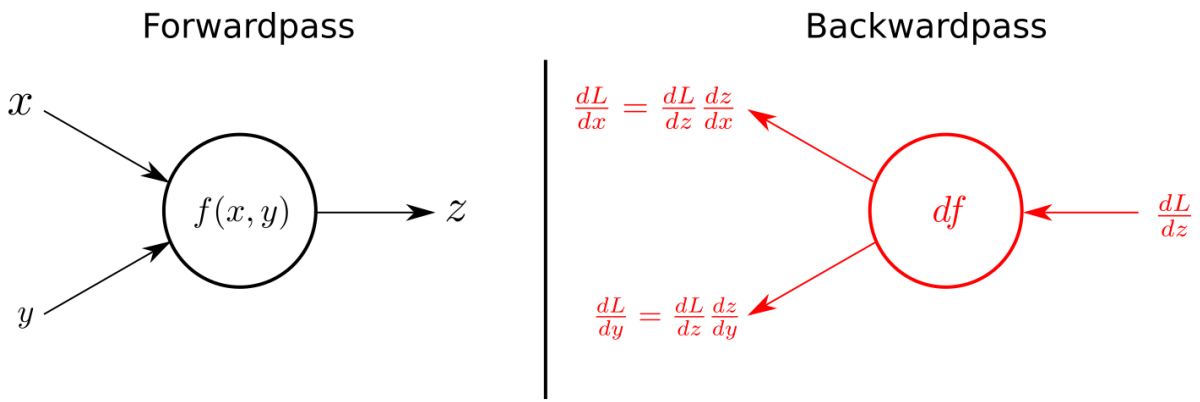
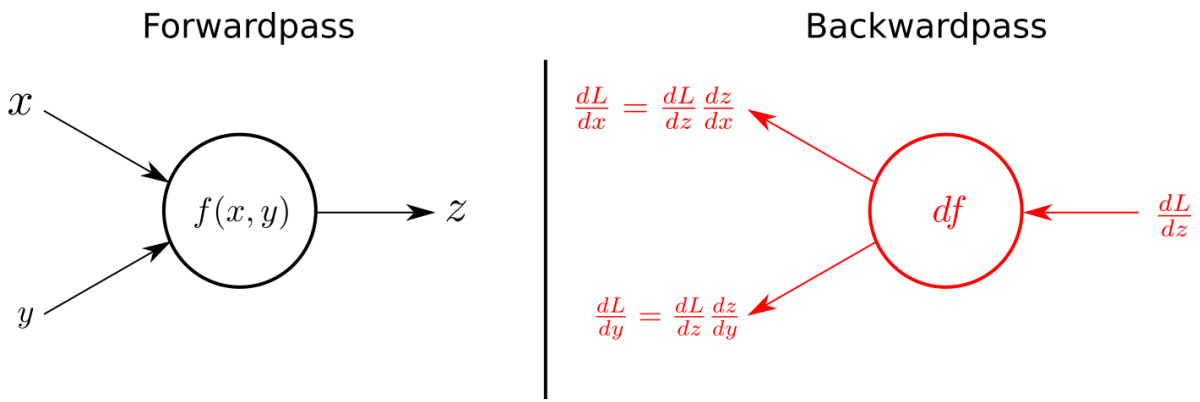
Backpropagation



The forward pass calculates **z** as a function **f(x,y)** using the input variables **x** and **y**

https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

Backpropagation



Backward pass receives **dL/dz**, the gradient of the loss function with respect to **z** from above

The gradients of **x** and **y** on the loss function can be calculated by applying the chain rule

https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

Convolution - Backpropagation

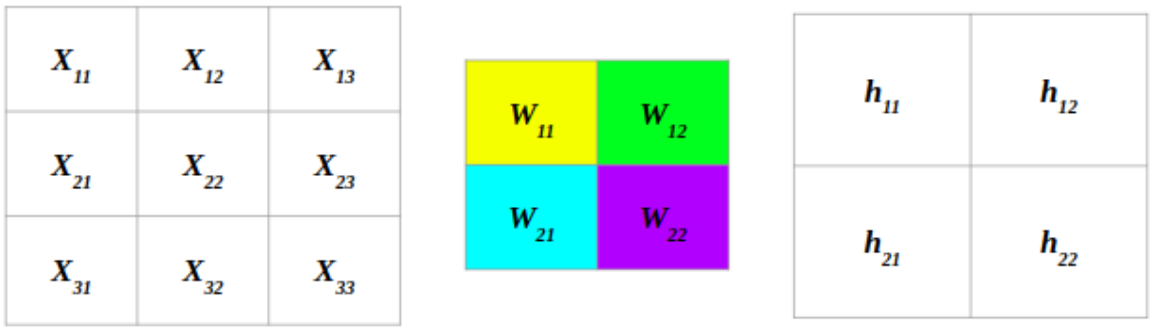
- Notice that each weight in the filter contributes to each pixel in the output map. Any change in a weight in the filter will affect all the output pixels.

- Thus, all these changes add up to contribute to the final loss. Thus, we can easily calculate the derivatives as follows.

- The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters)!

Source: https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Convolution - Forwardpropagation



Input Size : 3×3, **Filter Size** : 2×2, **Output Size** : 2×2

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$$

$$h_{22} = W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}$$

Source: https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Convolution - Backpropagation

Assume that:

$$\partial h_{ij} \text{ represents } \frac{\partial L}{\partial h_{ij}}$$

$$\partial w_{ij} \text{ represents } \frac{\partial L}{\partial w_{ij}}$$

- We can assume that we get $\partial \mathbf{h}$ as input (from the backward pass of the next layer) and our aim is to calculate $\partial \mathbf{w}$ and $\partial \mathbf{x}$.
- It is important to understand that $\partial \mathbf{x}$(or $\partial \mathbf{h}$ for previous layer) would be the input for the backward pass of the previous layer.
- This is the core principle behind the back propagation.

Source: https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Convolution - Backpropagation

X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

$X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$	$X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$
$X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$	$X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$

∂h_{11}	∂h_{12}
∂h_{21}	∂h_{22}

$$\partial W_{12} = X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$$

$$\partial W_{21} = X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$$

$$\partial W_{22} = X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$$

- The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with spatially-flipped filters).
- Each weight in the filter contributes to each pixel in the output map. Thus, any change in a weight in the filter will affect all the output pixels.

Source: https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199

Activation Function

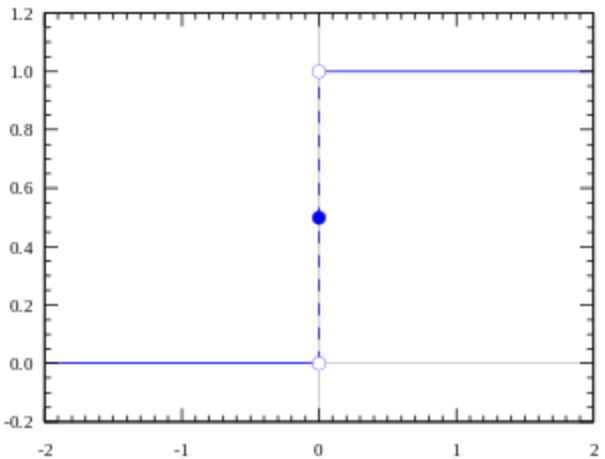
- A neuron simply calculates a “weighted sum” of its input, adds a bias and then decides whether it should be “fired” or not

$$y = w_1x_1 + w_2x_2 + ... + w_Nx_N + b$$

- Value of y could be anything (no bounds). So we add an activation function to decide a neuron should “fire” (activate) or not.

Activation Function

– Idea: Use a simple step function –activated when y>0



– This has a drawback – we will be combining multiple neuron outputs in the next layer- so we want something to give us (analog) activation values rather than saying “activated” or not (binary).

Activation Function

- Idea: Use a linear function

$$A = cx$$

- It gives a range of activations, so it is not binary activation and we can connect a few neurons together and if more than 1 fires, we could take the max (or softmax) and decide based on that.

Activation Function

- Idea: Use a linear function

- Now each layer is activated by a linear function. That activation in turn goes into the next level as input and the second layer calculates weighted sum on that input and it in turn, fires based on another linear activation function.

- No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer!

That means these N layers can be replaced by a single layer.
No matter how we stack, the whole network is still equivalent to a single layer with linear activation (a combination of linear functions in a linear manner is still another linear function)

Activation Function

Logistic function $$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

– A standard logistic function is called sigmoid function (k=1,x_0=0,L=1)

$$S(x) = \frac{1}{1+e^{-x}}$$

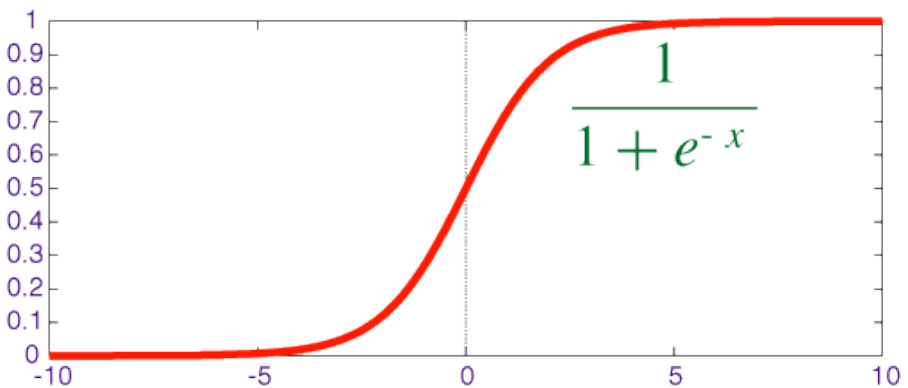
It is nonlinear in nature ➔ Combinations of this function are also nonlinear!
It will give an analog activation unlike step function.
It has a smooth gradient too.

Sigmoid

- The sigmoid function takes any range real number and returns the output value which falls in the range of 0 to 1.

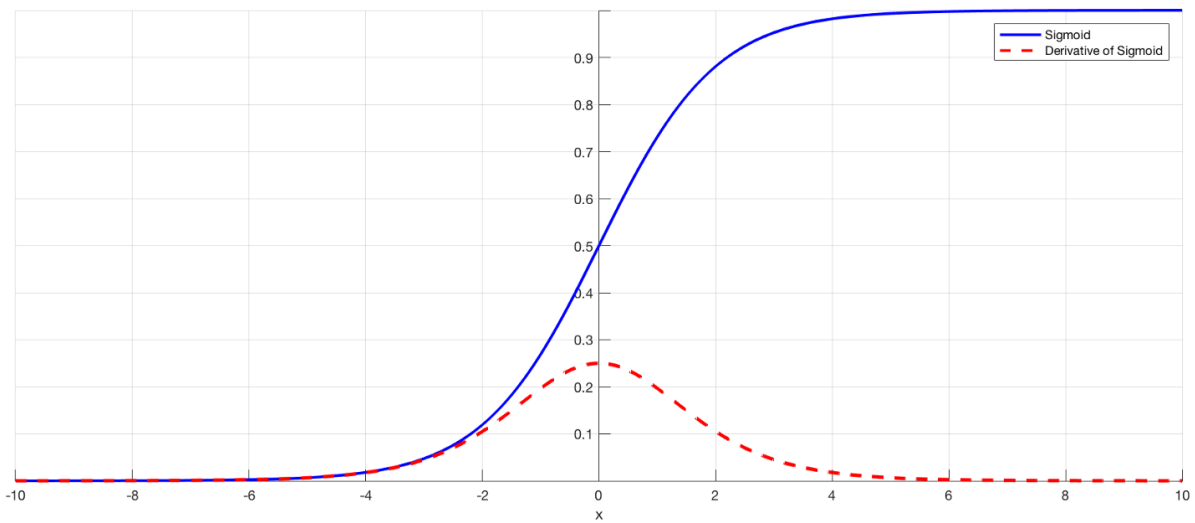
- It's often used in logistic regression for a binary classification and also as activation function in neural networks.

- The output of the sigmoid function can be interpreted as a probability.



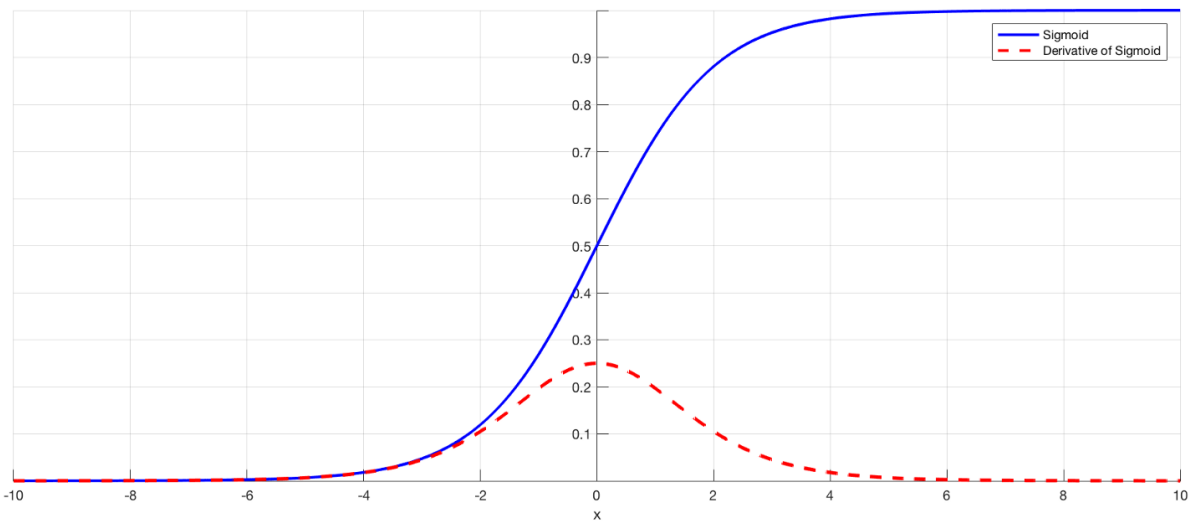
Sigmoid

- Towards either end of the sigmoid function, the Y values tend to respond very less to changes in X and the gradient at that region is going to be small.
- When *n* hidden layers use an activation like the sigmoid function, *n* small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers.



Sigmoid

- It gives rise to a problem of “vanishing gradients”. (cannot make significant change because of the extremely small value).
- The network won’t learn further or is drastically slow (depending on use case and until gradient /computation gets hit by floating point value limits).
- There are ways to work around this problem and sigmoid is still very popular in classification problems.

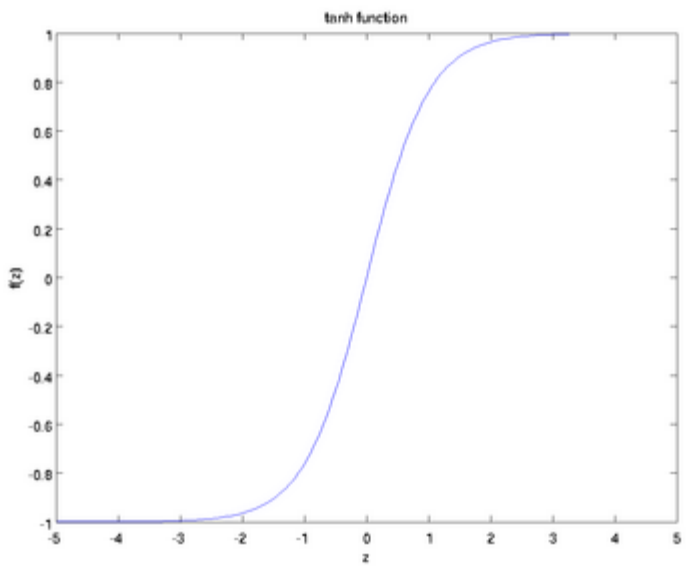


Tanh

– It is a scaled sigmoid function: $tanh(x) = 2\,sigmoid(2x) - 1$

– The gradient is stronger for tanh than sigmoid (derivatives are steeper). Deciding between the sigmoid or tanh will depend on your requirement of gradient strength.

– Like sigmoid, tanh also has the vanishing gradient problem.

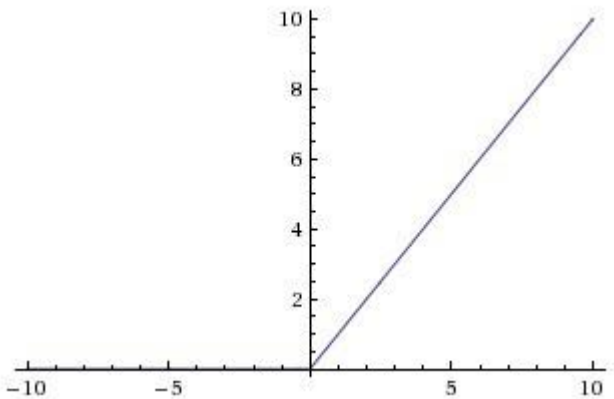


$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

ReLu

- A(x) = max(0,x)

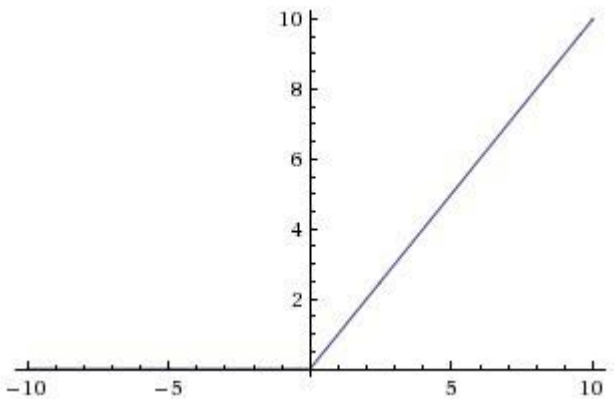
- It is nonlinear in nature and combinations of ReLu are also non linear! (in fact it is a good approximator. Any function can be approximated with combinations of ReLu).
- This means we can stack layers.
- It is not bound. The range of ReLu is [0, inf).



ReLu

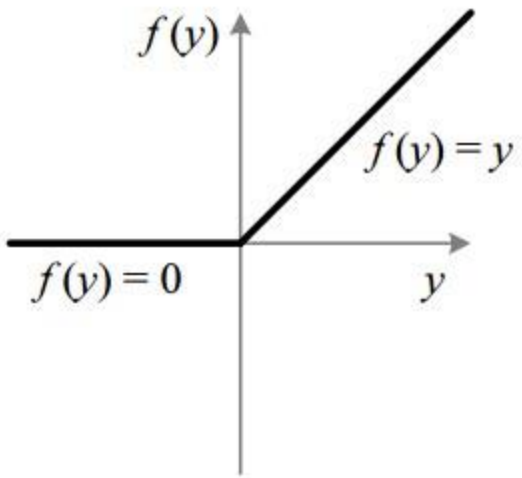
$$A(x) = \max(0,x)$$

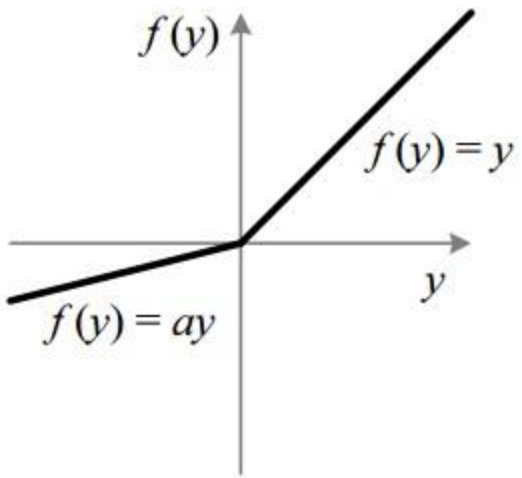
- Imagine a network with random initialized weights (or normalised) and almost 50% of the network yields 0 activation because of the characteristic of ReLu (output 0 for negative values of x).
- This means a fewer neurons are firing (sparse activation) and the network is lighter.



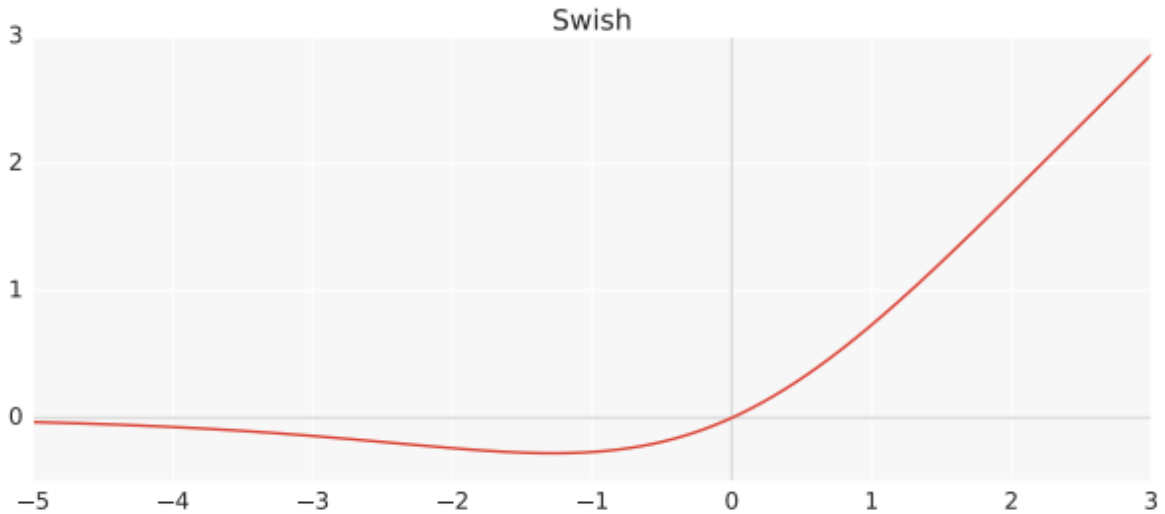
Leaky ReLu

- In ReLu, the gradient goes towards 0 for negative X : the weights will not get adjusted during descent.
- This is called dying ReLu problem. This problem can cause several neurons to just die and not respond making a substantial part of the network passive.
- There are variations in ReLu to mitigate this issue by simply making the horizontal line into non-horizontal component . for example y = 0.01x for x<0 will make it a slightly inclined line rather than horizontal line.
- The main idea is to let the gradient be non zero and recover during training eventually.





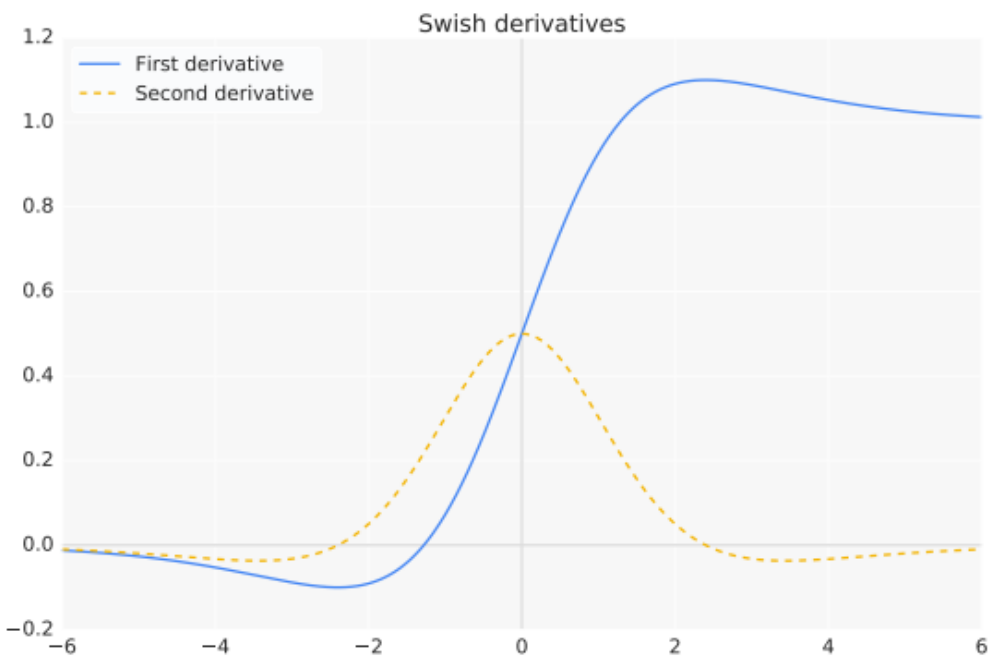
Swish



$$f(x) = x * (1 + \exp(-x))^{-1}$$

Swish

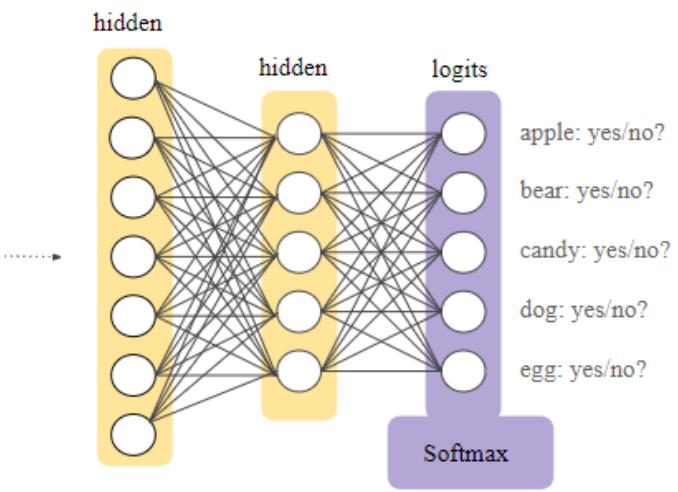
$$f'(x) = f(x) + \sigma(x)(1 - f(x))$$



Softmax

– Softmax function (or multinomial logistic regression) is a generalization of sigmoid function to the case where we want to handle multiple classes (multi-class classification).

$$S(y_i) = \frac{e^{y_i}}{\sum_{=1}^{i} e^{y_j}}$$



Softmax

$$S(y_i) = \frac{e^{y_i}}{\sum_{=1}^{i} e^{y_j}}$$

Properties of Softmax

– It normalizes your data (outputs a proper probability distribution),

– It is differentiable: “A "hardmax" function (i.e. argmax) is not differentiable. The softmax gives at least a minimal amount of probability to all elements in the output vector, and so is nicely differentiable, hence the term "soft" in softmax.

– It uses the exp -

Softmax

Example. There are 4 classes, and your log score *x* is vector [2, 4, 2, 1].

The simple argmax function outputs: [0, 1, 0, 0]

The argmax is the goal, but it's not differentiable and we can't train our model with it.

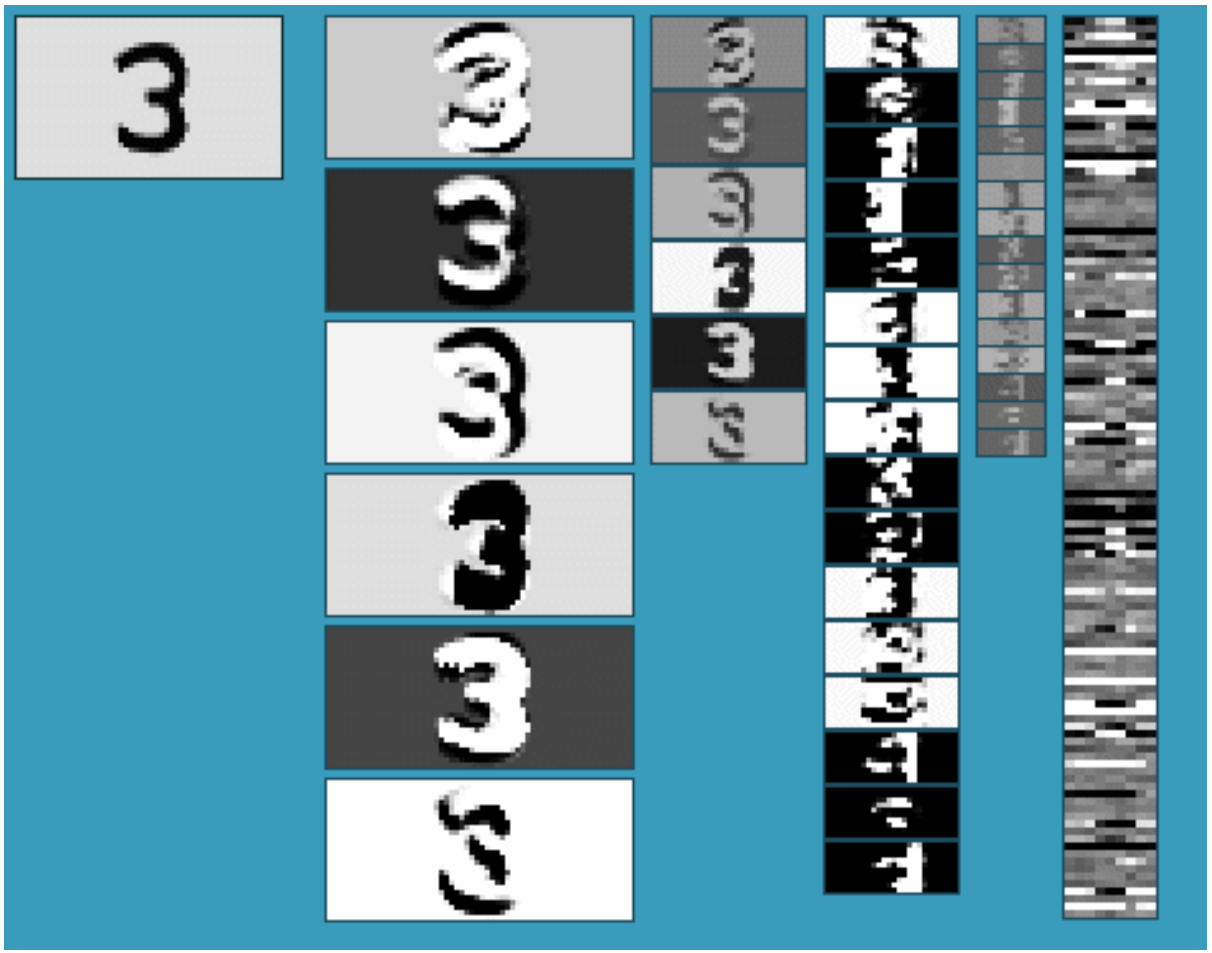
A simple normalization, which is differentiable, outputs the following probabilities: [0.2222, 0.4444 ,0.2222, 0.1111]
That's really far from the argmax!

Whereas the softmax outputs: [0.1025, 0.7573, 0.1025, 0.0377]
That's much closer to the argmax!

Because we use the natural exponential, we hugely increase the probability of the biggest score and decrease the probability of the lower scores when compared with standard normalization. Hence the "max" in softmax.

Convolutional Network (vintage 1990)

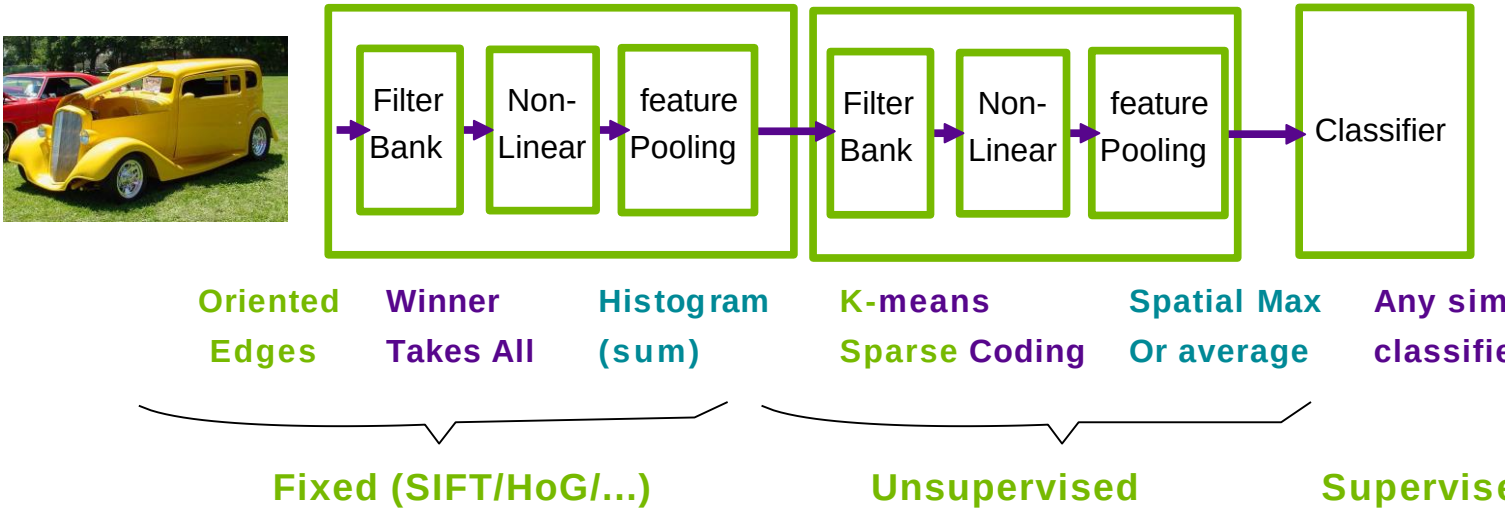
filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh



Curved manifold

Flatter manifold

“Mainstream” object recognition pipeline 2006-2012: somewhat similar to ConvNets



– Fixed Features + unsupervised mid-level features + simple classifier
  – SIFT + Vector Quantization + Pyramid pooling + SVM
    – [Lazebnik et al. CVPR 2006]
  – SIFT + Local Sparse Coding Macrofeatures + Pyramid pooling + SVM
    – [Boureau et al. ICCV 2011]
  – SIFT + Fisher Vectors + Deformable Parts Pooling + SVM
    – [Perronin et al. 2012]

Then., two things happened...

- The ImageNet dataset [Fei-Fei et al. 2012]
- Fast NVIDIA Graphical Processing Units (GPU)
  - Capable of 1 trillion operations/second

ImageNet large-scale visual recognition challenge

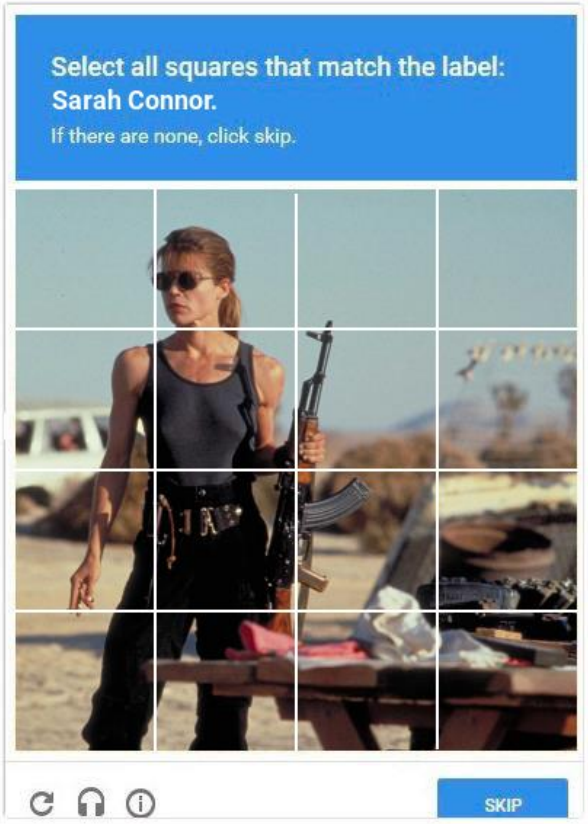
- The ImageNet dataset
  - 1.5 million training samples
  - 1000 fine-grained categories

There are 120 breeds of dogs in the training set!

Rob Lach
@lachrob

Follow

Hey @Google, exactly what kind of AI am I helping you guys train with this?



RETWEETS 3,903 LIKES 6,393

3:28 AM - 15 Feb 2017

106 3.9K 6.4K

Object Recognition
Top-N Error Rate

Top-N error rate is the **percentage** of the time that the classifier did not include the correct class among its **top** N guesses sorted by probability across all the classes

The **Top-1 error** is the **percentage** of the time that the classifier did not give the correct class the highest score.

Top-5 Example

1.Apple
2.Orange
3.Cherry
4.**Strawberry**
5.Plum

✓

1.Apple
2.Orange
3.Cherry
4.Plum
5.Cake

✗

Top-1 Example

1.**Strawberry**
2.Apple
3.Orange
4.Cherry
5.Plum

✓

1.Apple
2.**Strawberry**
3.Cherry
4.Plum
5.Cake

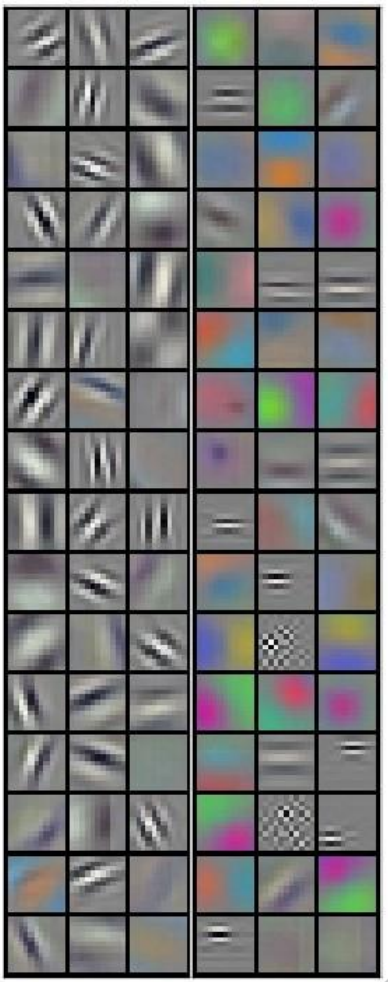
✗

Object Recognition
[Krizhevsky, Sutskever, Hinton 2012]

- Method: large convolutional net
  - 650K neurons, 832M synapses, 60M parameters
  - Trained with backprop on NVIDIA GPU
- Top-5 error rate: 15%
- Previous state of the art: 25%

- **A revolution in computer vision**

- Acquired by Google in Jan 2013
- Deployed in Google+ Photo Tagging in May 2013



Object Recognition
[Krizhevsky, Sutskever, Hinton 2012]

Albert Einstein

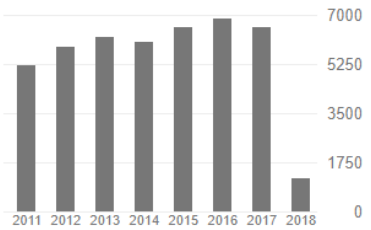
FOLLOW

Institute of Advanced Studies, Princeton
No verified email
Physics

TITLE	CITED BY	YEAR
Can quantum-mechanical description of physical reality be considered complete? A Einstein, B Podolsky, N Rosen Physical review 47 (10), 777	16638	1935
Uber einen die Erzeugung und Verwandlung des Lichtes betreffenden heurischen Gesichtpunkt A Einstein Ann. Phys. 17, 132-148	11025 *	1905

Cited by VIEW ALL

	All	Since 2013
Citations	115385	33579
h-index	106	62
i10-index	376	212



Alex Krizhevsky

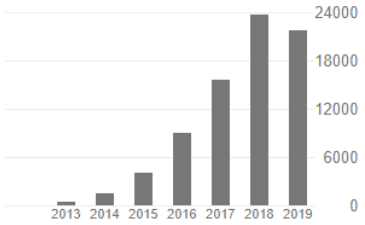
FOLLOW

Dessa
Verified email at dessa.com
Machine Learning

TITLE	CITED BY	YEAR
Imagenet classification with deep convolutional neural networks A Krizhevsky, I Sutskever, GE Hinton Advances in neural information processing systems, 1097-1105	49138	2012
Dropout: a simple way to prevent neural networks from overfitting N Srivastava, G Hinton, A Krizhevsky, I Sutskever, R Salakhutdinov The journal of machine learning research 15 (1), 1929-1958	15199	2014
Learning multiple layers of features from tiny images A Krizhevsky, G Hinton Technical report, University of Toronto 1 (4), 7	5291	2009

Cited by

	All	Since 2014
Citations	77791	76202
h-index	20	20
i10-index	20	20



Disclaimer: Number of citations is not a metric that could or should be used to compare the scientific quality or significance of the papers. This slide is only intended to demonstrate the **popularity** of deep learning.

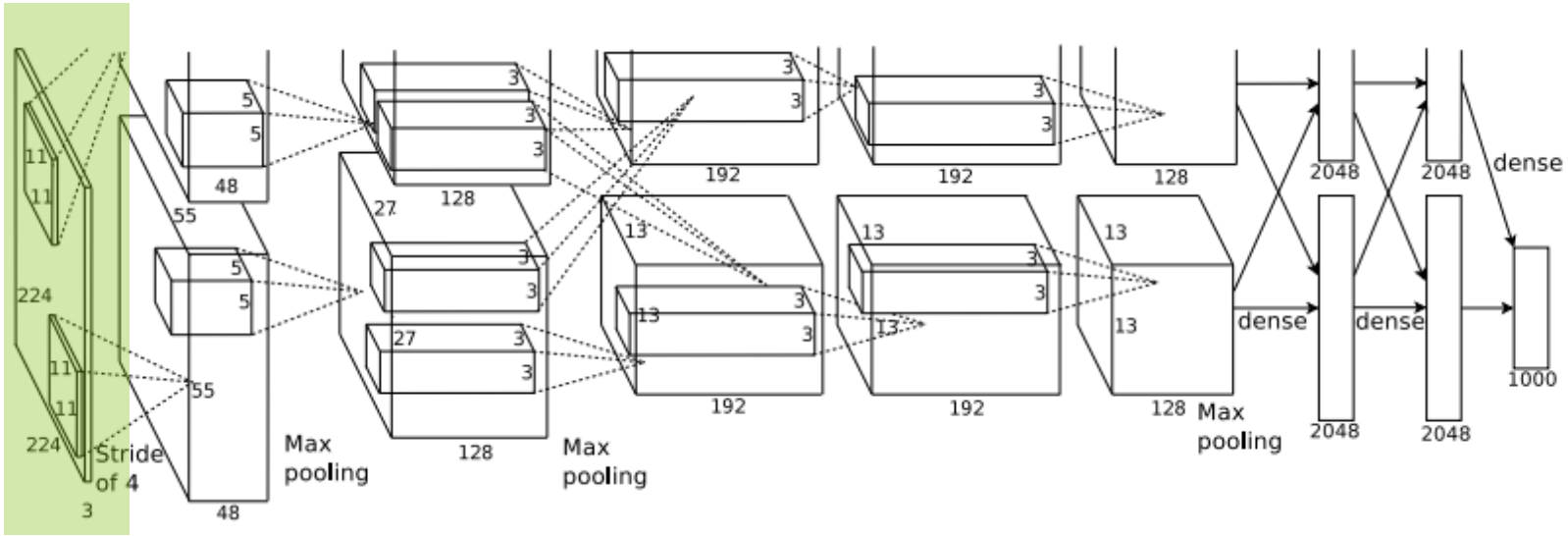
Object Recognition- AlexNet

[Krizhevsky, Sutskever, Hinton 2012]

- ImageNet consists of variable-resolution images, while the network requires a constant input dimensionality.
- Therefore, images are down-sampled to a fixed resolution of 256 × 256
- Given a rectangular image, image is first rescaled such that the shorter side was of length 256, and then cropped out the central 256×256 patch from the resulting image.
- No other pre-processing except for subtracting the mean activity over the training set from each pixel.
- So the network is trained on the (centered) raw RGB values of the pixels

Object Recognition- AlexNet
[Krizhevsky, Sutskever, Hinton 2012]

Won the 2012 ImageNet LSVRC. 60 Million parameters, 832M MAC ops

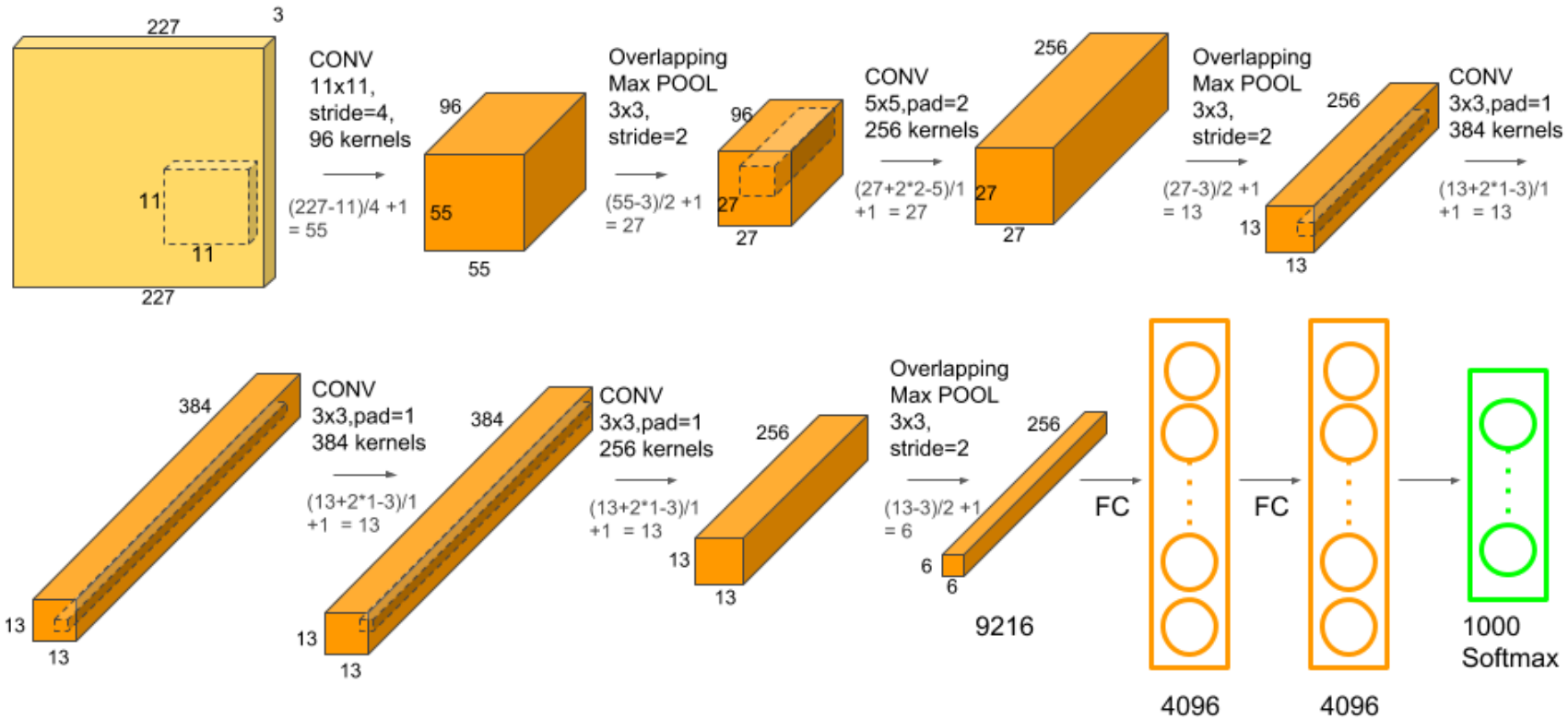


- The network's input is 150,528-dimensional (224x224x3)
- The first convolutional layer filters the 224×224×3 input image with 96 kernels of size 11×11×3 with a stride of 4 pixels
- Two GPUs are used: one GPU runs the layer-parts at the top of the figure (48 kernels) while the other runs the layer-parts at the bottom (48 kernels)

Note that the paper mentions the network inputs to be 224×224, but the numbers make sense with 227×227 instead.

Object Recognition- AlexNet

[Krizhevsky, Sutskever, Hinton 2012]



– ReLU is applied after all the convolution and fully connected layers.

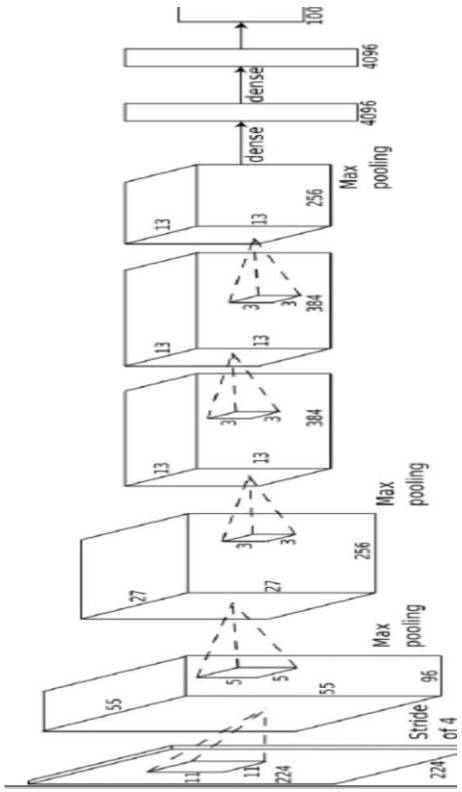
– The ReLU of the first and second convolution layers are followed by a local normalization step before doing pooling. But this was shown to be not very useful.

https://www.learnopencv.com/understanding-alexnet/

Object Recognition- AlexNet

[Krizhevsky, Sutskever, Hinton 2012]

4M	FULL CONNECT	4Mflop
16M	FULL 4096/ReLU	16M
37M	FULL 4096/ReLU	37M
	MAX POOLING	
442K	CONV 3x3/ReLU 256fm	74M
1.3M	CONV 3x3ReLU 384fm	224M
884K	CONV 3x3/ReLU 384fm	149M
	MAX POOLING 2x2sub	
	LOCAL CONTRAST NORM	
614K	CONV 5x5/ReLU 256fm	448M
	MAX POOL 2x2sub	
	LOCAL CONTRAST NORM	
35K	CONV 11x11/ReLU 96fm	105M



Number of parameters for first CONV
11x11x3x96 = 34848 ~ 35K

Number of flops for first CONV
34848 x ((227-11/4) + 1)^2 = 105415200 ~105M

Object Recognition- AlexNet

[Krizhevsky, Sutskever, Hinton 2012]

- Network architecture has 60 million parameters.

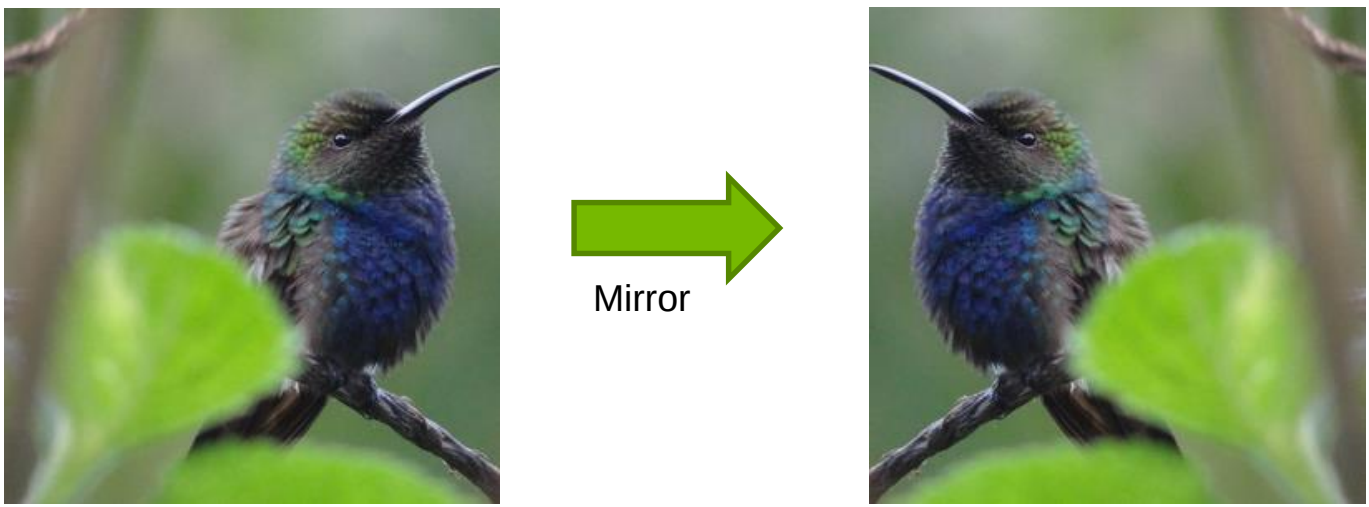
- Although the 1000 classes of ImageNet make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting.

- Two primary ways used to combat overfitting.
  - Data Augmentation
  - Dropout

Object Recognition- AlexNet
[Krizhevsky, Sutskever, Hinton 2012]

Data Augmentation

- The first form of data augmentation consists of generating image translations and horizontal reflections (mirror image).

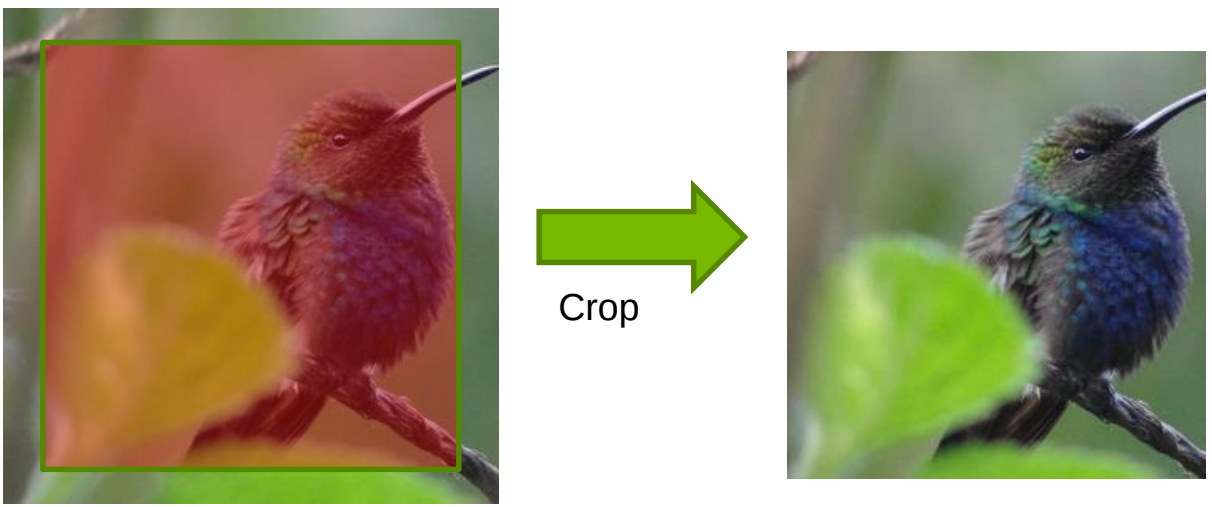


Object Recognition- AlexNet
[Krizhevsky, Sutskever, Hinton 2012]

Data Augmentation

- The first form of data augmentation consists of generating image translations and horizontal reflections.

- First, 224 × 224 patches (and their horizontal reflections) are extracted from the 256×256 images and training our network on these extracted patches.



Object Recognition- AlexNet
[Krizhevsky, Sutskever, Hinton 2012]

Data Augmentation

- The first form of data augmentation consists of generating image translations and horizontal reflections.
- First, 224 × 224 patches (and their horizontal reflections) are extracted from the 256×256 images.
- This increases the size of the training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent.
- Without this scheme, the network suffers from substantial overfitting, (which would have forced to use much smaller networks)
- At test time, the network makes a prediction by extracting five 224 × 224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

Object Recognition- AlexNet
[Krizhevsky, Sutskever, Hinton 2012]

Data Augmentation

- The second form of data augmentation consists of altering the intensities of the RGB channels in training images.

- **PCA Color Augmentation** is designed to shift those values based on which values are the most present in the image. Images with heavy red values and minimal green values will have their red values altered the most through PCA Color Augmentation.

- To each training image, multiples of the calculated principal components, with magnitudes proportional to the corresponding eigenvalues times a random variable drawn from a Gaussian with mean zero and standard deviation 0.1 are added.

Object Recognition- AlexNet
[Krizhevsky, Sutskever, Hinton 2012]

Dropout

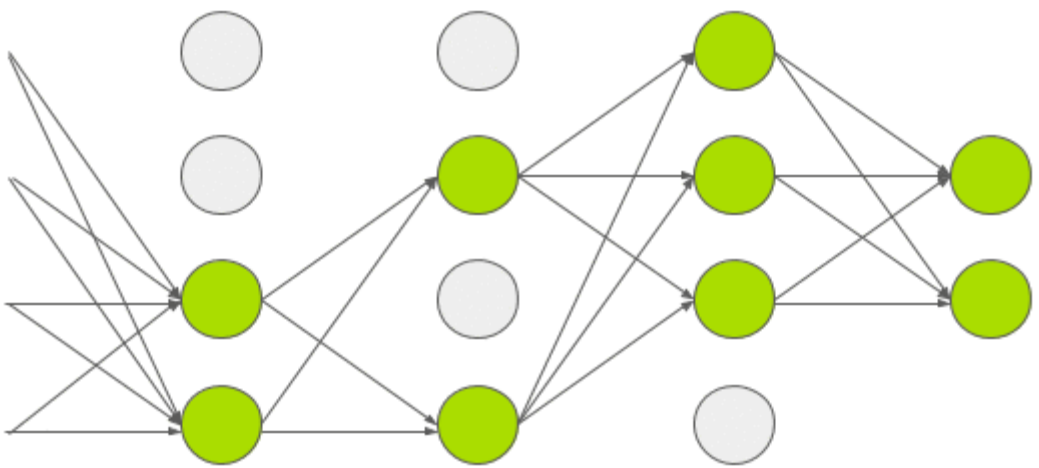
- Combining the predictions of many different models is a very successful way to reduce test errors, but it appears to be too expensive for big neural networks that already take several days to train.

- There is, however, a very efficient version of model combination that only costs about a factor of two during training.

- Dropout consists of setting to zero the output of each hidden neuron with probability 0.5. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in backpropagation.

Object Recognition- AlexNet
[Krizhevsky, Sutskever, Hinton 2012]

Dropout



Source: https://www.learnopencv.com/understanding-alexnet/

Object Recognition- AlexNet

[Krizhevsky, Sutskever, Hinton 2012]

Dropout

- So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.

- This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons.

- It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

Object Recognition- AlexNet

[Krizhevsky, Sutskever, Hinton 2012]

Dropout

- At test time, we use all the neurons but multiply their outputs by 0.5, which is a reasonable approximation to taking the geometric mean of the predictive distributions produced by the exponentially-many dropout networks.

- In AlexNet dropout is used in the first two fully-connected layers. Without dropout, the network exhibits substantial overfitting.

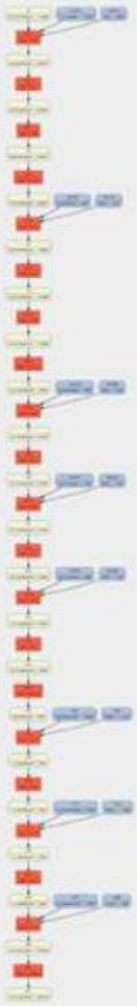
- Dropout roughly doubles the number of iterations required to converge

How Deep is Enough?

AlexNet (2012)

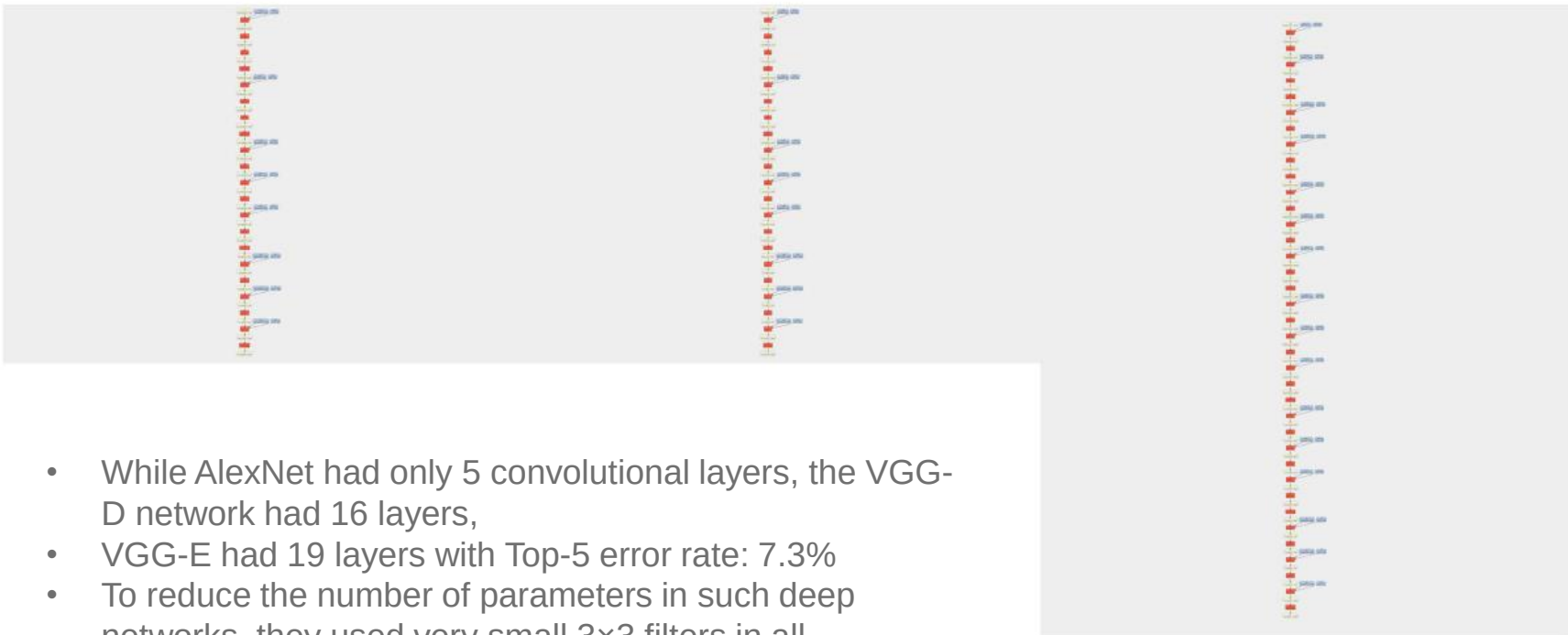
5 convolutional layers

3 fully-connected layers



How Deep is Enough?

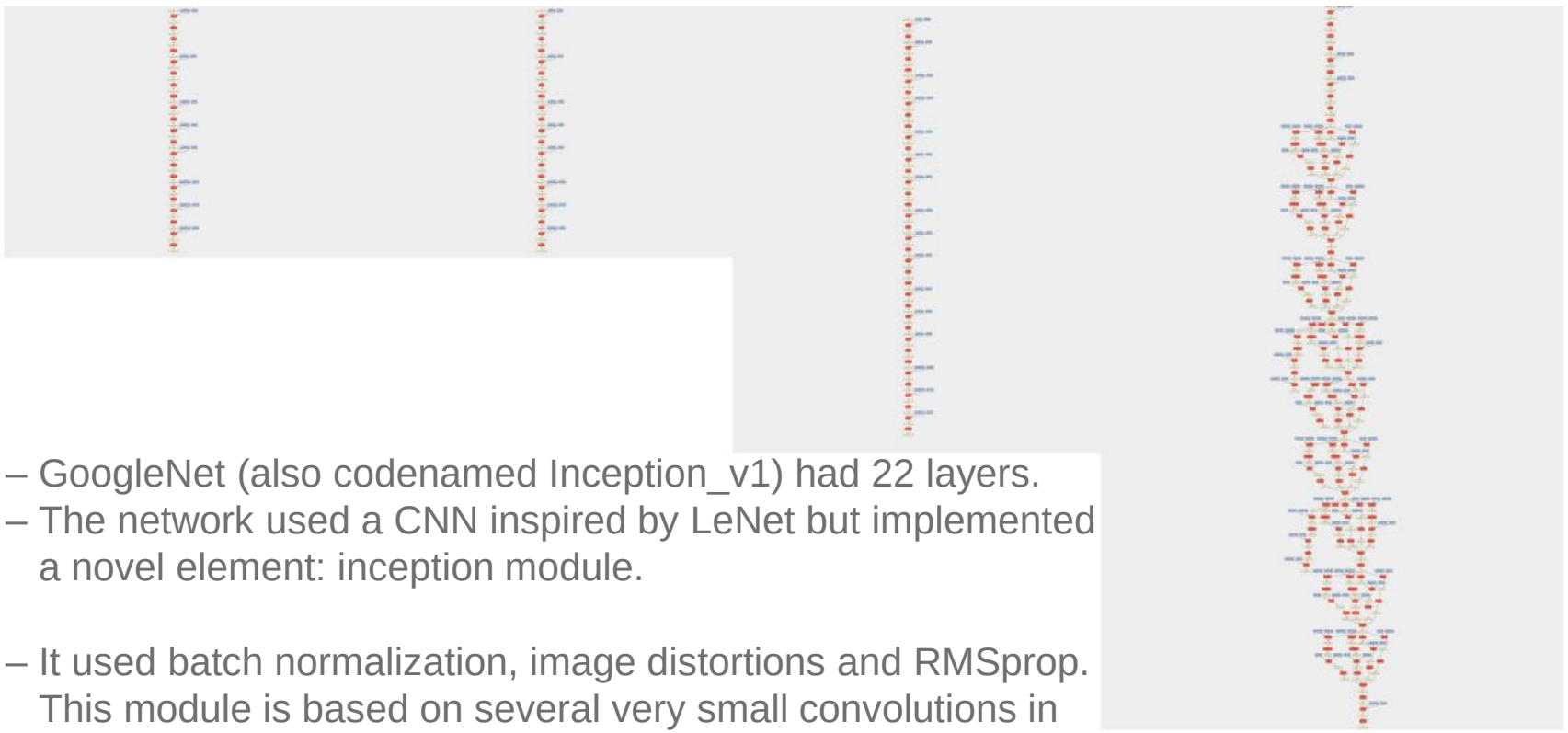
AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014)



- While AlexNet had only 5 convolutional layers, the VGG-D network had 16 layers,
- VGG-E had 19 layers with Top-5 error rate: 7.3%
- To reduce the number of parameters in such deep networks, they used very small 3×3 filters in all convolutional layers (the convolution stride is set to 1)

How Deep is Enough?

AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)

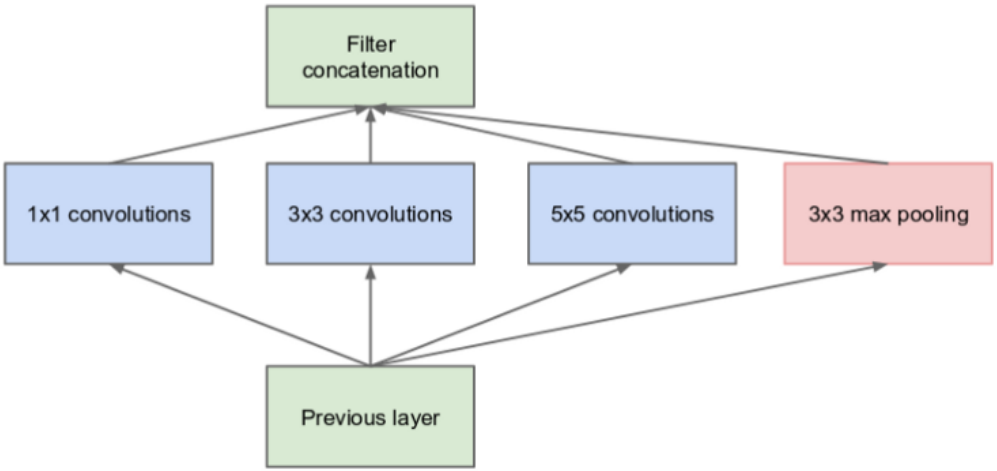


– GoogleNet (also codenamed Inception_v1) had 22 layers.
– The network used a CNN inspired by LeNet but implemented a novel element: inception module.

– It used batch normalization, image distortions and RMSprop. This module is based on several very small convolutions in order to drastically reduce the number of parameters.

– While their architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million.

Inception Module (I)

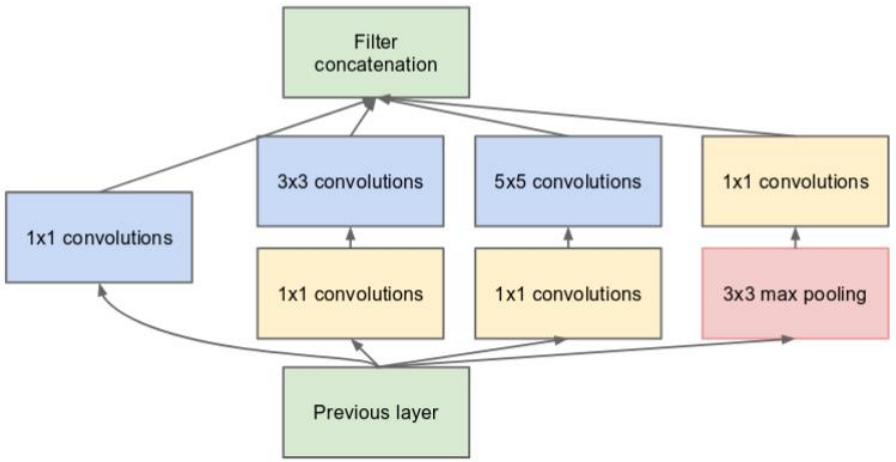


(a) Inception module, naïve version

- A way of reducing computational expense.
- Works by performing a convolution with three different sizes of filters (1x1, 3x3, 5x5) and a separate max pooling operation.
- The resulting outputs are concatenated and sent to the next layer.
- By structuring the CNN to perform its convolutions on the same level, the network gets progressively wider, not deeper.

https://deepai.org/machine-learning-glossary-and-terms/inception-module

Inception Module (II)



(b) Inception module with dimension reductions

- To make the process even less computationally expensive, the neural network can be designed to add an extra 1x1 convolution before the 3x3 ad 5x5 layers.
- So, the number of input channels is reduced before 3x3 or 5x5 convolutions.
- For the pooling part 1x1 convolution is added after the max-pooling operation.

https://deepai.org/machine-learning-glossary-and-terms/inception-module

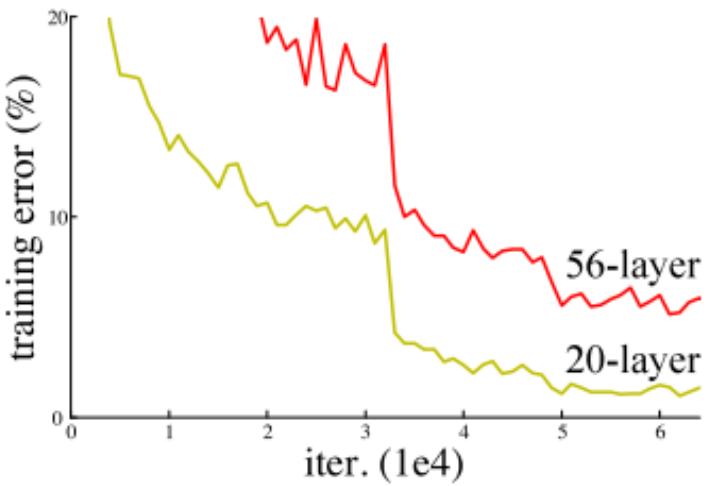
Inception Module (III)

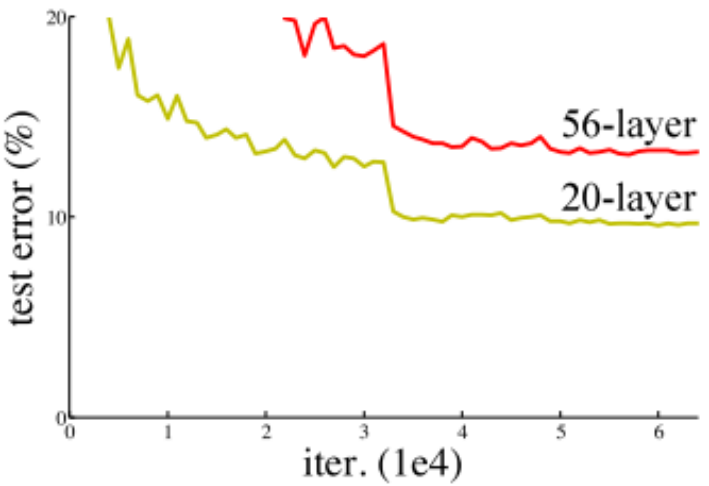
Filter Concatenation

type	patch size/ stride	output size	
convolution	$7\times7/2$	$112\times112\times64$	
max pool	$3\times3/2$	$56\times56\times64$	
convolution	$3\times3/1$	$56\times56\times192$	
max pool	$3\times3/2$	$28\times28\times192$	
inception (3a)		$28\times28\times256$	
inception (3b)		$28\times28\times480$	
max pool	$3\times3/2$	$14\times14\times480$	
inception (4a)		$14\times14\times512$	
inception (4b)		$14\times14\times512$	
inception (4c)		$14\times14\times512$	

- Different size convolutions are padded in a way to have the same output size.
- Pooling uses a stride of 1, so no down-sampling.
- So the resolution after the pooling layer also stays unchanged, and we can concatenate the pooling and convolutional layers together in the "depth" dimension.

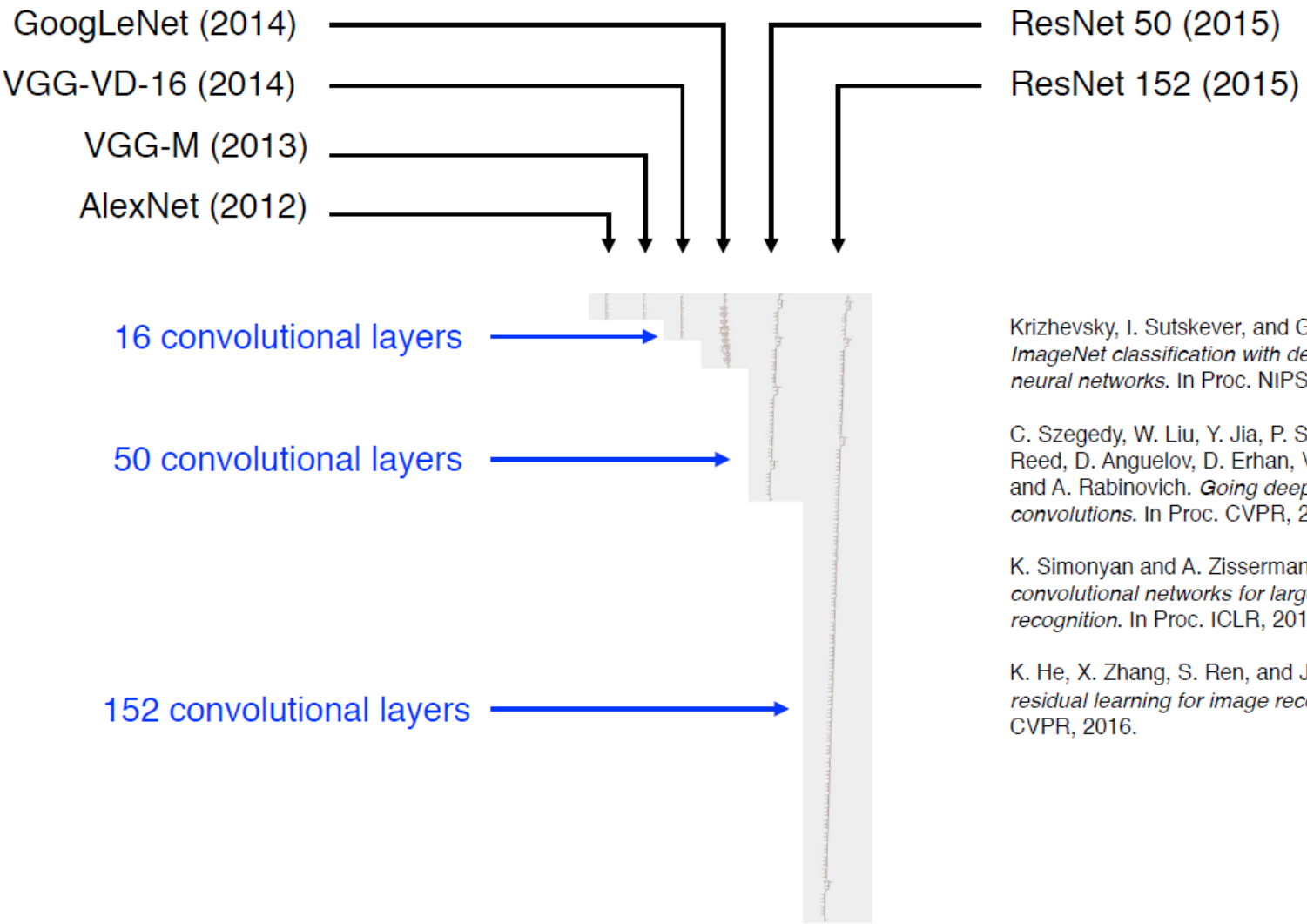
How Deep is Enough?





- Increasing network depth leads to worse performance!
- Vanishing gradients problem: gradients do not propagate through so many layers (they become smaller and smaller) to the earlier layers

How Deep is Enough?



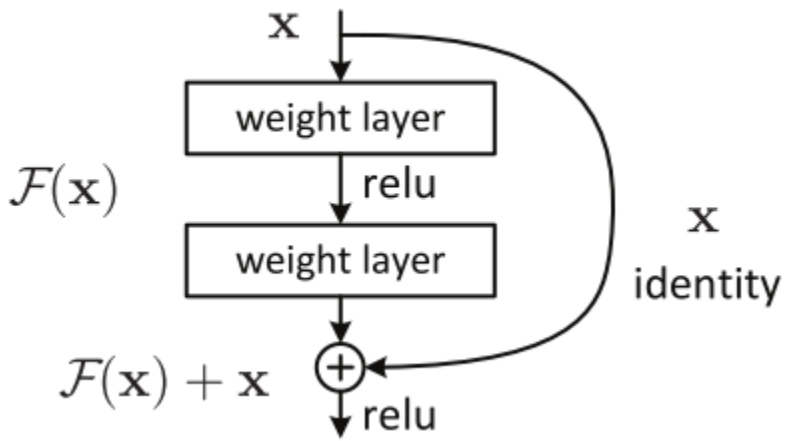
Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

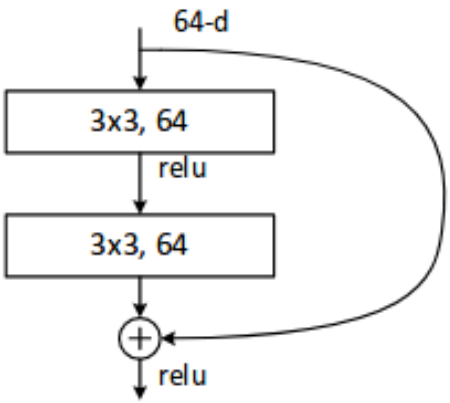
K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

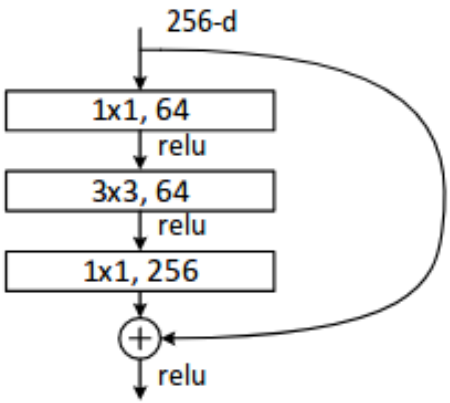
How Deep is Enough?



- The residual connection directly adds the value at the beginning of the block, x, to the end of the block (F(x)+x).
- This residual connection doesn't go through activation functions that "squashes" the derivatives, resulting in a higher overall derivative of the block.
- ResNet makes it possible to train up to hundreds or even thousands of layers

How Deep is Enough?

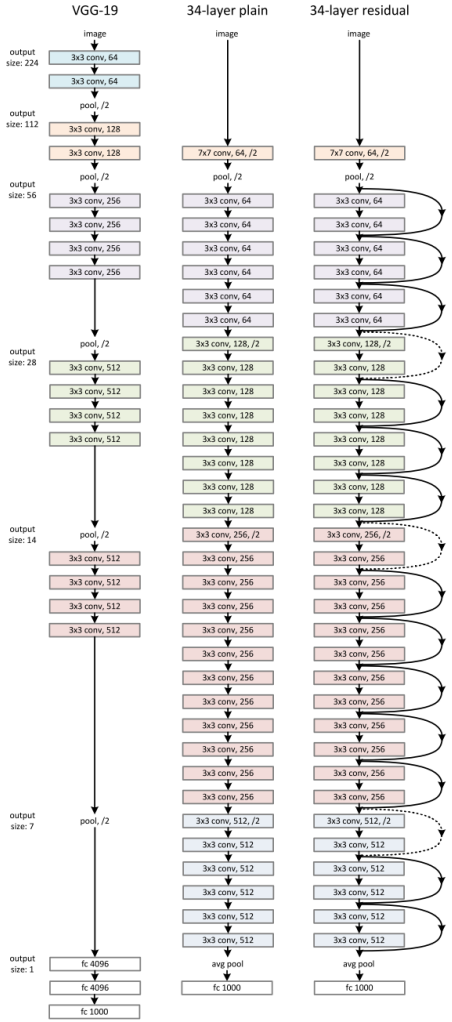




Each ResNet block could be:

- 2 layer deep (Used in smaller networks: ResNet 18, 34)
- 3 layer deep (ResNet 50, 101, 152).

How Deep is Enough?



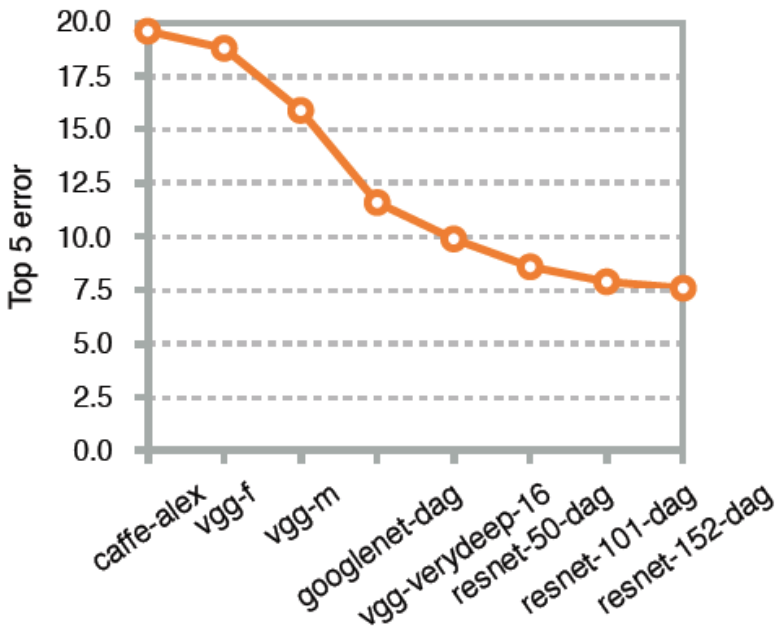
	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

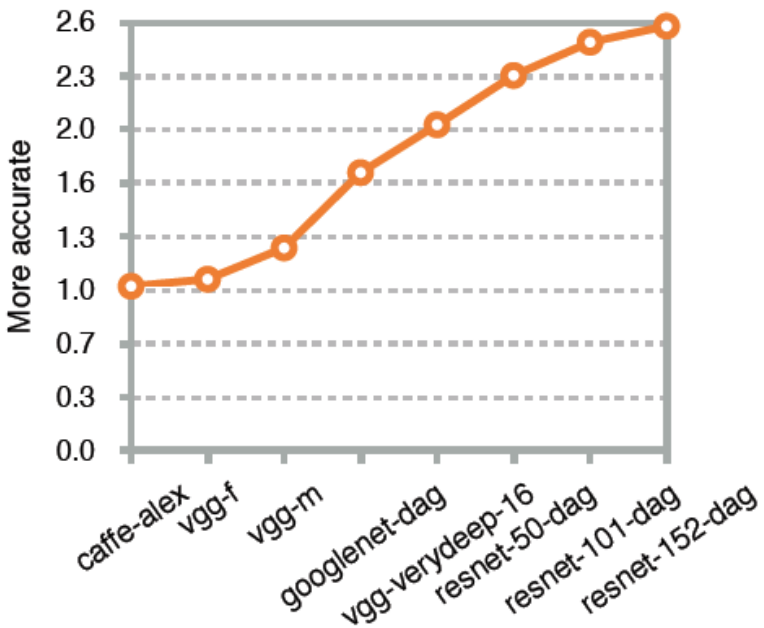
Table 2. Top-1 error (%, 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

How Deep is Enough?

Accuracy

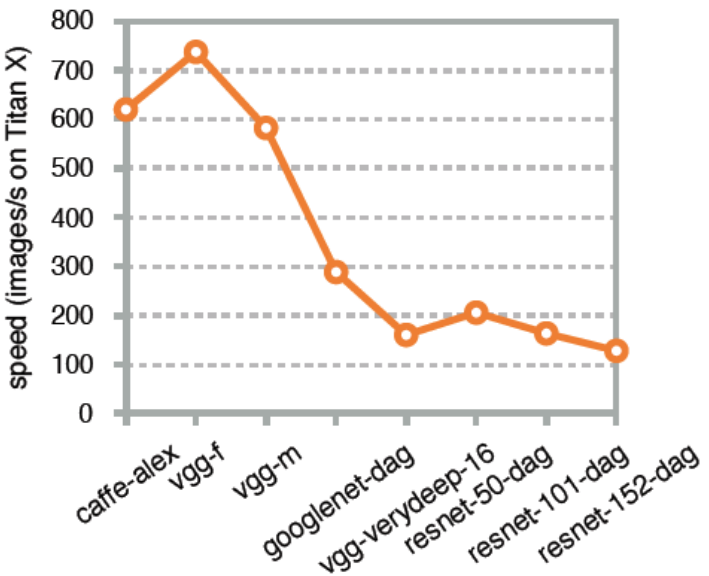
3 × more accurate in 3 years

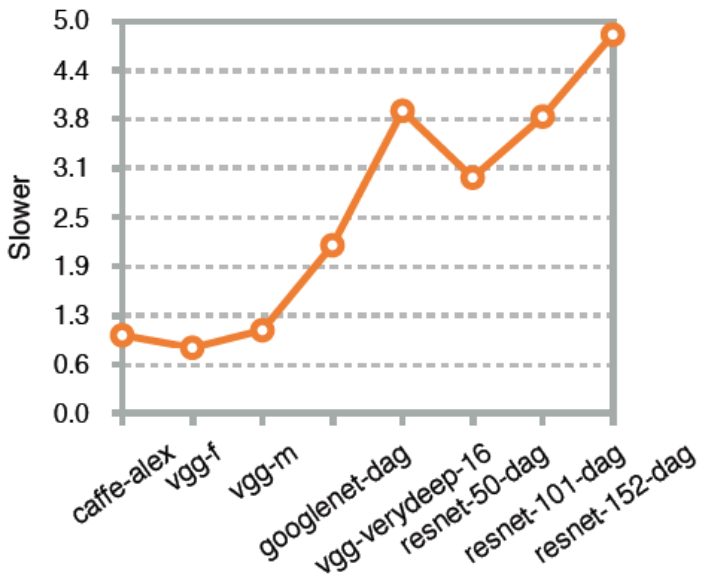




Speed

5 × slower





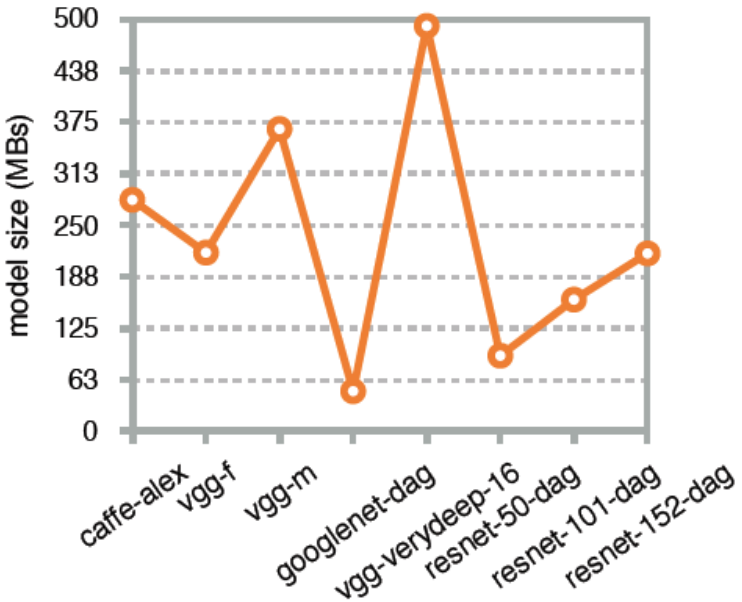
Remark: *101 ResNet* layers same size/speed as *16 VGG-VD* layers

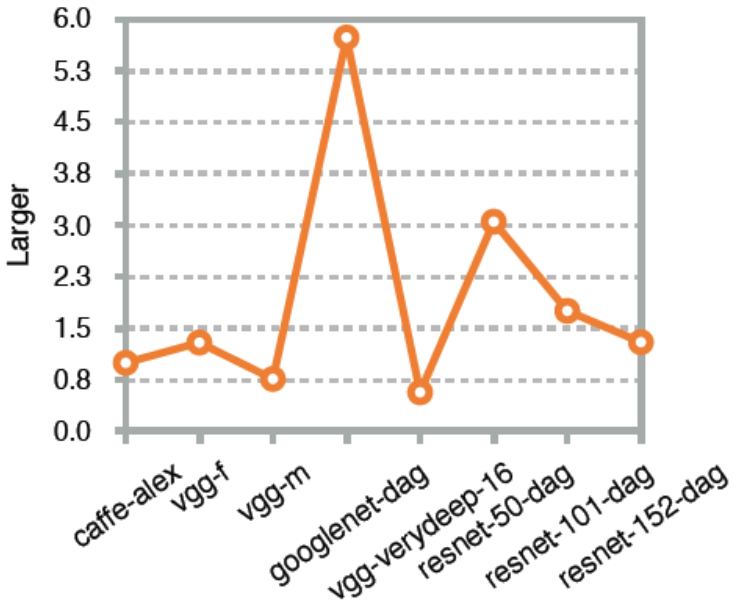
Reason: far fewer feature channels (quadratic speed/space gain)

Moral: optimize your architecture

Model size

Num. of parameters is about the same

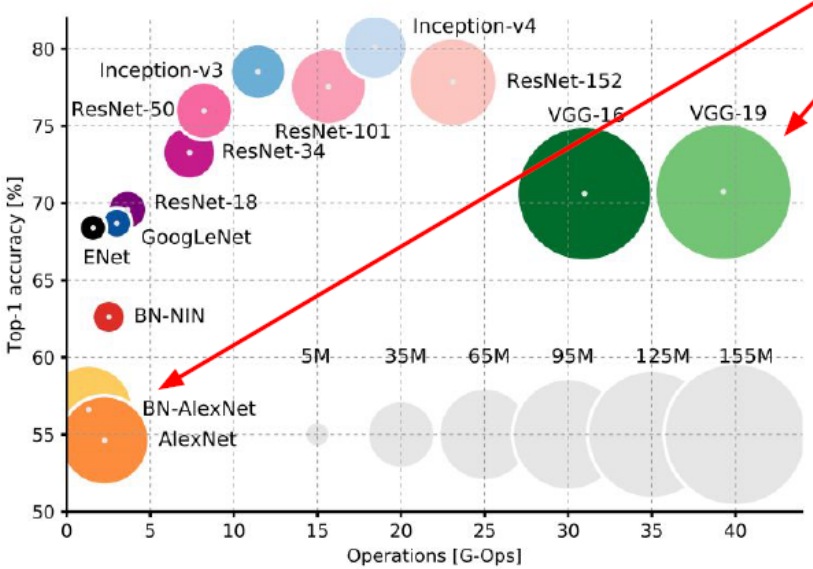




Remark: *101 ResNet* layers same size/speed as *16 VGG-VD* layers

Reason: far fewer feature channels (quadratic speed/space gain)

Moral: optimize your architecture



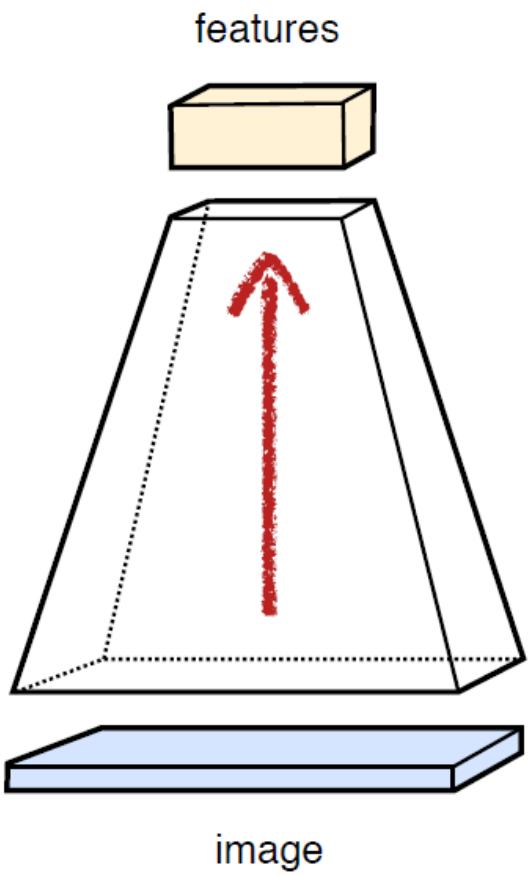
AlexNet and VGG have tons of parameters in the fully connected layers

AlexNet: ~62M parameters

FC6: 256x6x6 -> 4096: 38M params
FC7: 4096 -> 4096: 17M params
FC8: 4096 -> 1000: 4M params
~59M params in FC layers!

Design guidelines

Guideline 1: *Avoid tight bottlenecks*



From bottom to top

- The *spatial resolution* $H \times W$ decreases
- The *number of channels* C increases

Guideline

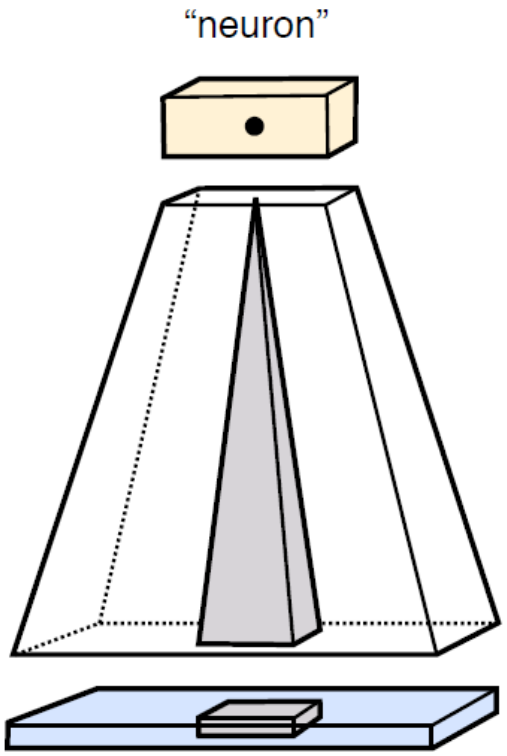
- Avoid tight information bottleneck
- Decrease the *data volume* $H \times W \times C$ slowly

K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens. *Rethinking the inception architecture for computer vision*. In Proc. CVPR, 2016.

Receptive field

Must be large enough



Receptive field of a neuron

- The image region influencing a neuron
- Anything happening outside is invisible to the neuron

Importance

- Large image structures cannot be detected by neurons with small receptive fields

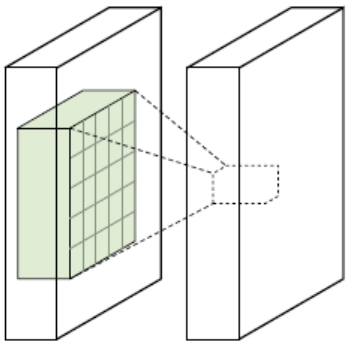
Enlarging the receptive field

- Large filters
- Chains of small filters

Design guidelines

Guideline 2: *Prefer small filter chains*

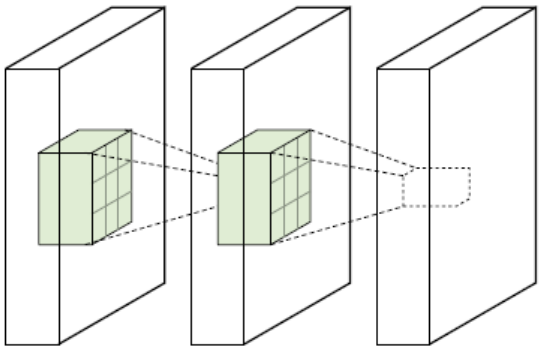
One big filter bank



5 × 5 filters
+ ReLU

prefer

Two smaller filter banks



3 × 3 filters
+ ReLU

3 × 3 filters
+ ReLU

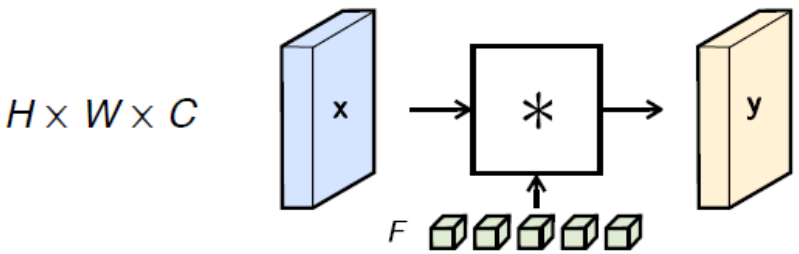
Benefit 1: less parameters, possibly faster

Benefit 2: same receptive field of big filter

Benefit 3: packs two non-linearities (ReLUs)

Design guidelines

Guideline 3: *Keep the number of channels at bay*



C = num. input channels

K = num. output channels

Num. of operations

$$\frac{H \times H_f}{\text{stride}} \times \frac{W \times W_f}{\text{stride}} \times C \times K$$

Num. of parameters

$$H_f \times W_f \times C \times K$$

complexity $\propto C \times K$