HE VD
IG

8. CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES

Stephan Robert-Nicoud
HEIG-VD/HES-SO

Credit: Andres Perez-Uribe

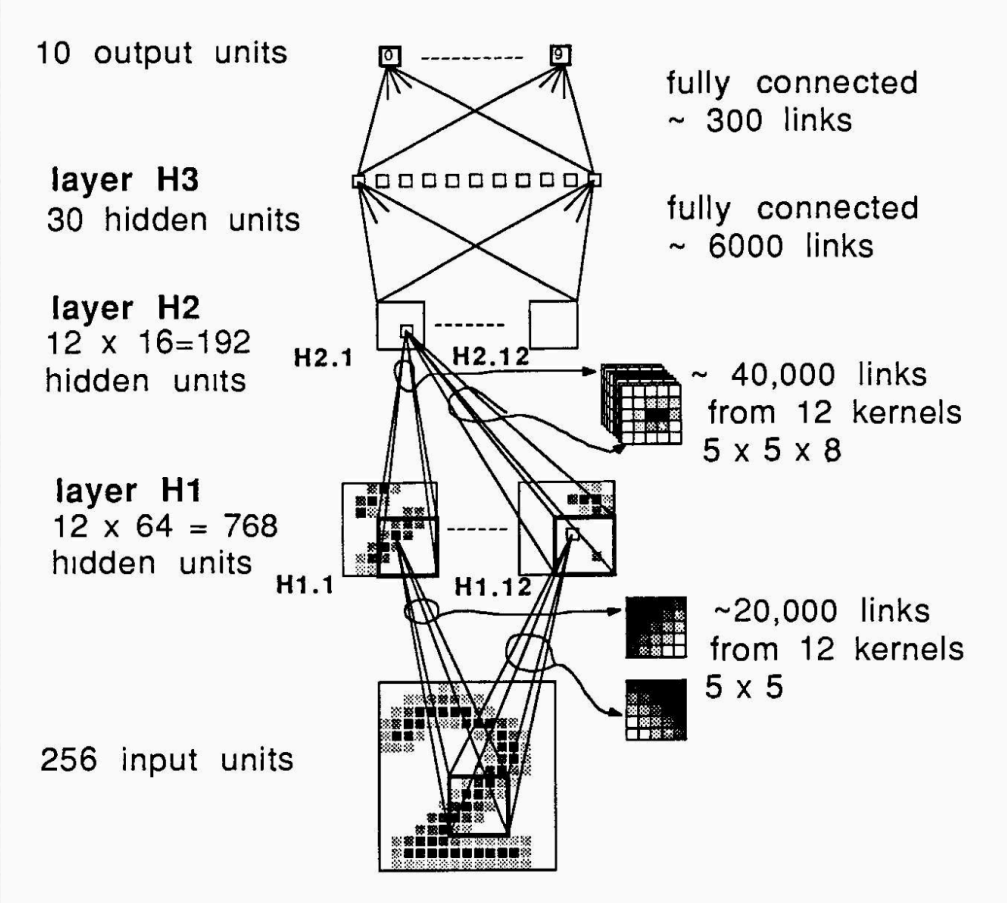
CC BY NC SA

Objectives

- Understand the evolution of the architectures of Convolutional Neural Networks
- Understand the motivation and the characteristics of the main architectures of Convolutional Neural Networks
- Train a Convolutional Neural Network for an image recognition problem

LeNet-5 (1989)

- By Yann LeCun and colleagues (AT&T Bell Labs)

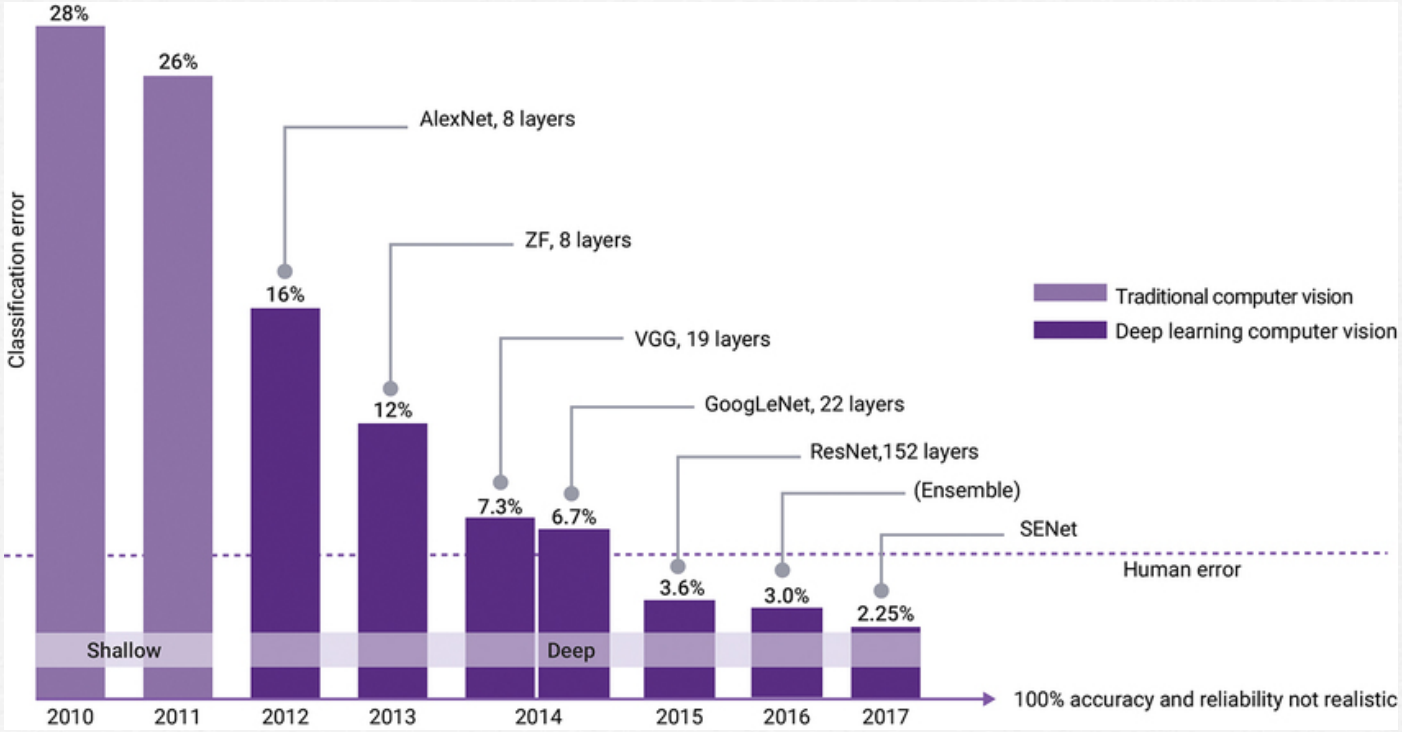


- "Backpropagation applied to Handwritten Zip code Recognition", Neural Computation 1
- "Large networks trained by Backpropagation can be applied to real image-recognition problems without complex pre-processing requiring detailed engineering"
- Trained on 9298 digits from US mail during 3 days on a Sun Sparc Station 1

from http://yann.lecun.com/exdb/lenet/

The ILSVRC challenge

ImageNet Large Scale Visual Recognition Challenge

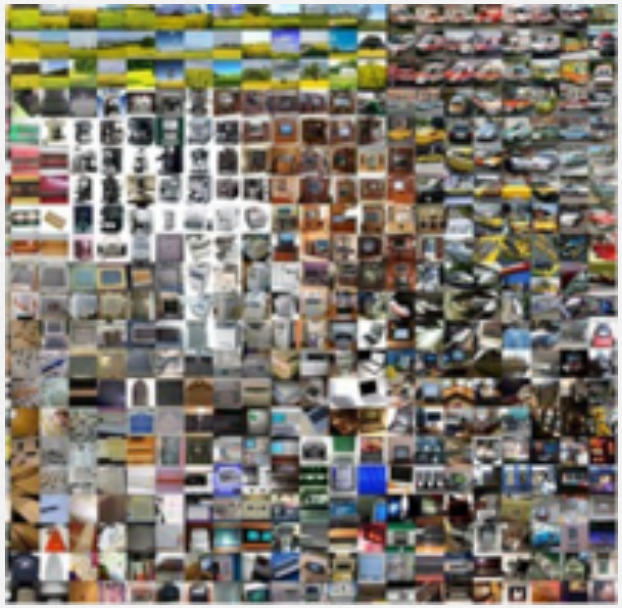


from synopsys.com

ImageNet: 15 million images (22k categories), human labeled

ILSVRC Challenge: ~1000 images of 1000 categories (top-1/top-5 error rates)

ImageNet

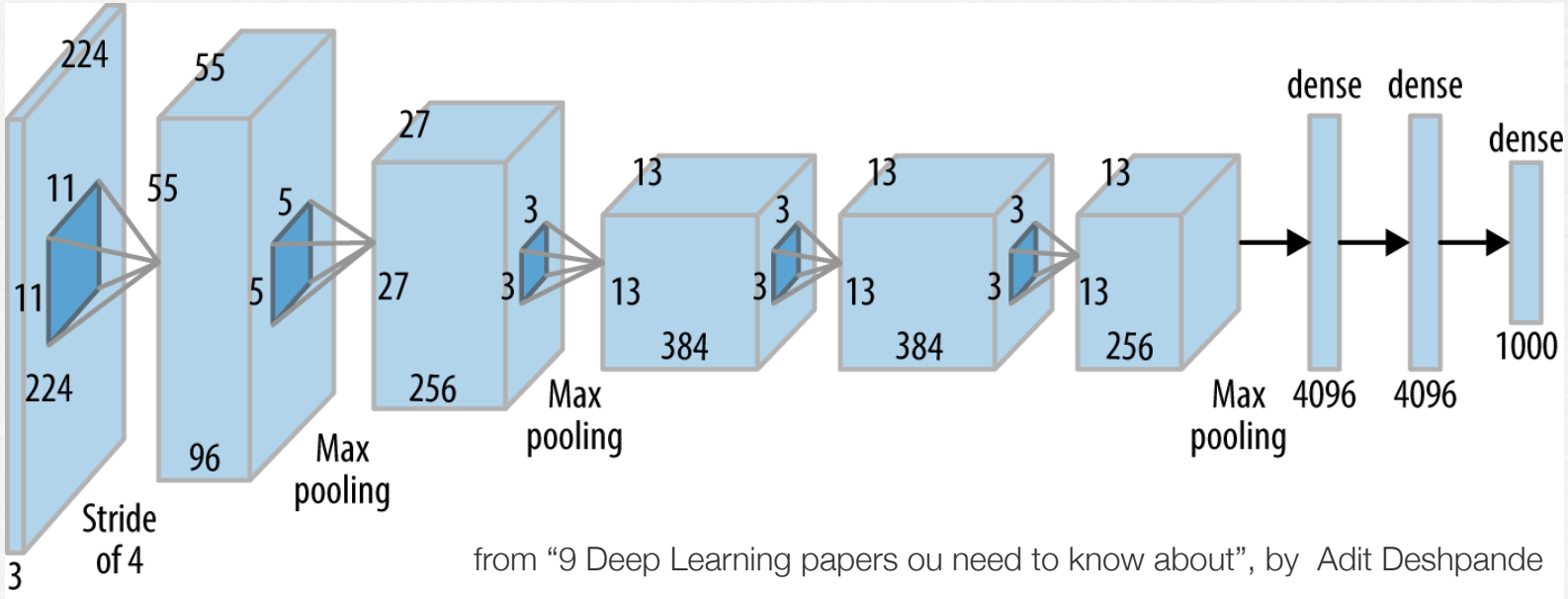


- 2007-2010 human labeling using Amazon Mechanical Turk
- 49K workers from 167 countries
- Human performance: ~5% (top-5 error) due to class unawareness, need for fine-grained recognition

from https://www.image-net.org/static_files/files/imagenet_ilsvrc2017_v1.0.pdf

AlexNet (2012)

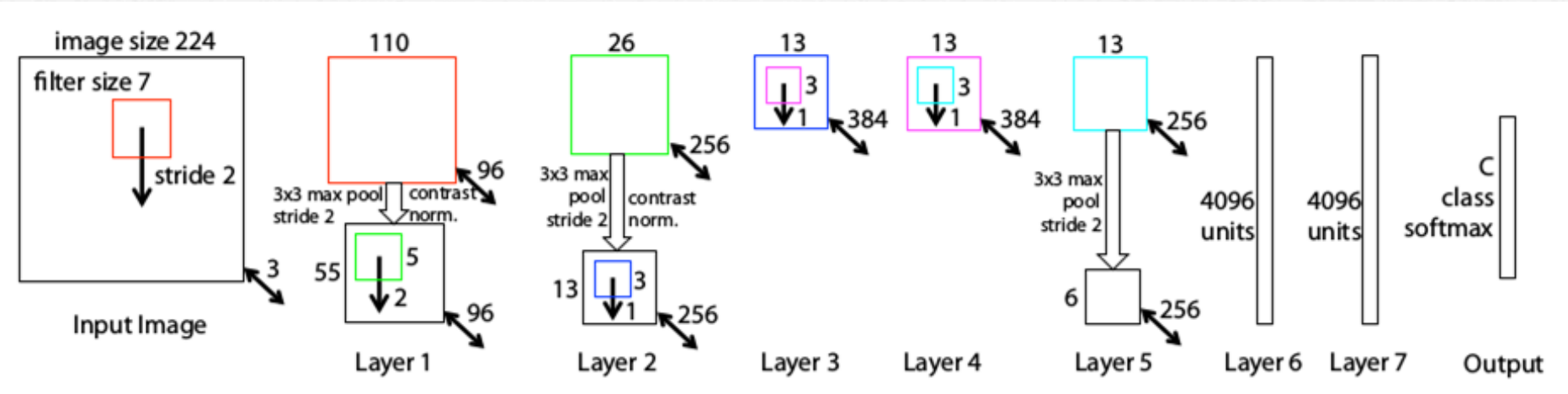
- By Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton
- 62-million parameters model trained on ImageNet data using batch stochastic gradient descent, with specific values for momentum and weight decay. Trained on two GTX 580 GPUs for 5 to 6 days.
- Used ReLus (instead of tanh), data augmentation and dropout.
- Won the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge): error = 15.4% (second place 26.2%).



from "9 Deep Learning papers ou need to know about", by Adit Deshpande

ZF Net (2013)

- by Matthew Zeiler and Rob Fergus from NYU
- Trained on only 1.3M images (ImageNet) using batch stochastic gradient descent to minimize cross-entropy.
- Used smaller kernel sizes (7x7) and increasing number of filters
- Trained on two GTX 580 GPUs for 12 days.
- Won the 2013 ILSVRC: error = 11.2%

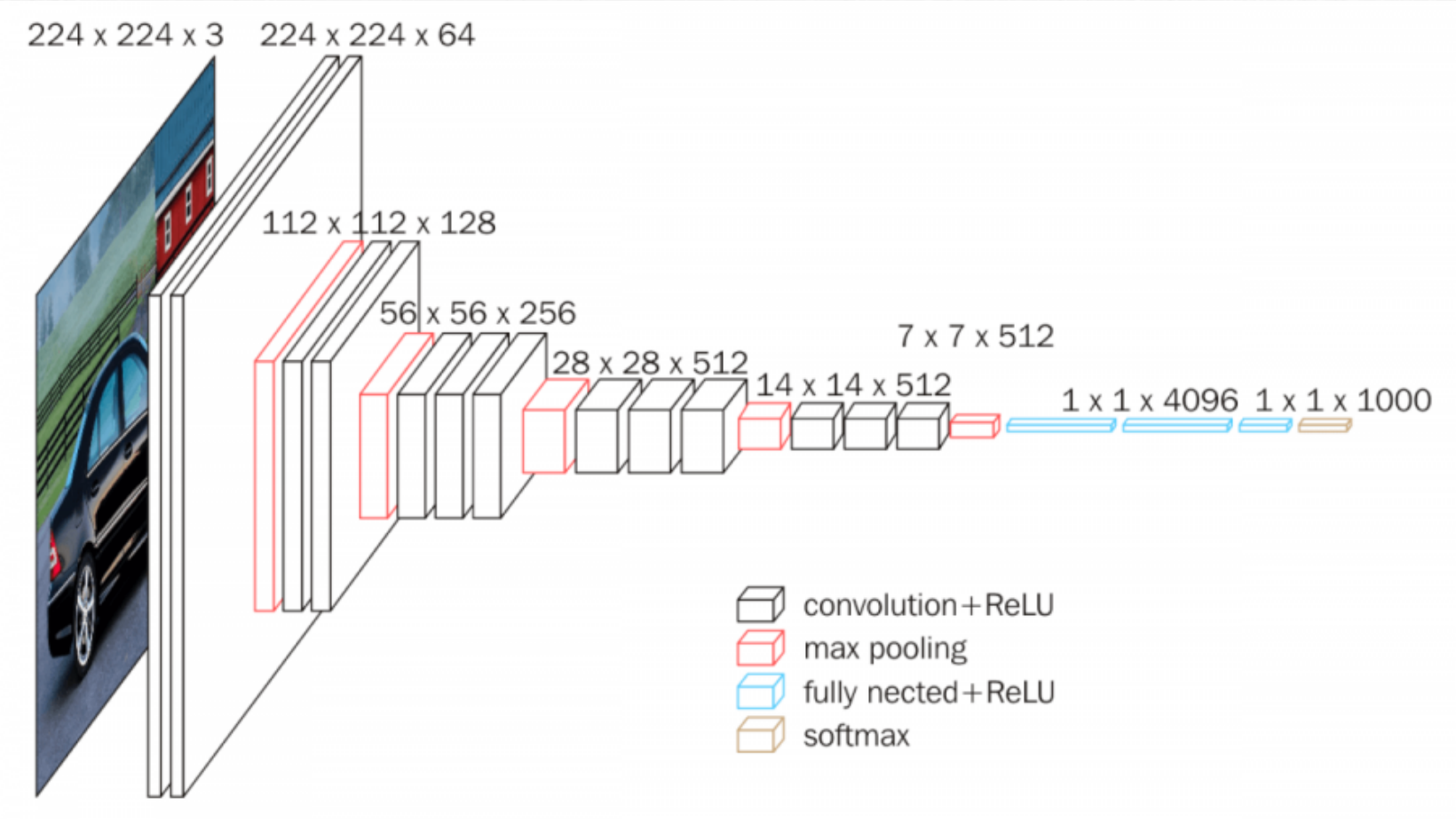


from "9 Deep Learning papers ou need to know about", by Adit Deshpande

VGG Net (2014)

- by K. Simonyan and A. Zisserman from the Visual Geometry Group of the University of Oxford
- VGG-16 is a 16-layer CNN and VGG-19 is a 19-layer CNN that strictly used 3x3 filters with stride and pad of 1, along with 2x2 maxpooling layers with stride 2
- Number of kernels: 64, 128, 256, 512, 512
- Did not win ILSVRC (error 7.3%) but introduced a simple architecture
- Trained on 4 Nvidia Titan Black GPUs for two to three weeks
- VGG-16 has more than 138-million parameters and VGG-19 has almost 143-million parameters.

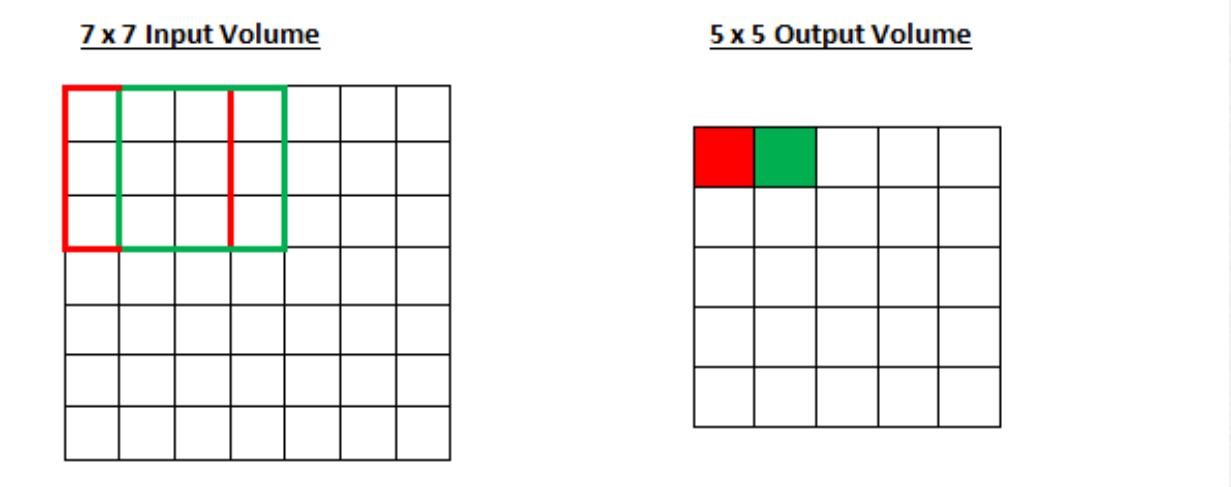
Ideas behind VGG Nets



https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

Downsample input with stride

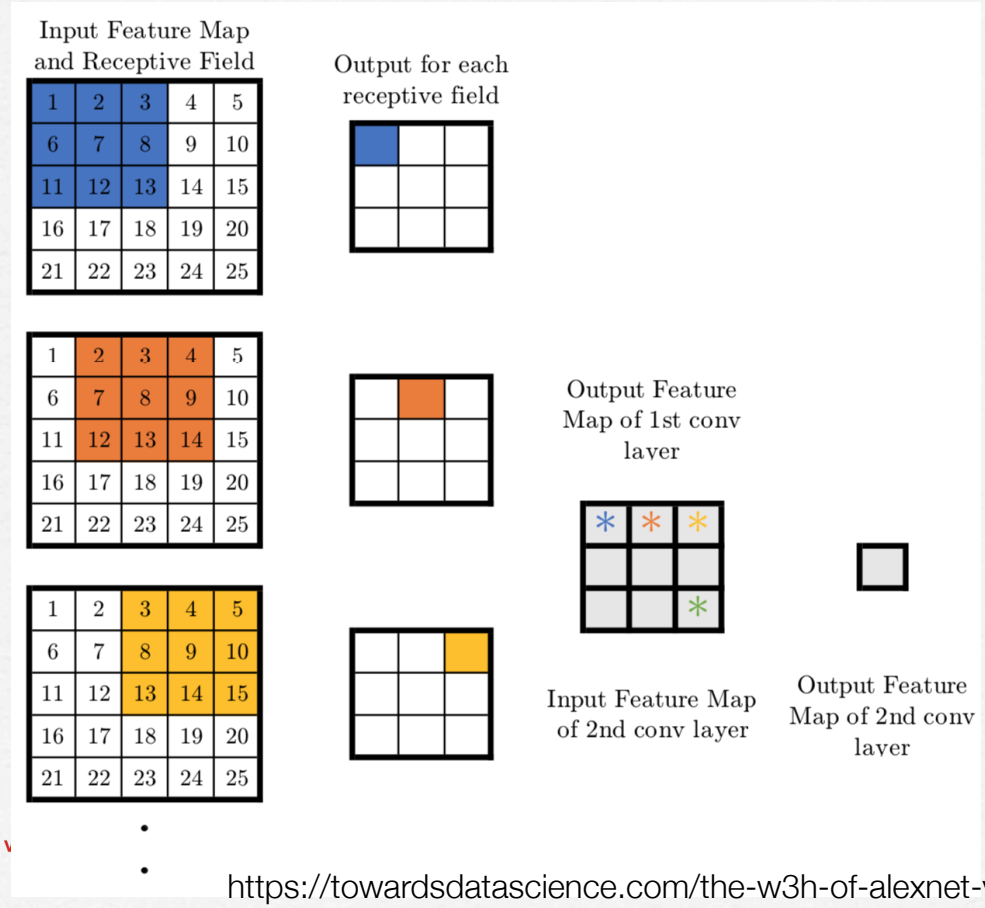
- Stride: the amount of movement between applications of the filter to the input image.
- Example: no padding and stride=1



The result is a downsampled image at the output, e.g., 5x5 instead of 7x7.

Ideas behind VGG Nets (2)

The Alexnet convolutional kernels of sizes 11x11, 5x5, and 3x3 can be replicated by making use of multiple 3x3 kernels as building blocks, but reducing the number of parameters to learn.

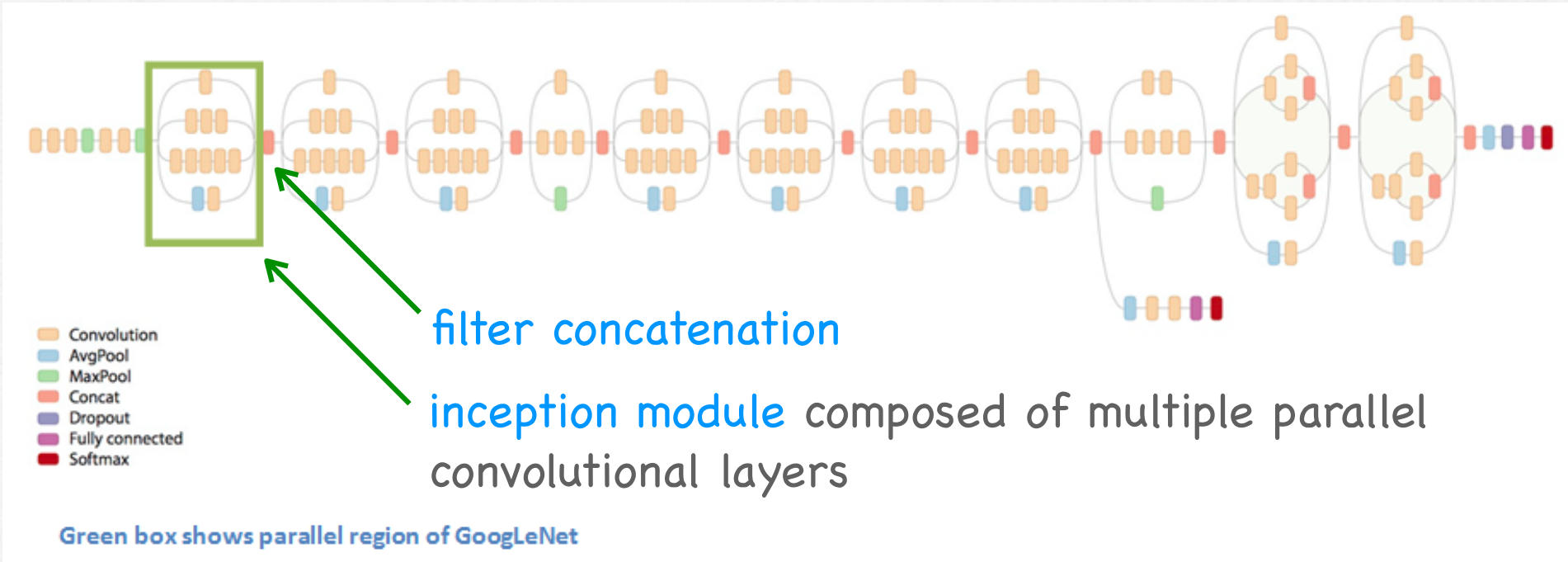


- Suppose a 5x5x1 input. A 5x5 kernel (25 parameters) will produce a 1x1 output
- This can be computed by two layers of 3x3 kernels, stride=1 too (2 x 9 parameters)
- Similarly a 11x11 filter (121 parameters) can be computed by five 3x3 kernels, stride=1 (45 parameters)

https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

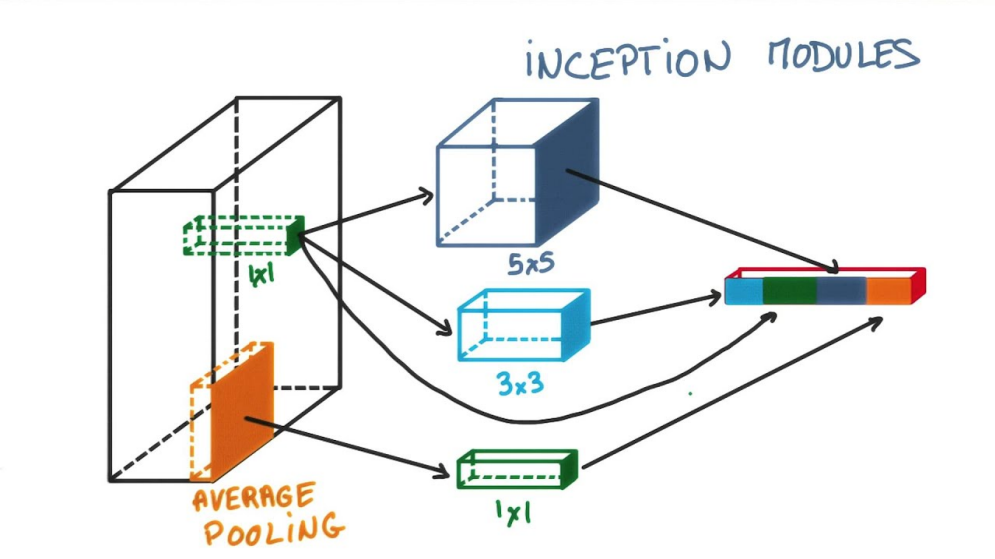
GoogLeNet (2015)

- Winner of ILSVRC'14 (error 6.7%)
- A new architecture with more than 100 layers in total, but not necessarily stacked up sequentially and without fully-connected layers. It uses 12x less weights than AlexNet

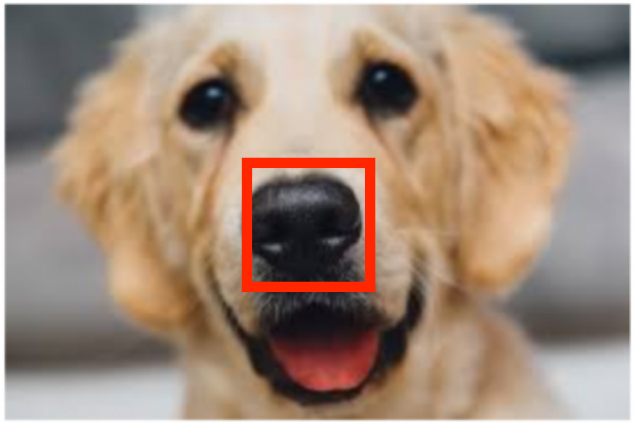


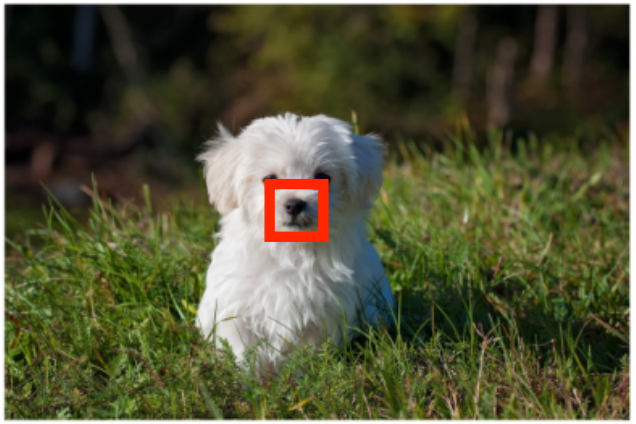
from "9 Deep Learning papers ou need to know about", by Adit Deshpande

Ideas behind Inception Networks (1)



- Instead of choosing a single size (1x1, 3x3 or 5x5) kernel for a layer, use them "all" and concatenate the outputs

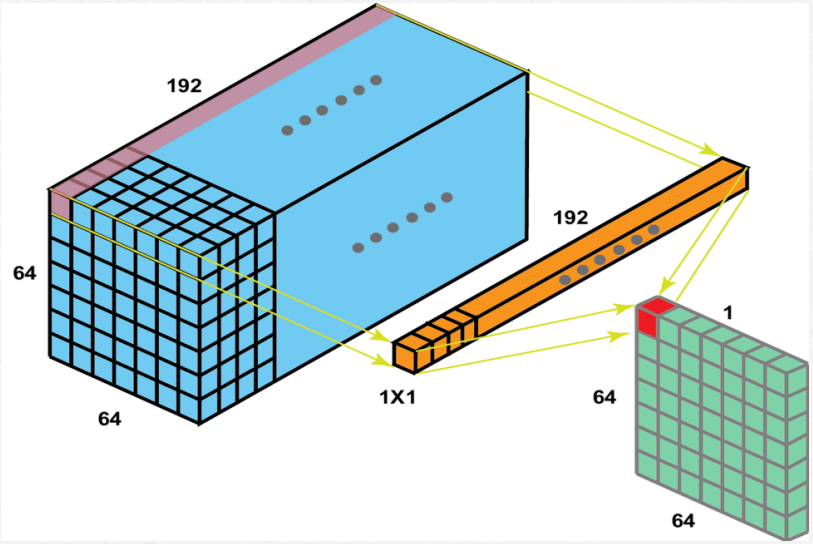




- different kernel sizes allows the identification of features at different scales

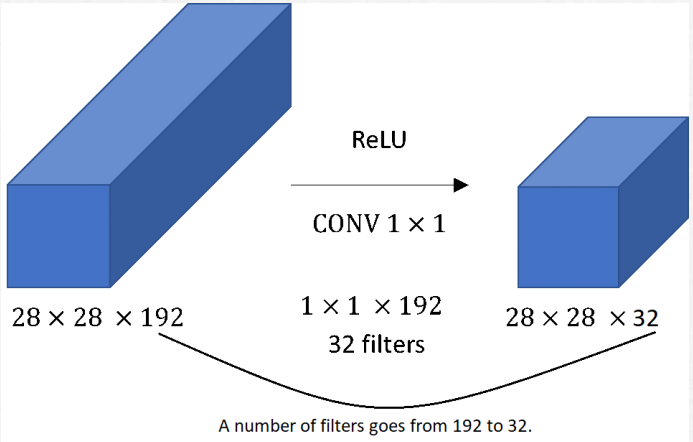
Ideas behind Inception Networks (2)

a)



- Use 1x1 convolutions (Fig a) to perform feature pooling or dimensionality reduction.

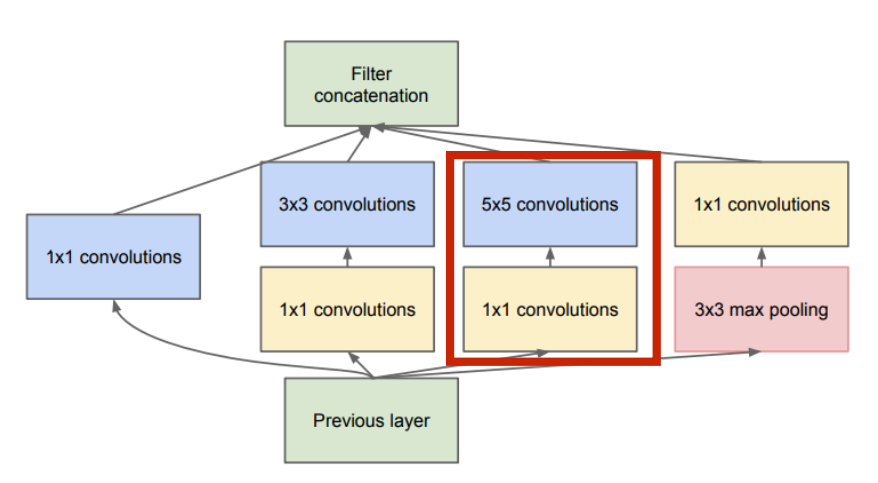
b)



- Example: applying 32 1x1x192 filters to the 192 28x28 images in the figure b) produces 32 28x28 images

from https://www.youtube.com/watch?v=C86ZXvgpejM (by A. Ng)

Ideas behind Inception Networks (3)



- Use 1x1 convolutions before 3x3 or 5x5 convolutions to perform feature pooling and reduce the number of operations.

- Example: applying 16 1x1x192 filters to the incoming 192 28x28 images before applying 32 5x5 filters requires 12.4M multiplications instead of ~120M multiplications without the 1x1 convolutions.

from https://www.youtube.com/watch?v=C86ZXvgpejM (by A. Ng)

Batch-normalization (1)

Sergey Ioffe and Christian Szegedy from Google (2015)

- Problem: when two inputs are in completely different scales, say a range of x1 is [1000-2000] and range of x2 is [0.1-0.5], using them as-is has implications on optimizing the loss function (e.g., by gradient descent). Intuitively, the model will tend to be more influenced by x1.

- To deal with this problem, we generally normalize inputs before using a neural network. Normalization, in general refers to squashing a diverse range of numbers to a fixed range.

- Ioffe and Szegedy did the same analysis within the network and noticed that the learning was enhanced when the inputs to a subsequent sub-network or layer are also normalized.

Batch-normalization (2)

A "learnable" normalization transform that is applied to every mini-batch during the learning process:

start by normalizing the inputs (per dimension) per batch (i.e., by setting the mean to zero and the standard deviation to 1), but then use the learnable parameters γ and β to scale and shift the input data. γ and β are learned over all the database and serve to restore the expressiveness of the network.

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

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

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

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

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

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch-normalization (3)

- It has been shown that batch normalization accelerates training convergence (i.e., less learning steps are required), allows larger learning rate values, the training of models using saturating nonlinearities (e.g., sigmoids), and not needing dropout.

- It was originally proposed to be used before the non-linearity (e.g., relu), but it has been reported to work even better after it.

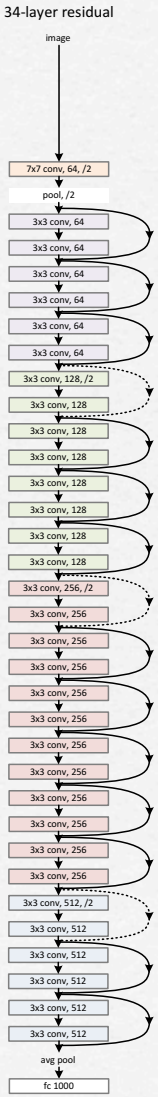
MLP model

```
# example of batch normalization for an mlp
from keras.layers import Dense
from keras.layers import BatchNormalization
...
model.add(Dense(32, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(1))
...
```

CNN model

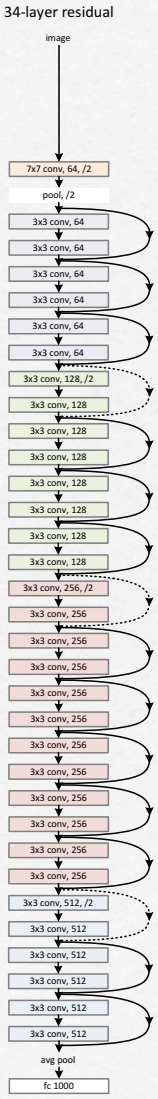
```
# example of batch normalization for an cnn
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import BatchNormalization
...
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dense(1))
...
```

ResNet (2015)



- A degradation of performance was observed when going deeper (e.g., from 20 to >50 layers) on the ImageNet benchmark.
- A team from Microsoft won ILSVRC'15 (error: 3.6%) using a 152-layer CNN architecture exploiting « shortcut connections ».
- These so-called ResNet networks appear to have a better performance than plain networks with the same number of weights (ResNet152V2 has 60-million weights). Training took 3 weeks using 8 GPUs.

ResNet's residual blocks



- The residual block assumes that it is easier to learn a mapping from x -> F(x) + x than from x -> to a new function H(x)
- A similar architecture called "Highway networks" was proposed by Schmidhuber et al. He claims to have successfully trained the first deep neural network with more than 100 layers.
- The team from Microsoft tested a 1202-layer network that performed less well.

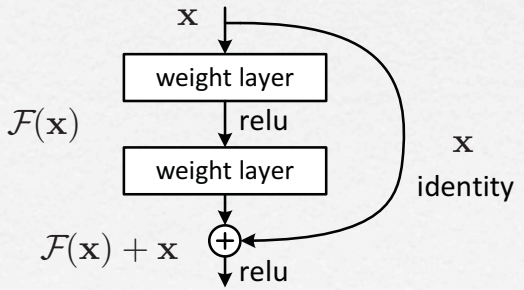
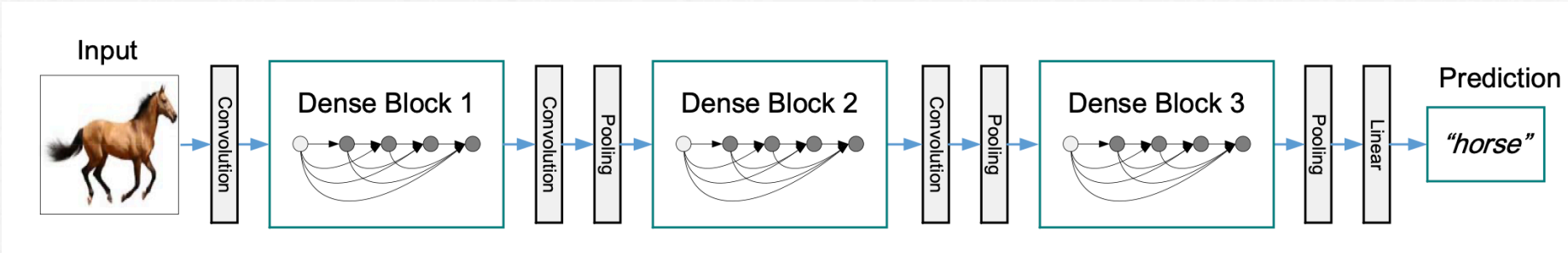


Figure 2. Residual learning: a building block.

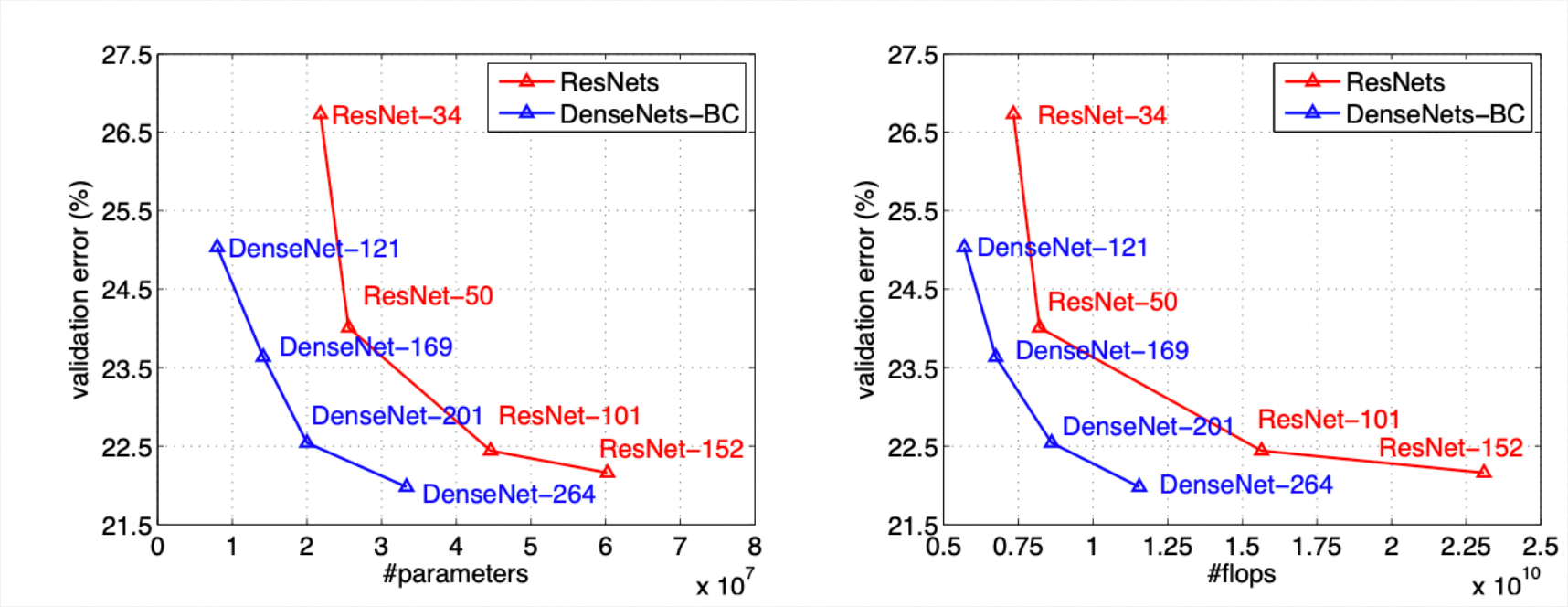
DenseNet (2017)



- Proposed by researchers from Cornell & Tsinghua universities and Facebook.
- For each layer, the feature-maps (i.e., the output of a convolution) of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.
- DenseNets have several advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

from https://arxiv.org/pdf/1608.06993.pdf

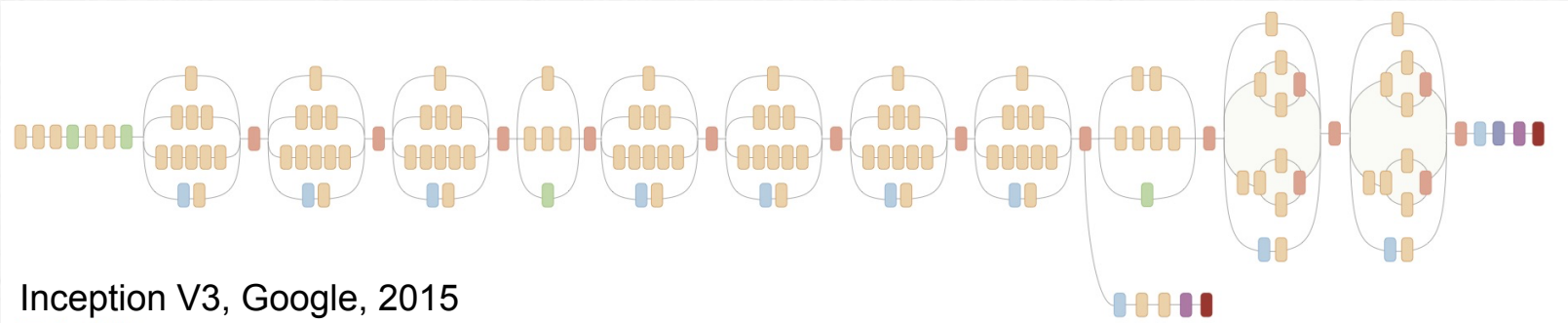
DenseNets vs ResNets



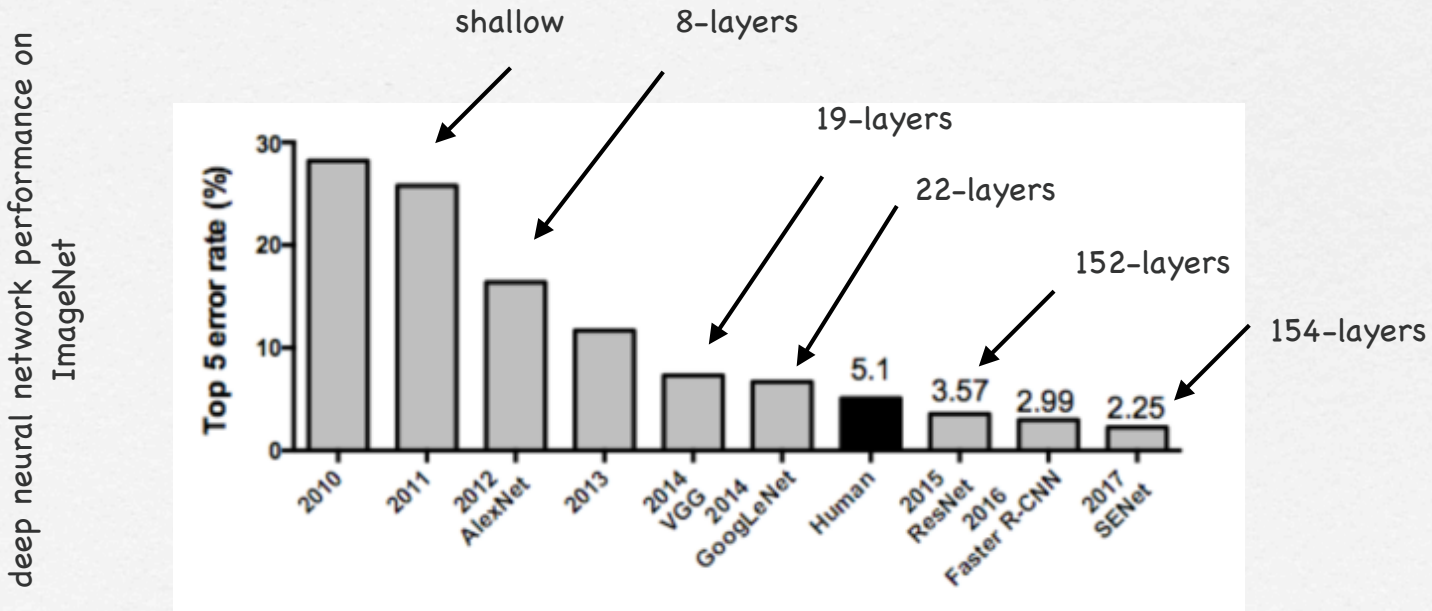
- DenseNets reach similar performance than ResNets with many less parameters (left) and require less computation (right).

from https://arxiv.org/pdf/1608.06993.pdf

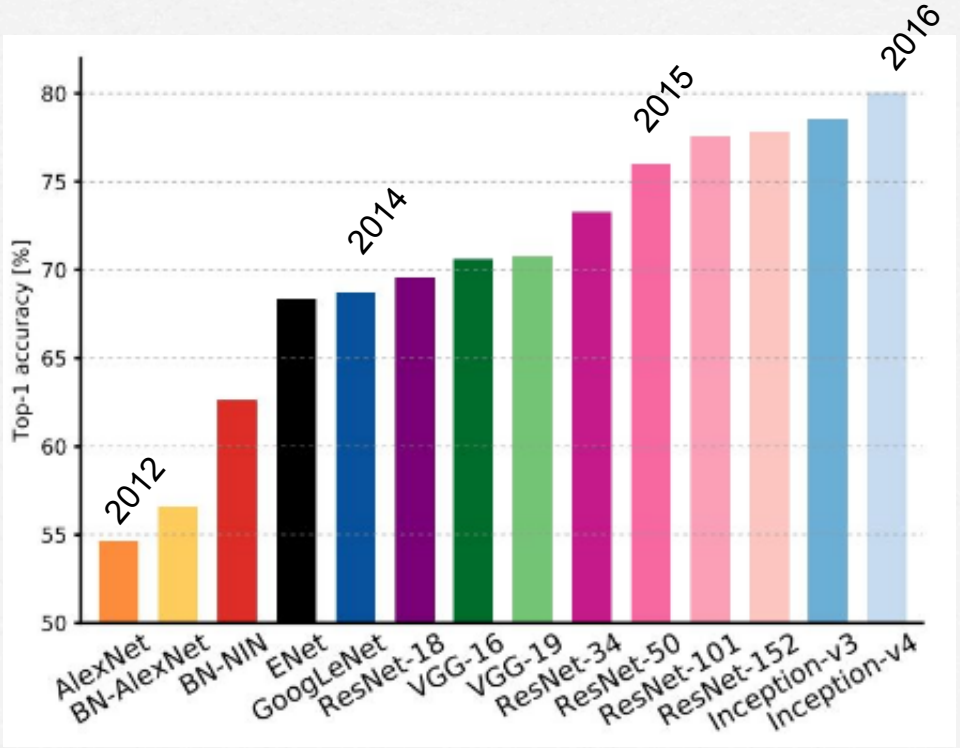
The alchemy of Machine Learning



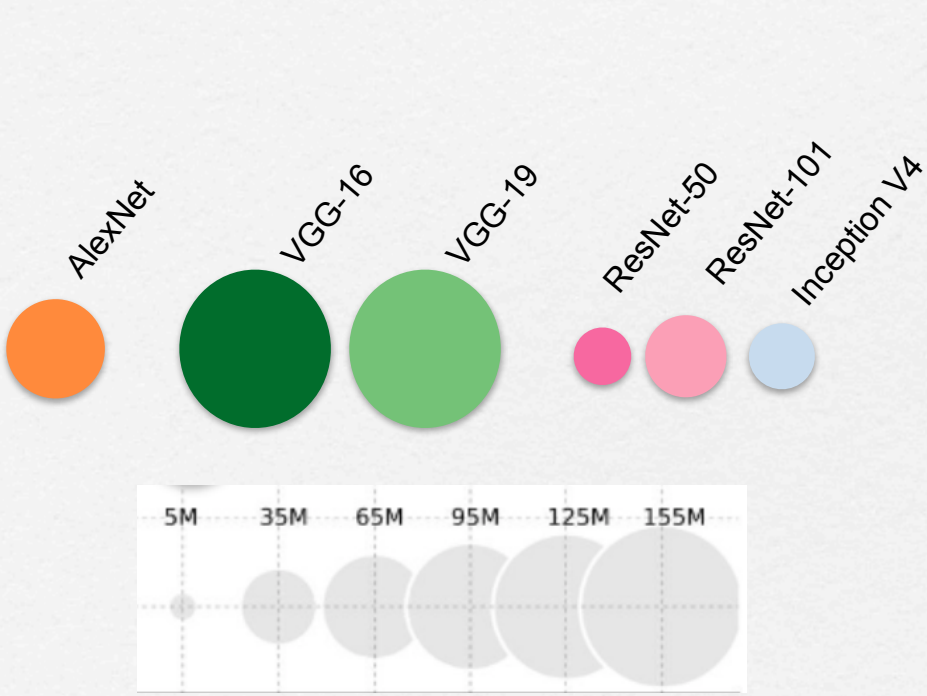
Inception V3, Google, 2015



Evolution of CNN architectures



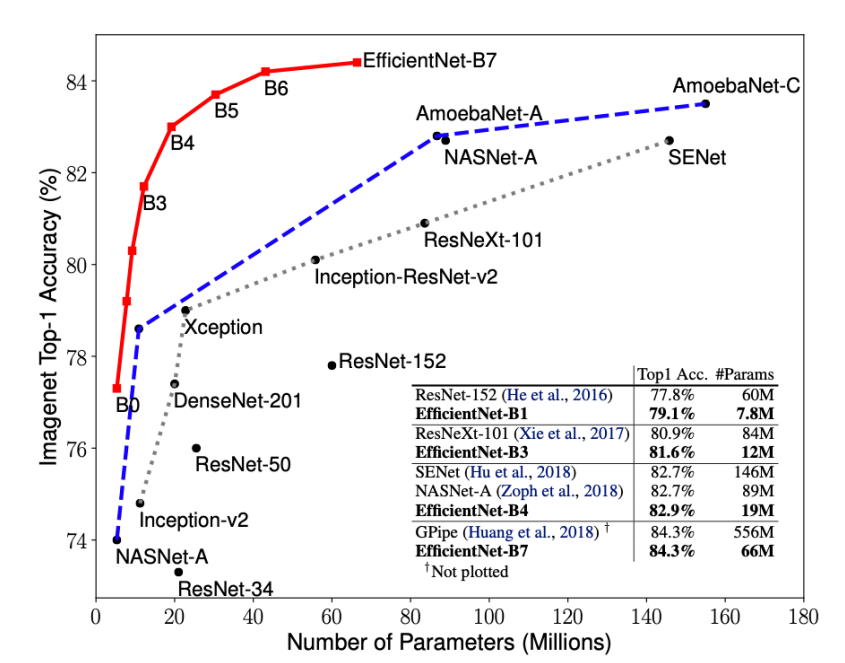
Canziani et al., 2017



Number of parameters

EfficientNets (2020)

- Mingxing Tan and Quoc Le from the Brain Team of Google Research systematically studied model scaling and identified that carefully balancing network depth, width, and resolution can lead to better performances.



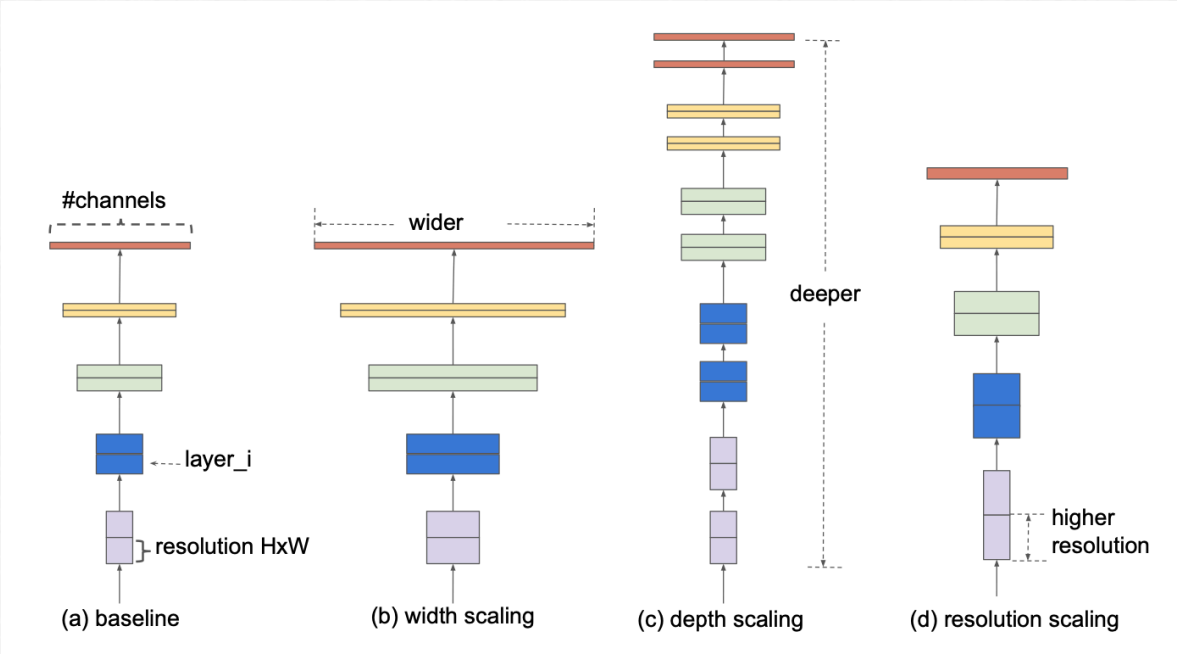
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

from https://arxiv.org/pdf/1905.11946v5.pdf

EfficientNets scaling (1)

- Example: assume ResNet-18 as a reference architecture processing images of 224x224 resolution: d=1.0(depth) r=1.0(resolution)
- A new architecture with d=2.0 and r=1.3 will be a ResNet-36 processing 299x299 images. The width factor modifies the number of filters per layer.



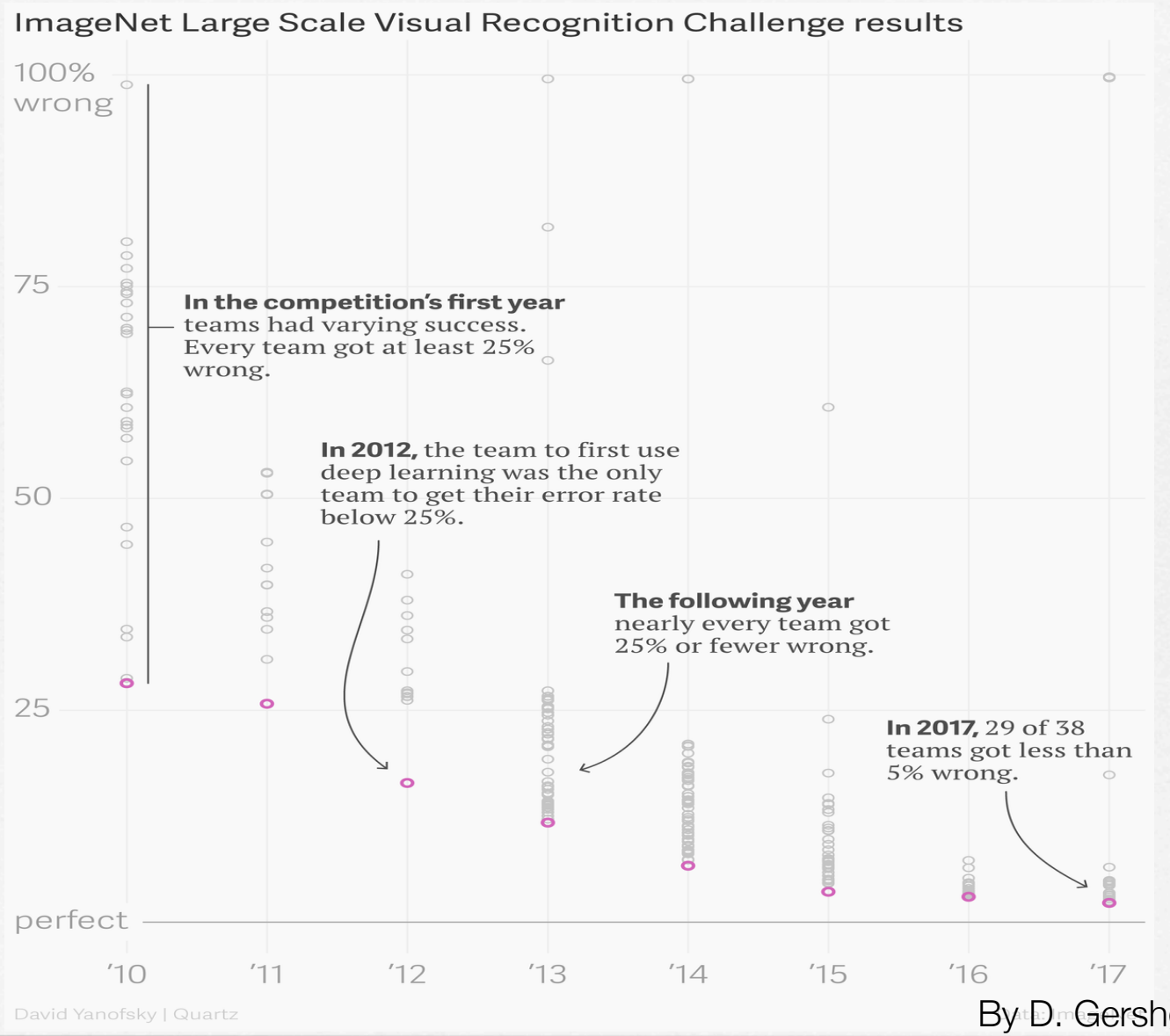
from https://arxiv.org/pdf/1905.11946v5.pdf

EfficientNets scaling (2)

- Tan et Le found that they could define a compound scaling factor Φ: d= α^{Φ}, w = β^{Φ}, r = γ^{Φ} , where $\alpha,\beta,\gamma \geq 1$
- Given that the computational load (FLOPS) of a deep network is proportional to d, w^2 and r^2. Thus, using the constraint $\alpha.\beta^2.\gamma^2 \approx 2$, the FLOPS get proportional to 2^{Φ}.
- EfficientNet-B0 performs better than ResNet-50 (~5x less parameters; 11x less FLOPS) and DenseNet-169 (~2.5x less parameters; 9x less FLOPS); EfficientNet-B1 performs better than ResNet-152 (~7.5x less parameters; 16x less FLOPS), etc.

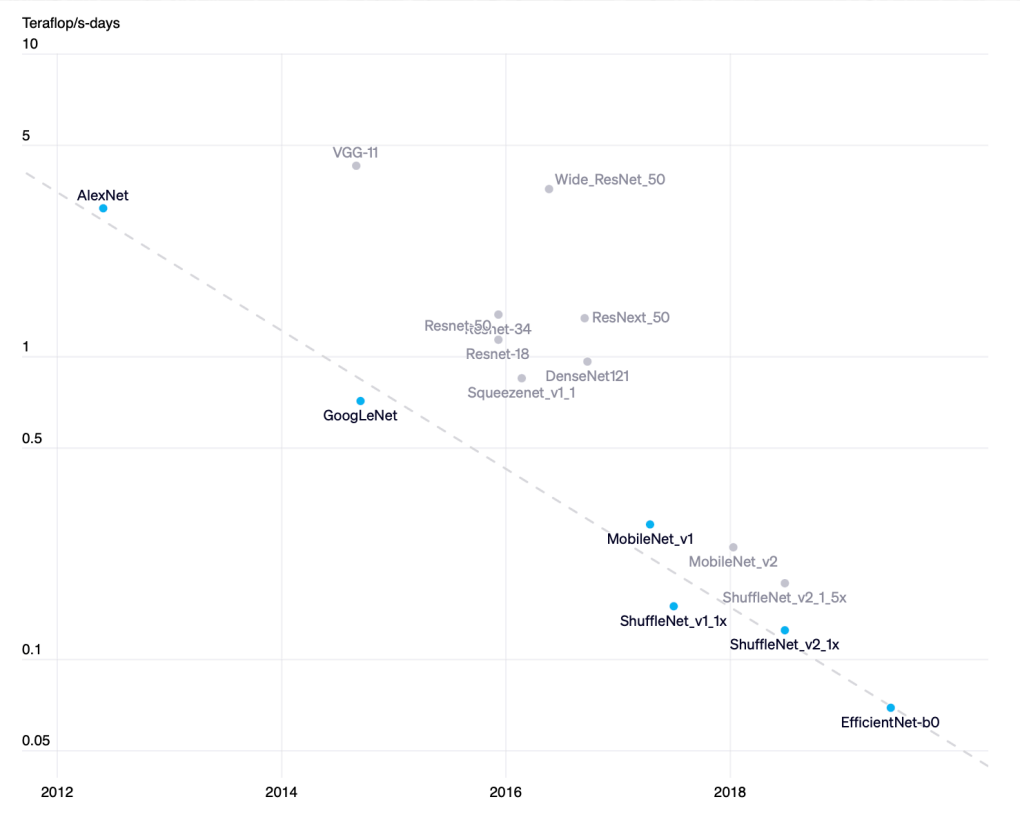
from https://arxiv.org/pdf/1905.11946v5.pdf

« Democratization » of Deep Learning



By D. Gershgorn, Quartz. 2017

Training efficiency improvement



- Besides the improvement of hardware (Moore's law), the amount of compute needed to train a neural net has decreased by 2 every 16 months (thanks to better hyperparameter search, architectures and training procedures).

https://openai.com/research/ai-and-efficiency