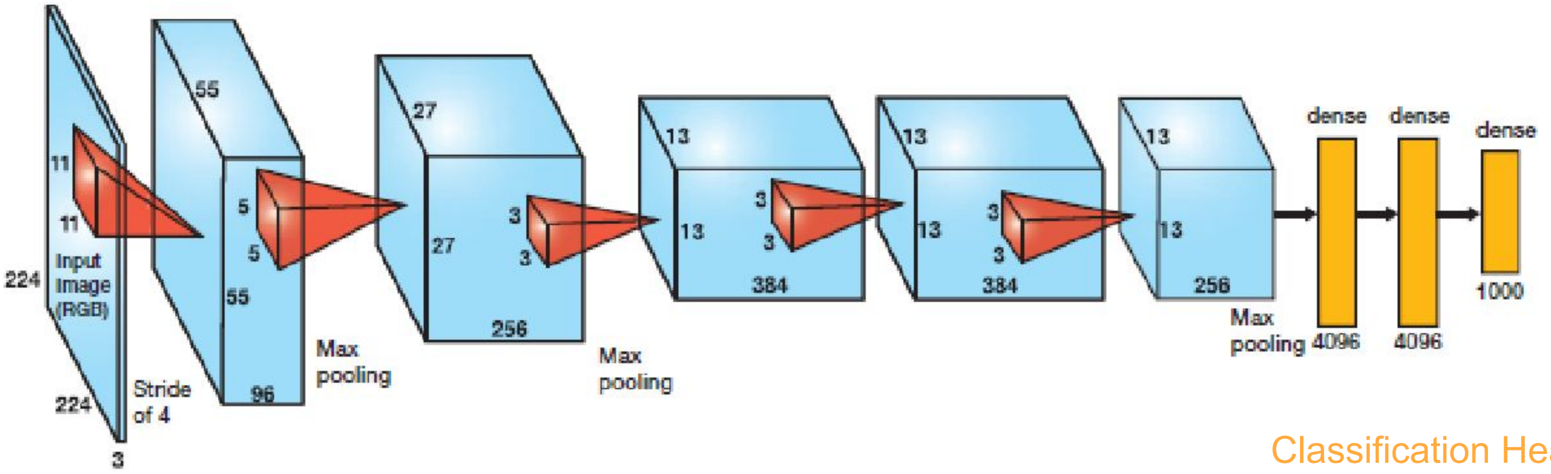
A pooling layer’s output size can be changed by modifying its

- Kernel size
- Stride
- Padding

Recap

What we have discussed last week …

AlexNet

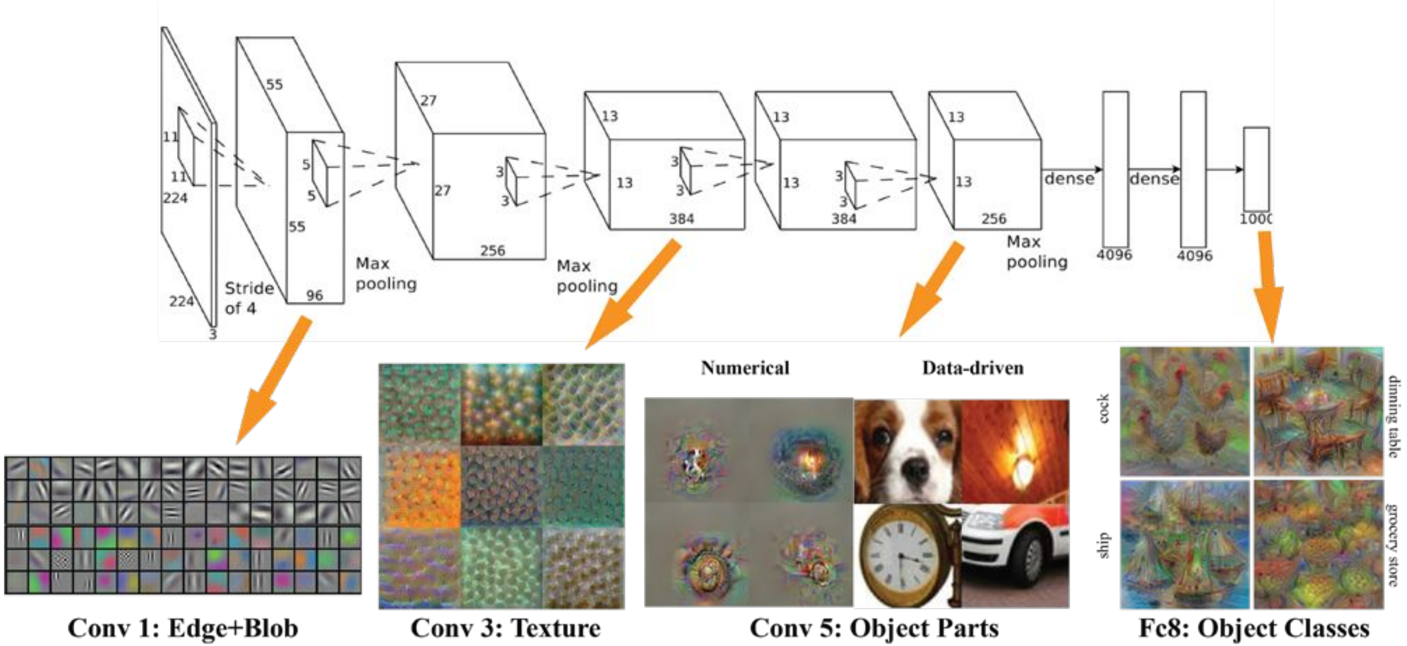


Feature extractor

Classification Head

Recap

What we have discussed last week …

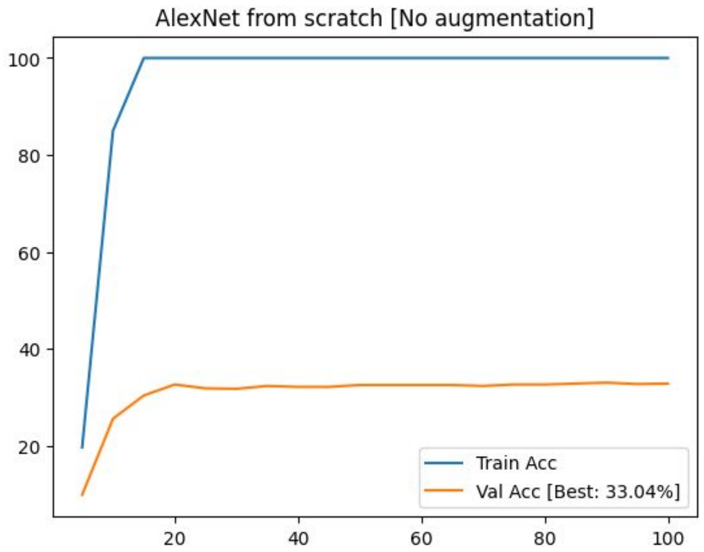


Recap

What we have discussed last week …

We trained AlexNet from scratch on a flower classification dataset.
100 classes. Only 10 images / class.

What was the result?

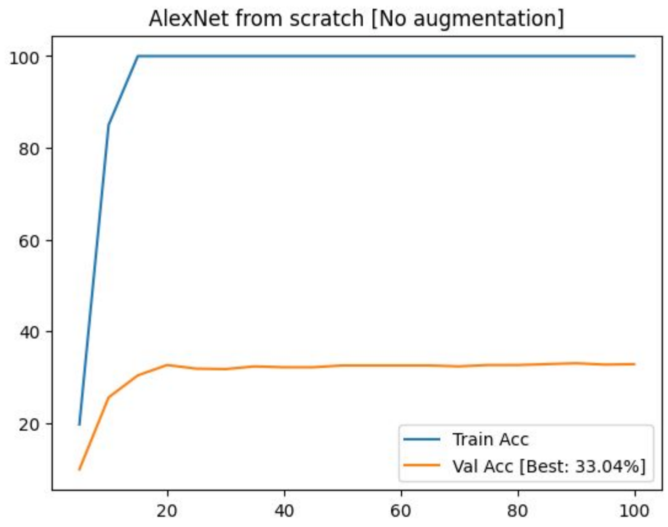


Recap

What we have discussed last week …

We trained AlexNet from scratch on a flower classification dataset.
→ 100 classes. Only 10 images / class.

Attempt 1:



Our network just memorizes training data!

Recap

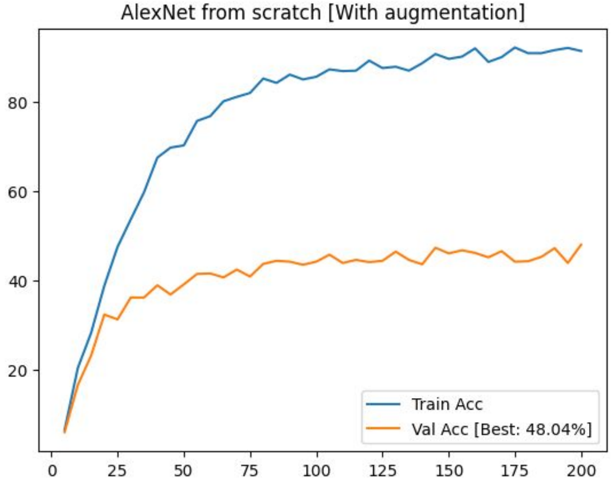
What we have discussed last week …

We trained AlexNet from scratch on a flower classification dataset.
→ 100 classes. Only 10 images / class.

Let’s increase this number artificially!

Attempt 2:

Applying image augmentation



Examples of image augmentations:
- translation
- rotation
- stretching
- shearing

Recap
What we have discussed last week …

Attempt 3: <u>Transfer Learning</u>

Let’s re-use the weights of an AlexNet model **pretrained** on ImageNet

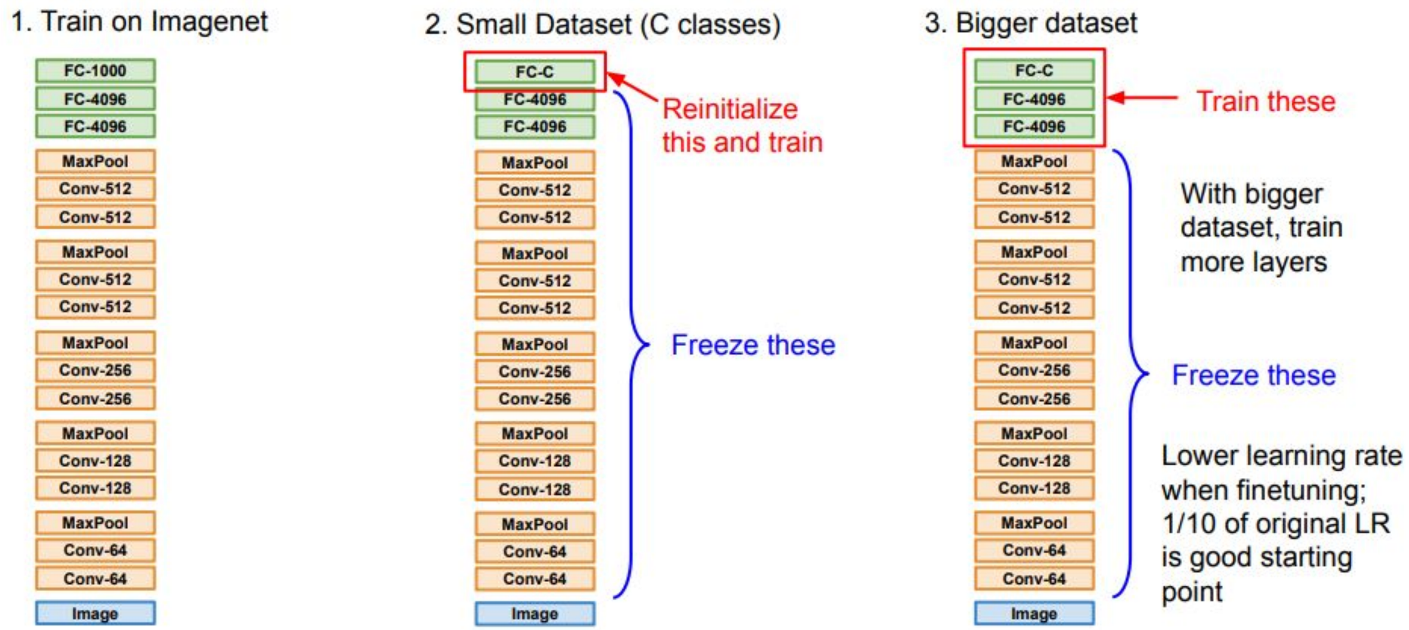
There are two ways to accomplish this:

(a) Finetune only the classification head
(b) Finetune the entire network (or at least including parts of the feature extractor)

Recap

What we have discussed last week …

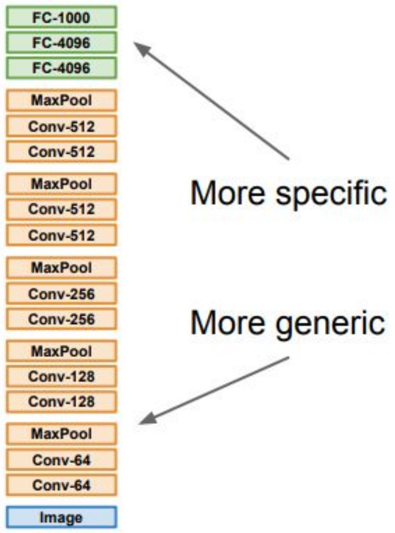
(a) Finetune only the classification head



Recap

What we have discussed last week …

(b) Finetune the entire network

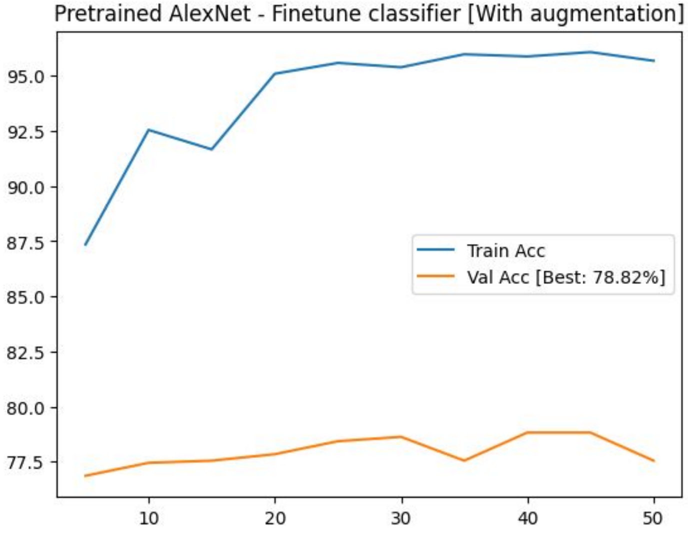


	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

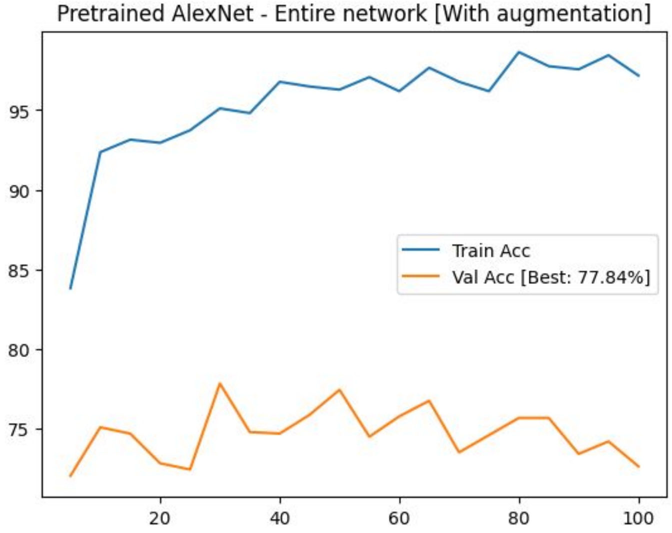
Recap

What we have discussed last week …

Attempt 3: Transfer Learning



Finetuned classifier



Finetuned entire network

Recap

What we have discussed last week …

Can we improve this even further?

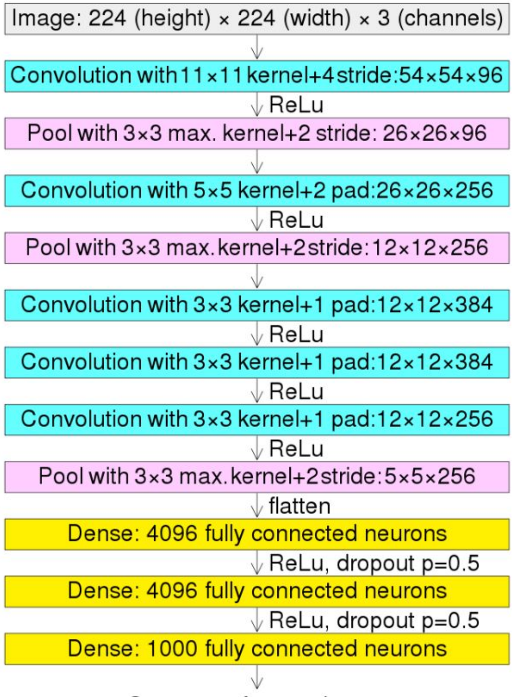
… AlexNet was publish in 2012. Maybe we should try a newer architecture …

CNN Architectures

AlexNet

[Krizhevsky et al. 2012]

Large kernel size in the first two layers



Details/Retrospectives:

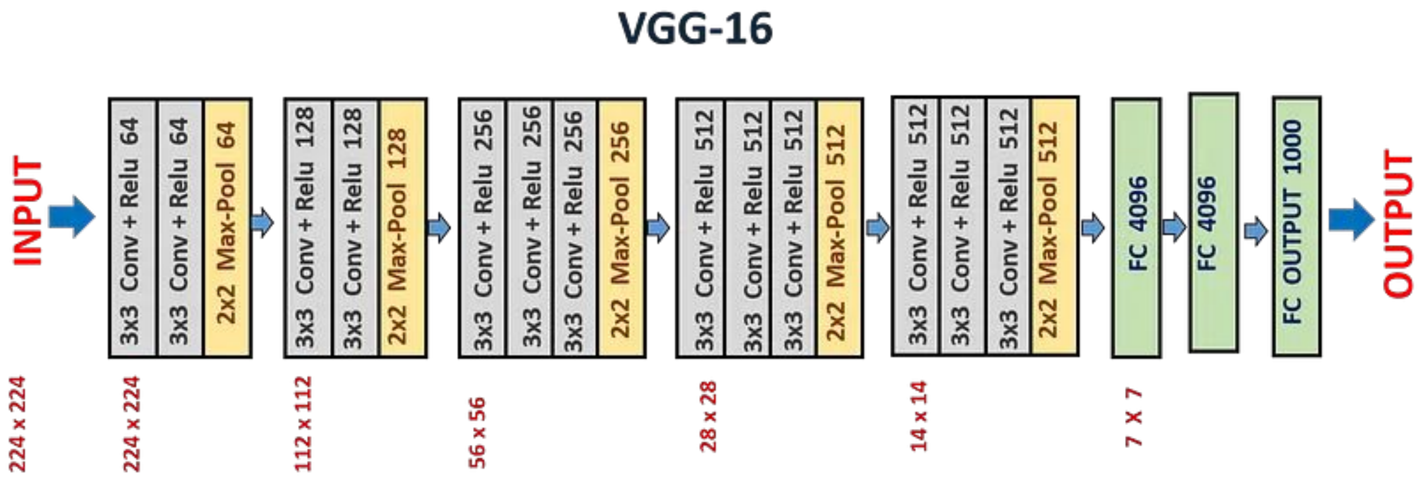
- **first use of ReLU**
- **used Norm layers (not common anymore)**
- **heavy data augmentation**
- **dropout 0.5 - batch size 128**
- **SGD Momentum 0.9 - Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus - L2 weight decay 5e-4**

- **7 CNN ensemble: 18.2% -> 15.4% on ImageNet**

VggNet

[Simonyan and Zisserman, 2014]

... Let's go deeper!



- **16 Layers with learnable params**
- **Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2**

VggNet

[Simonyan and Zisserman, 2014]

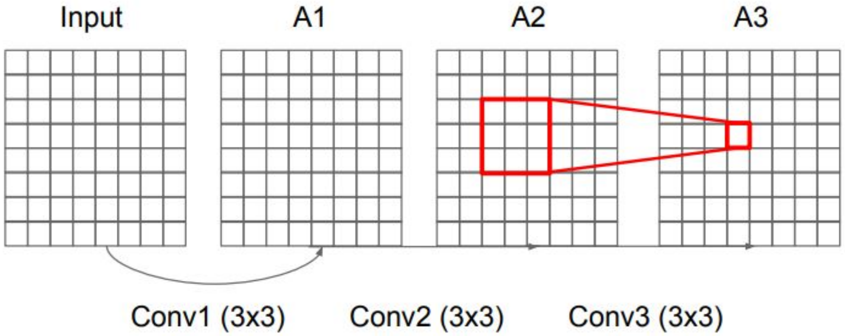
And there exists an even deeper version with 19 learnable layers …

7.3% top 5 error in ILSVRC'14

VggNet
[Simonyan and Zisserman, 2014]

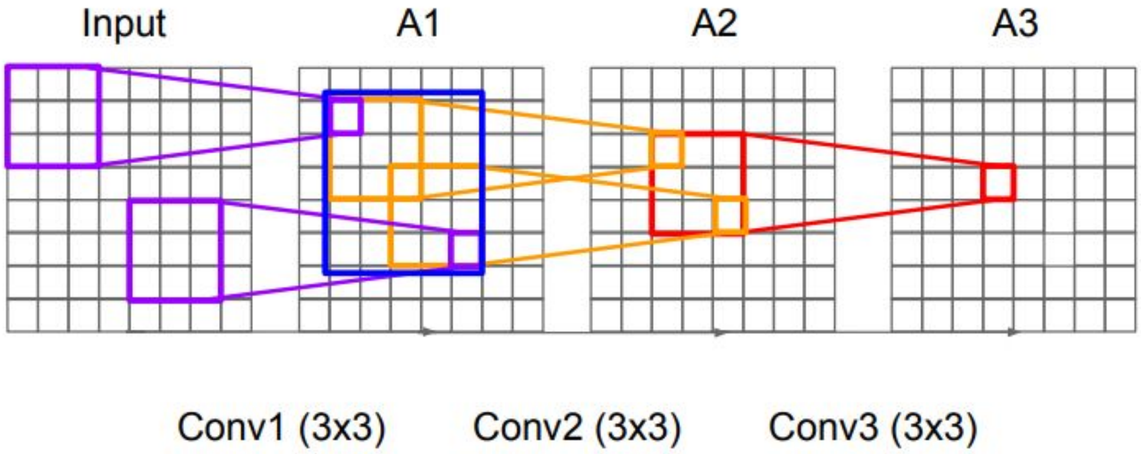
Notice that each block consists of multiple 3x3 convolution.

Question: How does stacking multiple convolutional layers affect the receptive field?



VggNet
[Simonyan and Zisserman, 2014]

Three 3x3 convolutional layers (s=1, p=0) have an receptive field of 7x7 pixel.



VggNet
[Simonyan and Zisserman, 2014]

Hence, three 3x3 conv layers have the same receptive field as one 7x7 conv layer.

But deeper, more non-linearities!

VggNet
[Simonyan and Zisserman, 2014]

Hence, three 3x3 conv layers have the same receptive field as one 7x7 conv layer.

But deeper, more non-linearities!

Question:

- **How many learnable weight parameters does one BLOCK have?**
- **How many learnable weight parameters does a 7x7 convolutional layer have?**

VggNet

[Simonyan and Zisserman, 2014]

- **How many learnable weights does one BLOCK have?**

  $(3^2 C_{IN})C + 2*(3^2 C)C$

  e.g. $C = C_{IN} = 256$ => 1 769 472

  Less weight params!

- **How many learnable weights does a 7x7 convolutional layer have?**

  $7^2 C_{IN}\, C$

  e.g. $C = C_{IN} = 256$ => 3 211 264

VggNet

[Simonyan and Zisserman, 2014]

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Note:

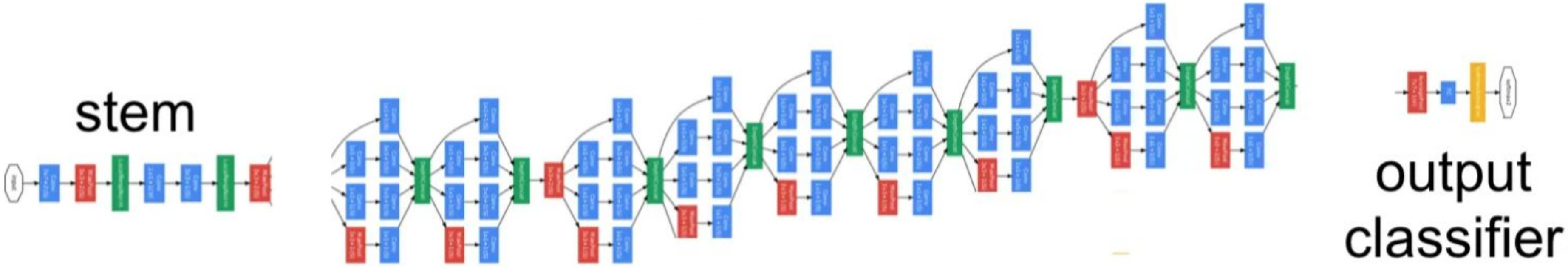
Most memory is in early CONV

Most params are in late FC

InceptionNet v1
[Szegedy et al., 2014]

Also referred to as GoogLeNet

Going deeper, but with less parameters



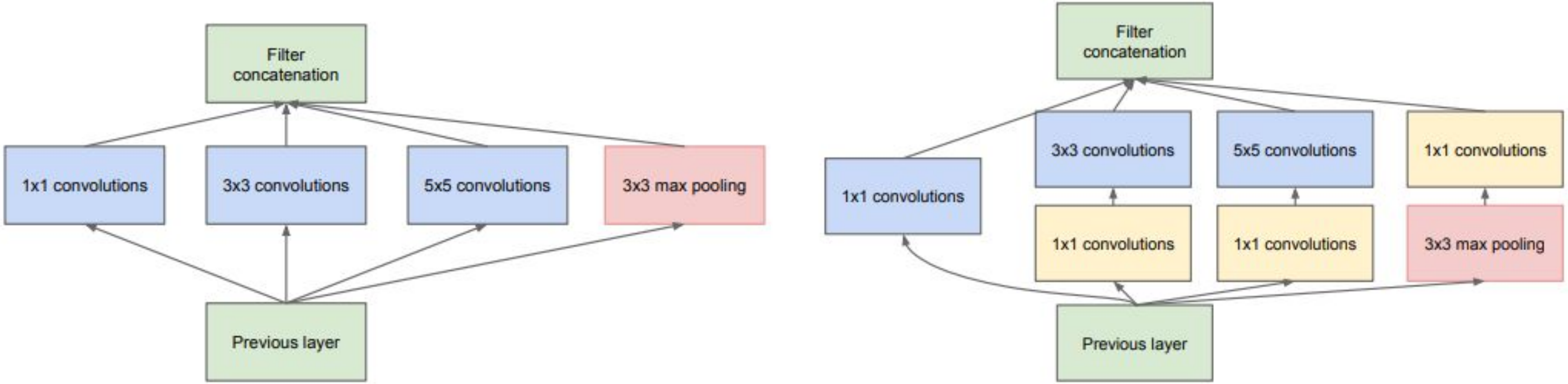
- 22 layers with learnable parameters
- No fully connected layers
- Only 5 million parameters!
  - 12x less than AlexNet
  - 27x less than VGG-16

6.7% top 5 error on ImageNet

InceptionNet v1

[Szegedy et al., 2014]

Composed of Inception modules

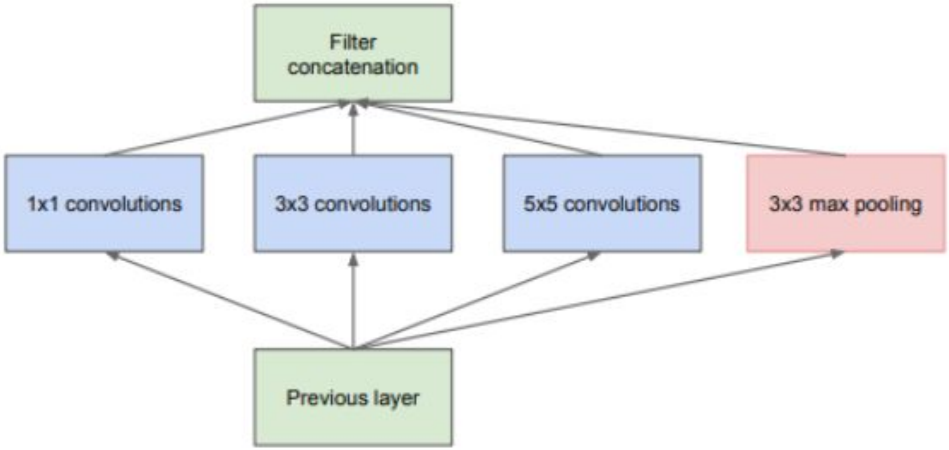


(a) Inception module, naïve version

(b) Inception module with dimension reductions

InceptionNet v1

[Szegedy et al., 2014]



(a) Inception module, naïve version

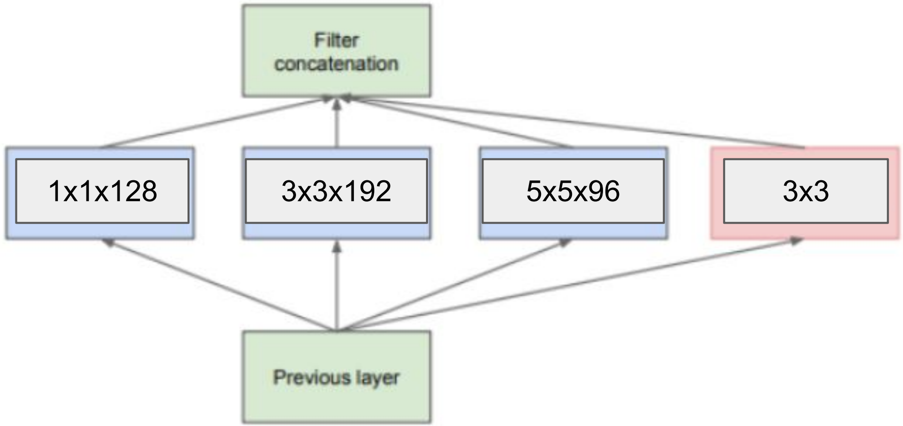
Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise.

InceptionNet v1

[Szegedy et al., 2014]

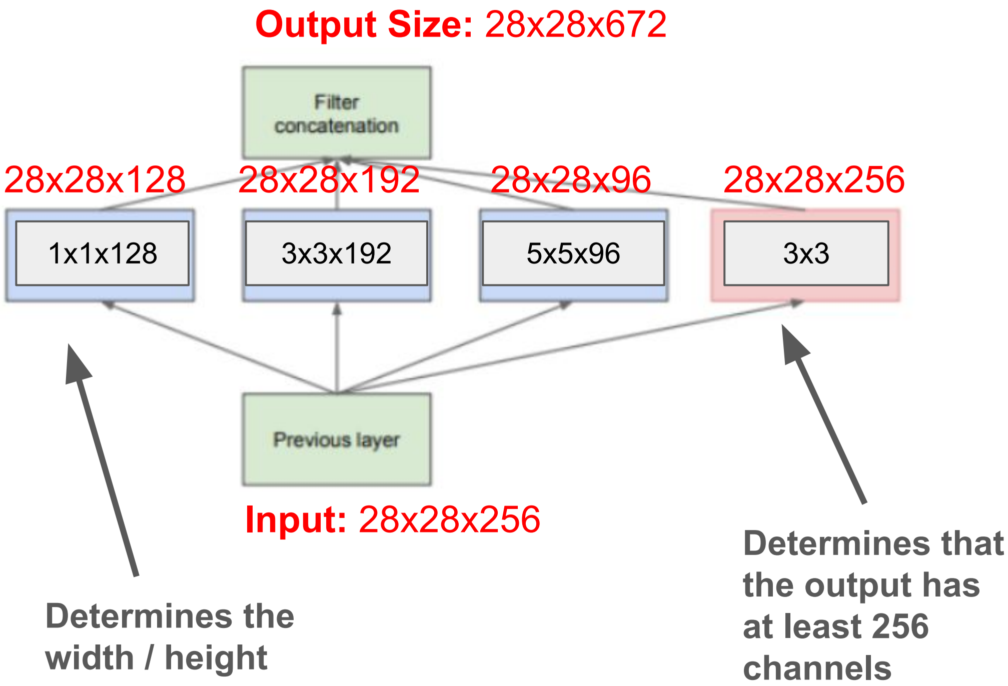


Input: 28x28x256

Question:
What is the output size of this module?

InceptionNet v1

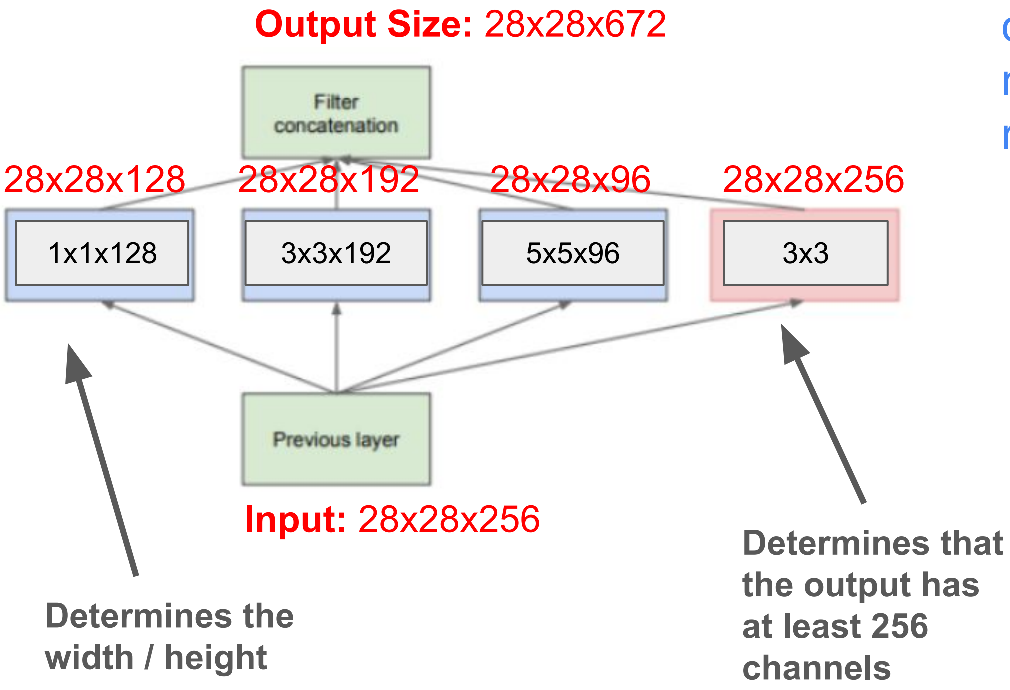
[Szegedy et al., 2014]



Question:
What is the output size of this module?

InceptionNet v1

[Szegedy et al., 2014]

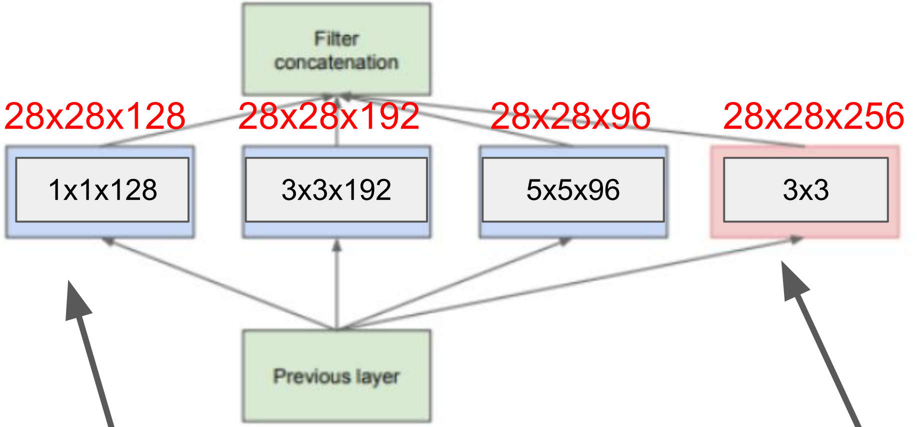


Question:
What is the computational complexity? (=> How many multiplication operations are required?)

Pooling preserves the feature depth, which means total depth after concatenation can only grow at every layer!

InceptionNet v1
[Szegedy et al., 2014]

Output Size: 28x28x672



Input: 28x28x256

Determines the width / height

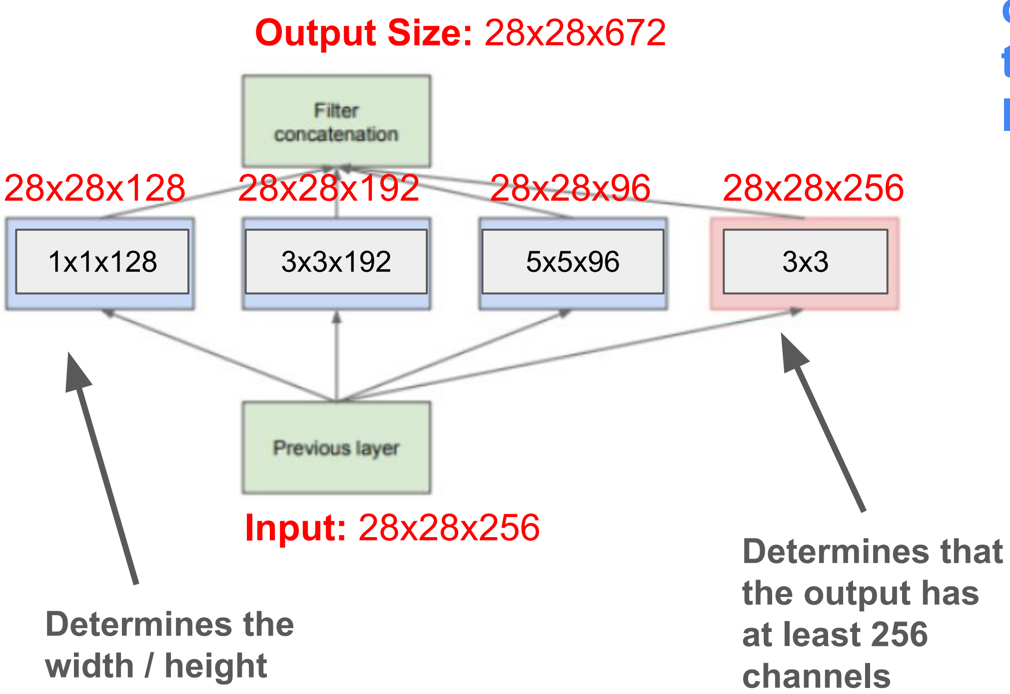
Determines that the output has at least 256 channels

Question:
What is the computational complexity? (=> How many multiplications are required?)

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x**192x3x3x256**
[5x5 conv, 96] 28x28x**96x5x5x256**
Total: 854M ops

InceptionNet v1

[Szegedy et al., 2014]

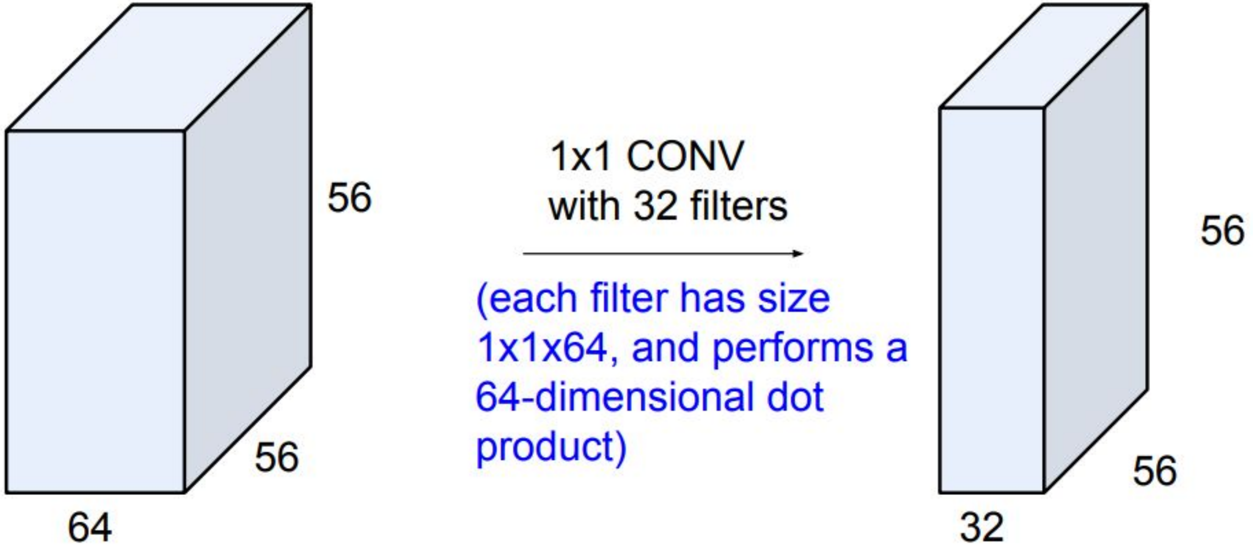


Even a modest number of 5×5 convolutions can be expensive on top of a convolutional layer with a large number of filters!

We need some tool to reduce the number of channels.

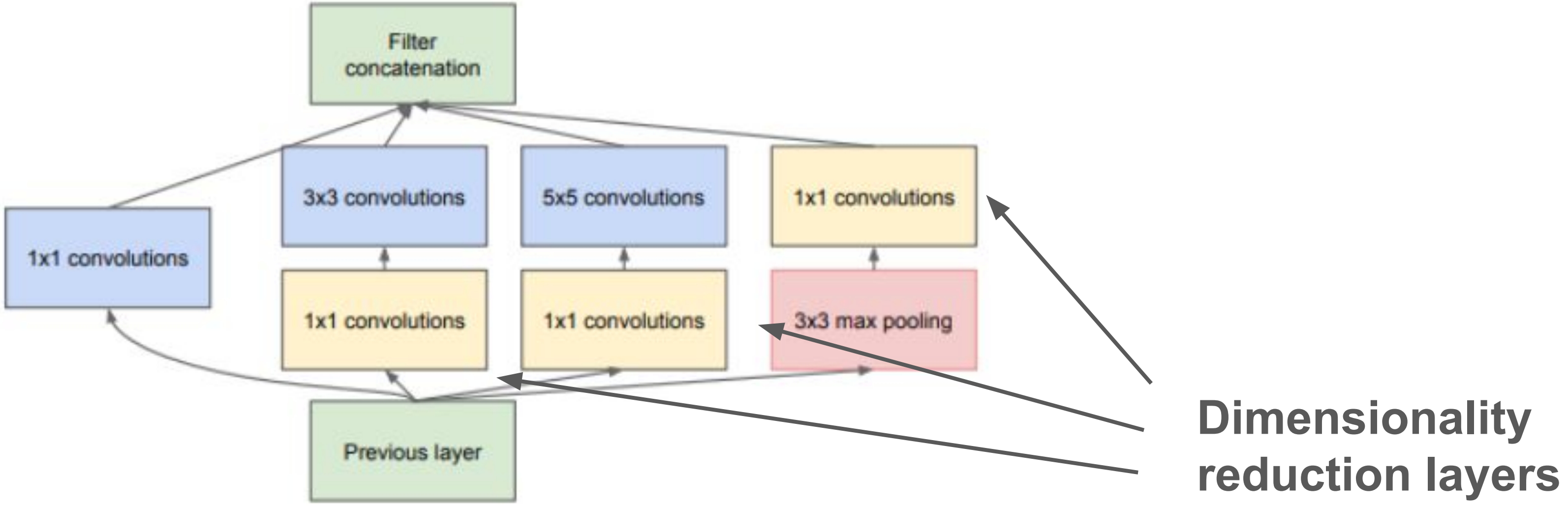
InceptionNet v1
[Szegedy et al., 2014]

Reduce the dimensionality with a 1x1 convolution



InceptionNet v1

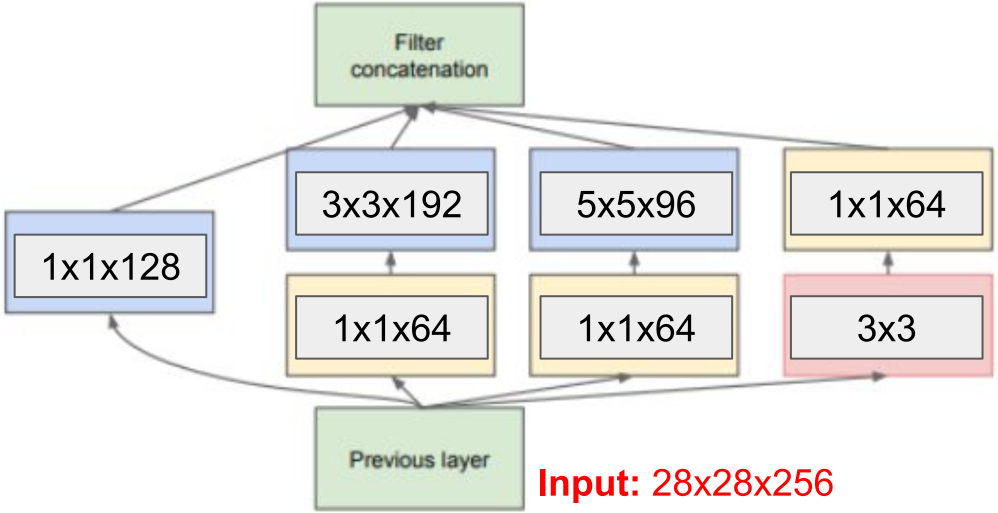
[Szegedy et al., 2014]



(b) Inception module with dimension reductions

InceptionNet v1
[Szegedy et al., 2014]

Output: 28x28x480



Input: 28x28x256

(b) Inception module with dimension reductions

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

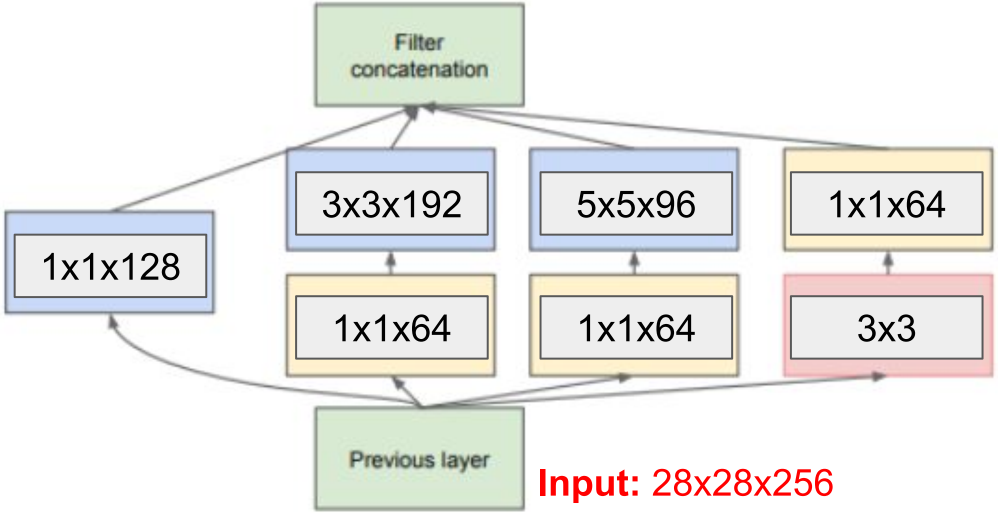
Compared to 854M ops for the naive version

InceptionNet v1

[Szegedy et al., 2014]

Add ReLU activations after each conv layer!

Output: 28x28x480



Input: 28x28x256

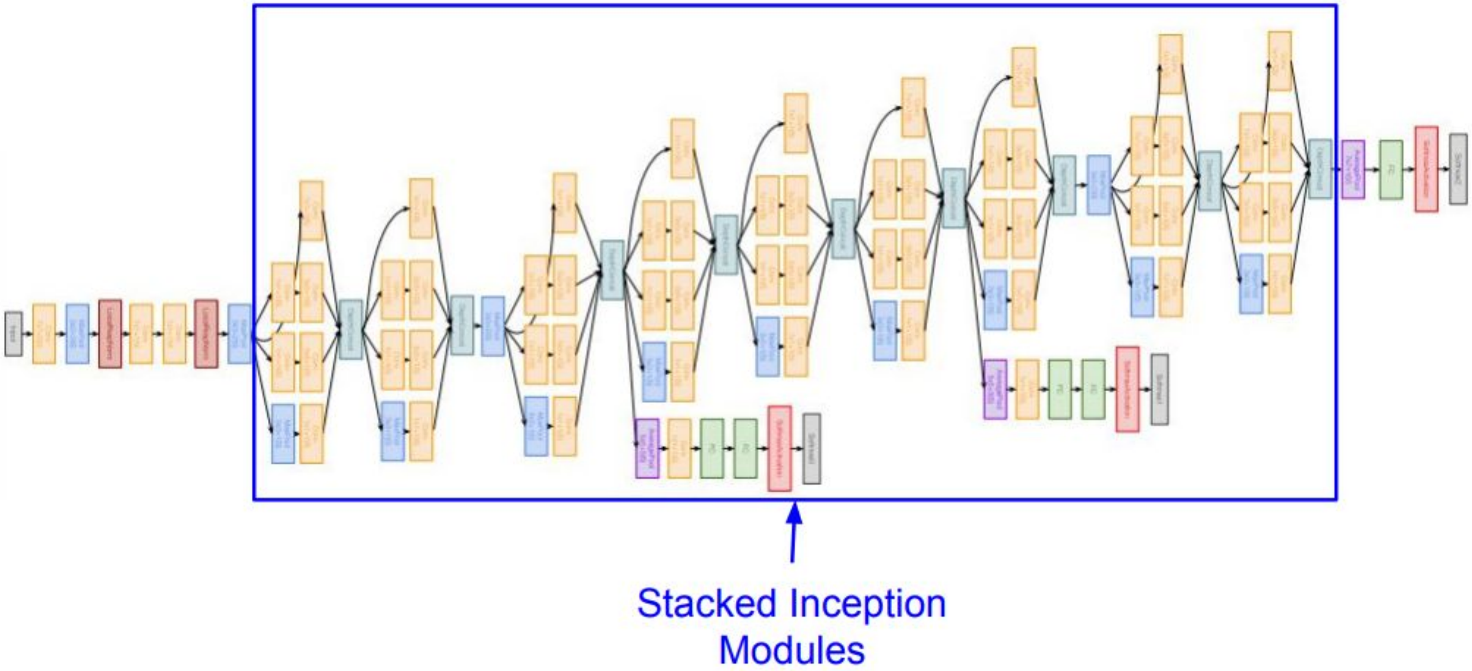
(b) Inception module with dimension reductions

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Compared to 854M ops for the naive version

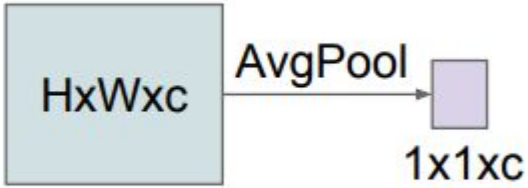
InceptionNet v1

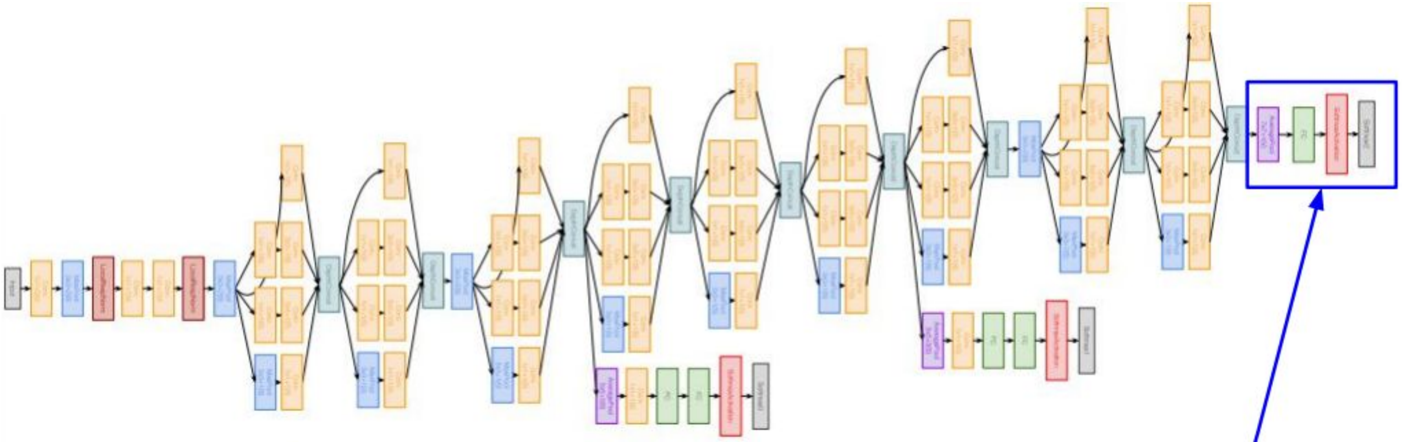
[Szegedy et al., 2014]



InceptionNet v1

[Szegedy et al., 2014]





Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Classifier output

What if we go deeper?

Based on the previous results, it seems that deeper networks always result in a better performance.

What happens when we continue stacking deeper layers?

What if we go deeper?

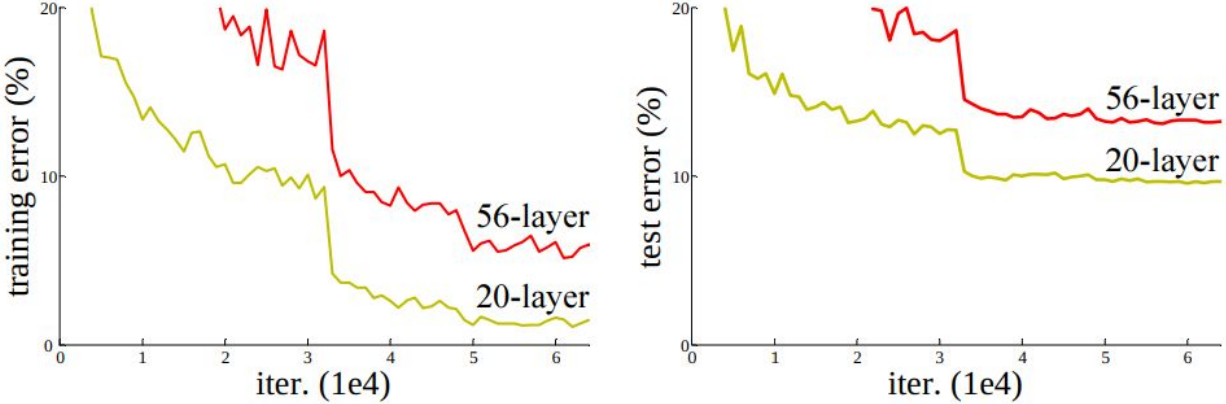


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

What if we go deeper?

The 56-layer model performs worth on the test and on the TRAINING set!
This is not a sign of overfitting.

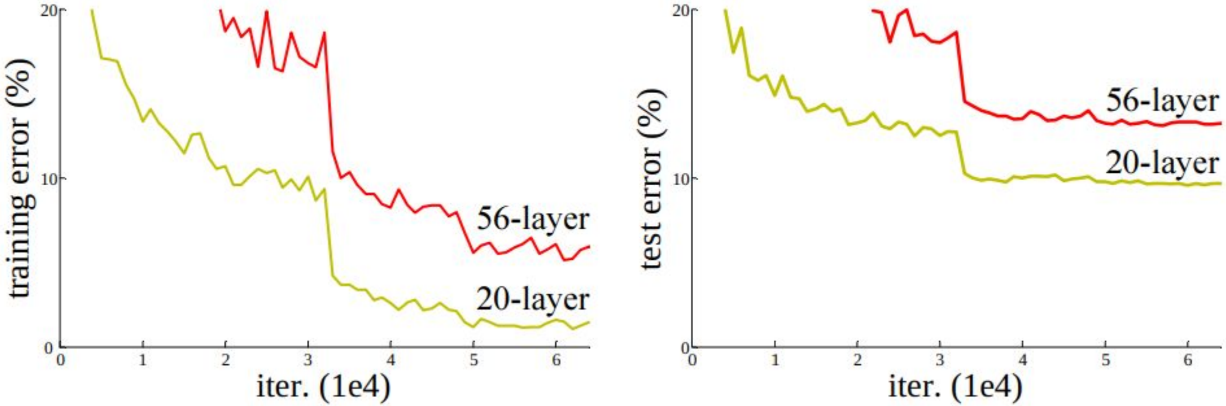


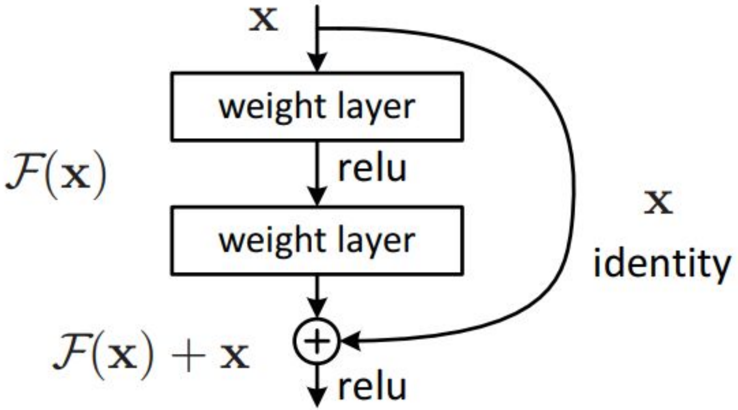
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

ResNet

[He et al., 2015]

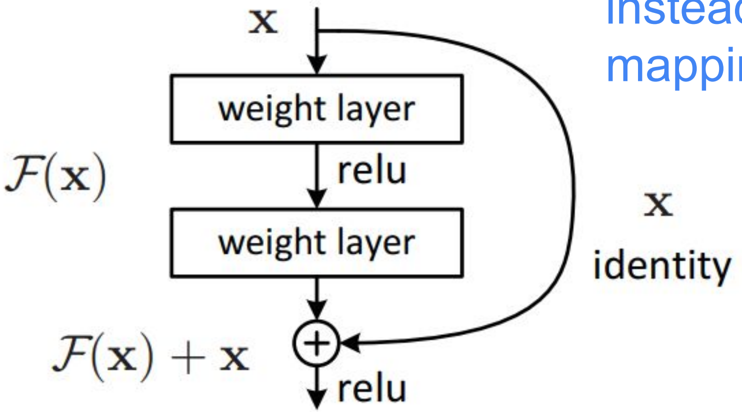
Deep models have more parameters than shallower models.
Hence, a deeper model should be at least as good as a shallower model.
Hypothesis: The problem is related to optimization.

ResNet proposes skip-connections to solve this problem.



ResNet

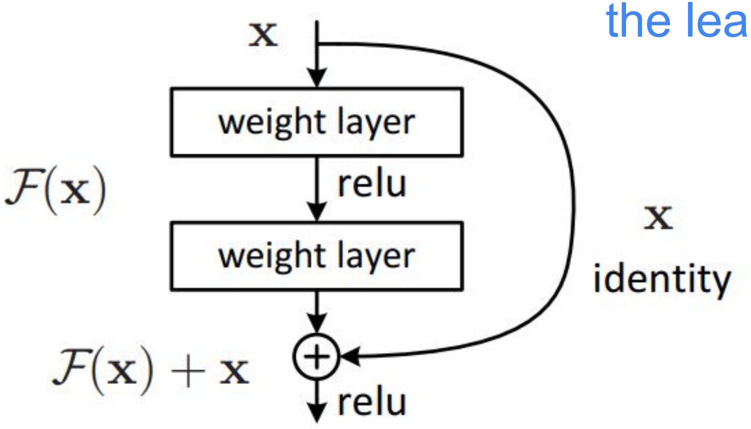
[He et al., 2015]



We learn the residual ("the difference") instead of the desired underlying mapping.

ResNet

[He et al., 2015]



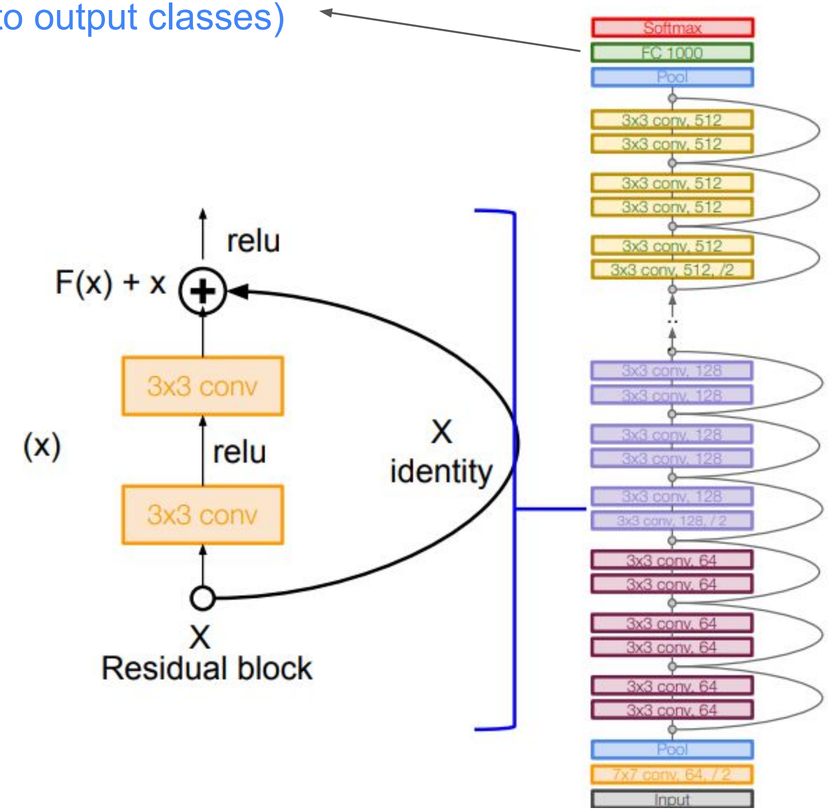
If we want to learn the identity mapping, the learned residual will be zero.

ResNet

[He et al., 2015]

No FC layers at the end (only FC 1000 to output classes)

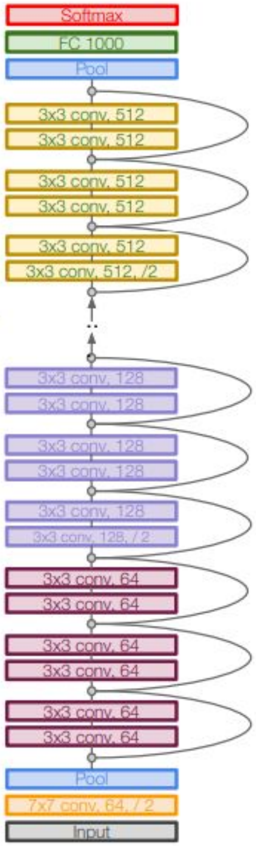
- We stack residual blocks.
- Every residual block has two 3x3 conv layers.
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension) Reduce the activation volume by half.



ResNet

[He et al., 2015]

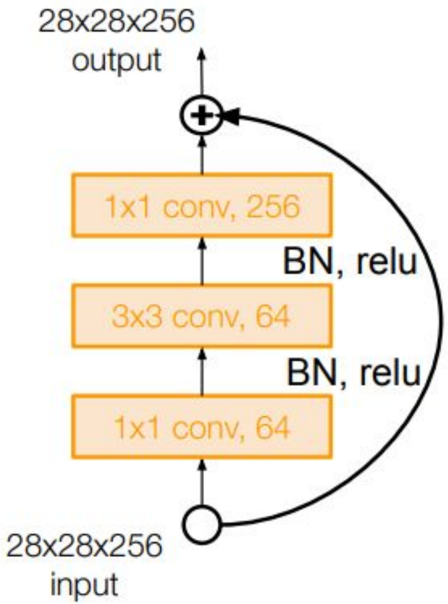
Total depths of 18, 34, 50, 101, or 152 layers



ResNet

[He et al., 2015]

For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)



ResNet

[He et al., 2015]

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"!

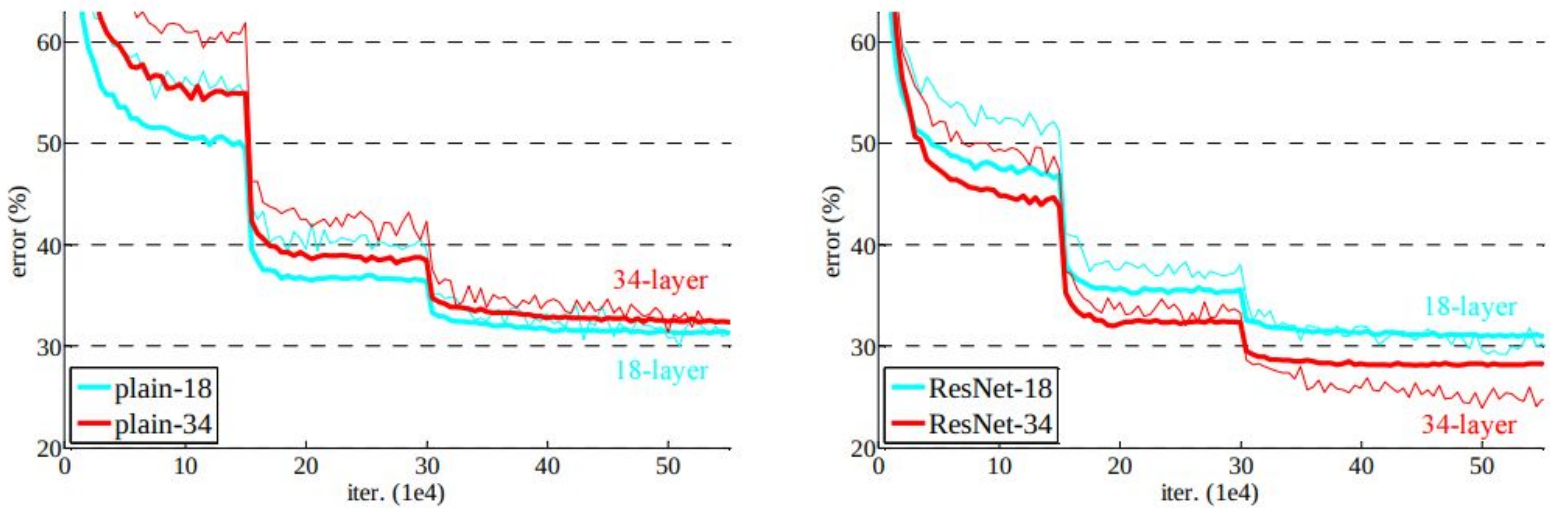
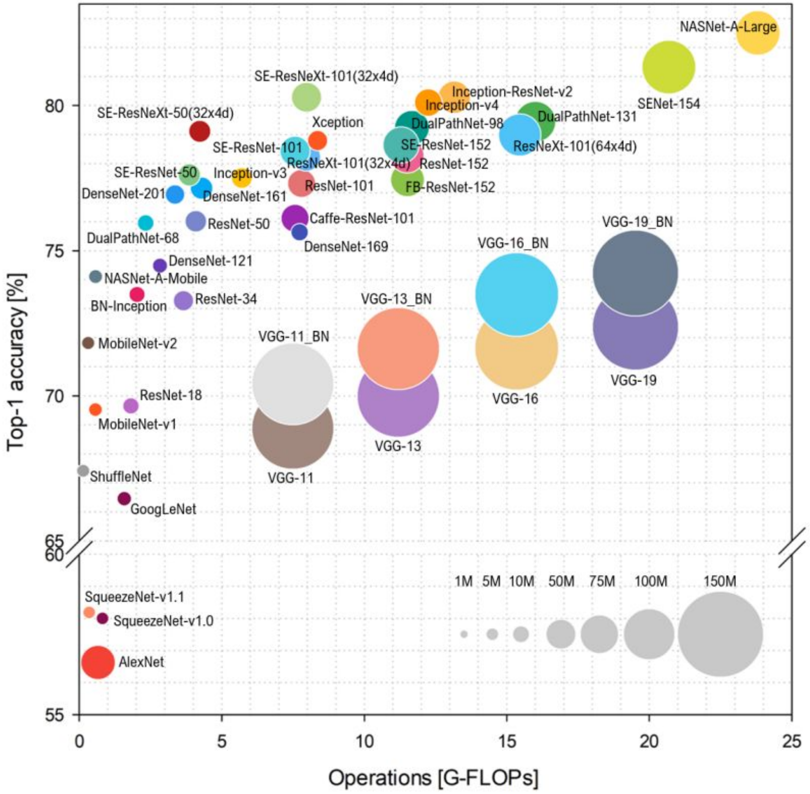


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

Tradeoff: Accuracy - Model Size - FLOPs



Floating-point operations (FLOPs) required for a single forward pass are reported. The size of each ball corresponds to the model complexity.

Main takeaways

AlexNet showed that you can use CNNs to train Computer Vision models.
VGG shows that bigger networks work better
InceptionNet (V1) is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers
ResNet showed us how to train extremely deep networks - Limited only by GPU & memory! - Showed diminishing returns as networks got bigger
After ResNet: CNNs were better than the human metric and focus shifted to efficient networks ⇒ Lots of tiny networks aimed at mobile devices: **MobileNet**, **SqueezeNet**