An Introduction to Convolutional Neural Networks

Yuan YAO
HKUST

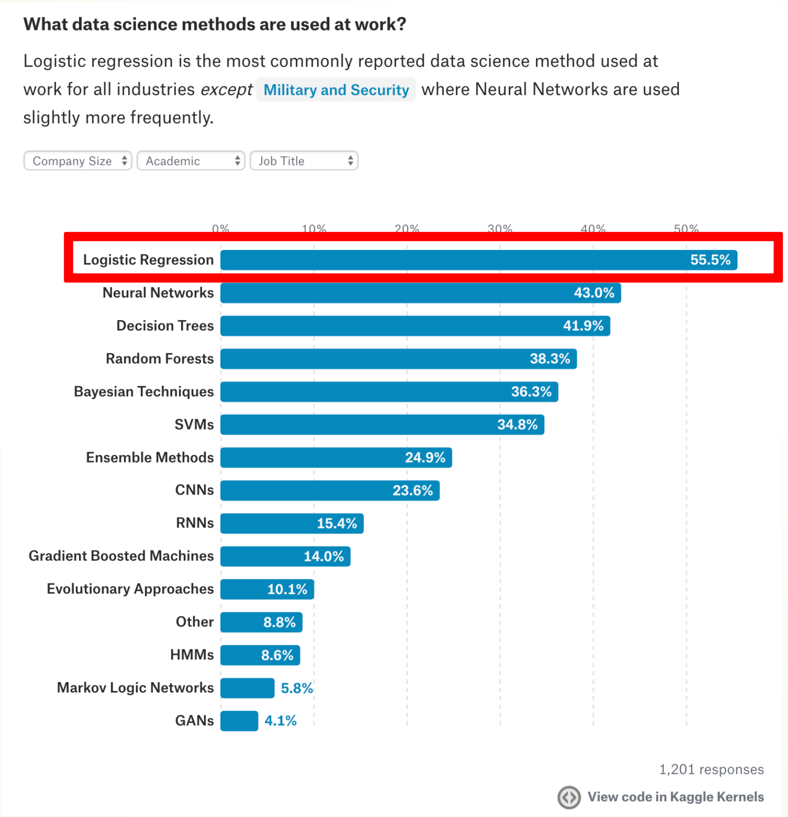
Summary

- We had covered so far
  - Linear models (linear and logistic regression) – always a good start, simple yet powerful
  - Model Assessment and Selection – basics for all methods
  - Trees, Random Forests, and Boosting – good for high dim mixed-type heterogeneous features
  - Support Vector Machines – good for small amount of data but high dim geometric features
- Next, neural networks for unstructured data (image, language etc.):
  - **Convolutional Neural Networks** – image data
  - Recurrent Neural Networks, LSTM – sequence data
  - Transformer, BERT, GPT – machine translation, NLP, etc.
  - Generative models, GANs, Diffusion – new unsupervised learning for image, etc.
  - Reinforcement Learning – Markov decision process, playing games, etc.

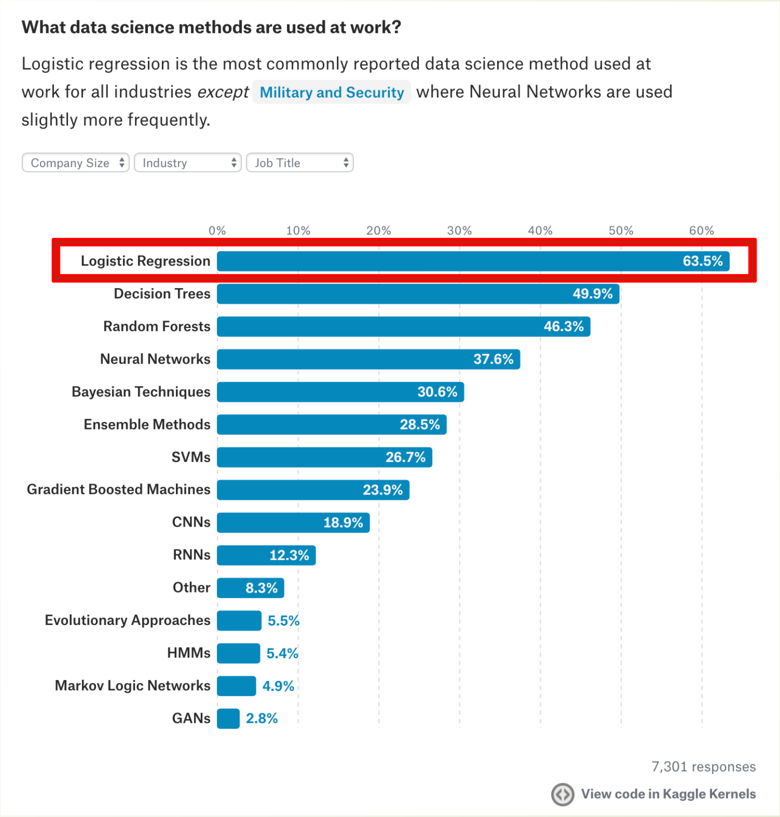
Kaggle survey: Top ML Methods

https://www.kaggle.com/surveys/2017

Academic



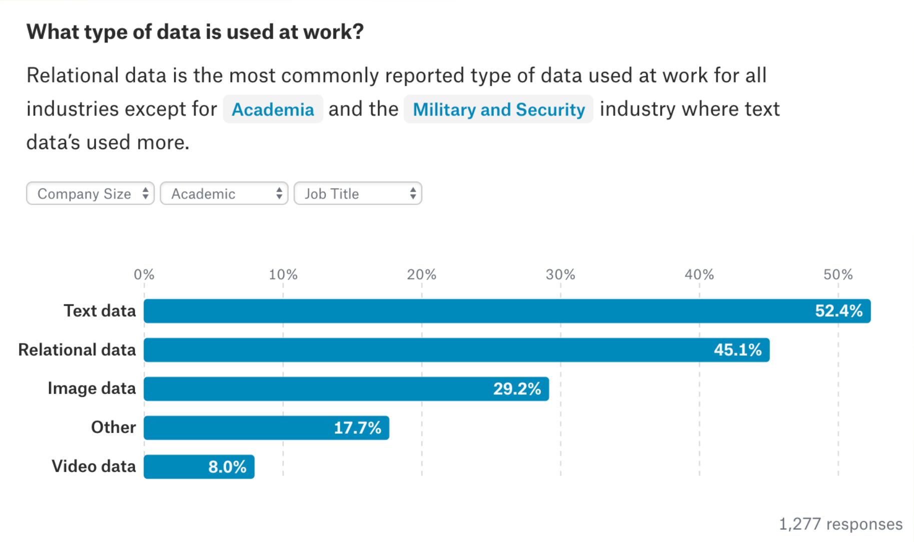
Industry



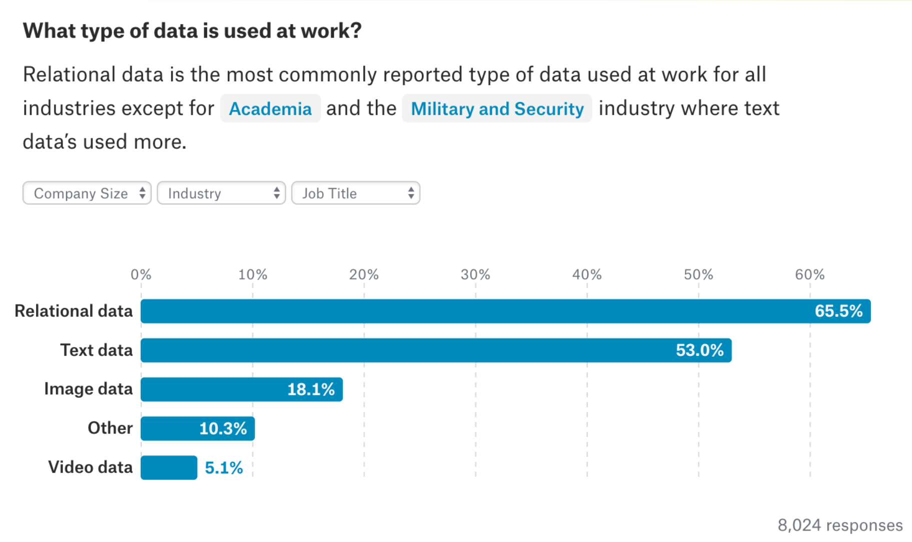
What type of data is used at work?

https://www.kaggle.com/surveys/2017

Academic



Industry



Some reference books on Deep Learning

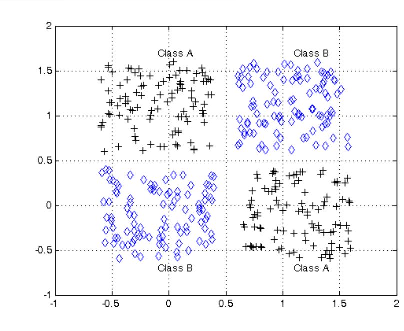
- Deep Learning with Python, Manning Publications 2017
  - by François Chollet
  - https://www.manning.com/books/deep-learning-with-python?a_aid=keras&a_bid=76564dff
- Deep Learning, MIT Press 2016
  - By Ian Goodfellow, Yoshua Bengio, and Aaron Courville,
  - http://www.deeplearningbook.org/
- Many other public resources

Locality or Sparsity of Computation

Minsky and Papert, 1969

Perceptron can't do **XOR** classification

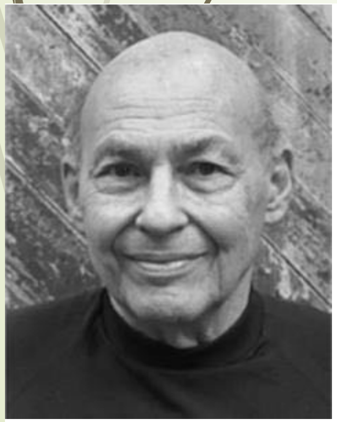
Perceptron needs infinite global information to compute **connectivity**



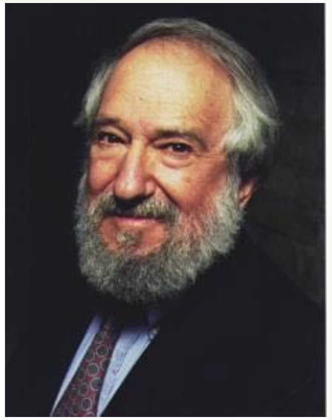
Locality or **Sparsity** is important:

Locality in time?

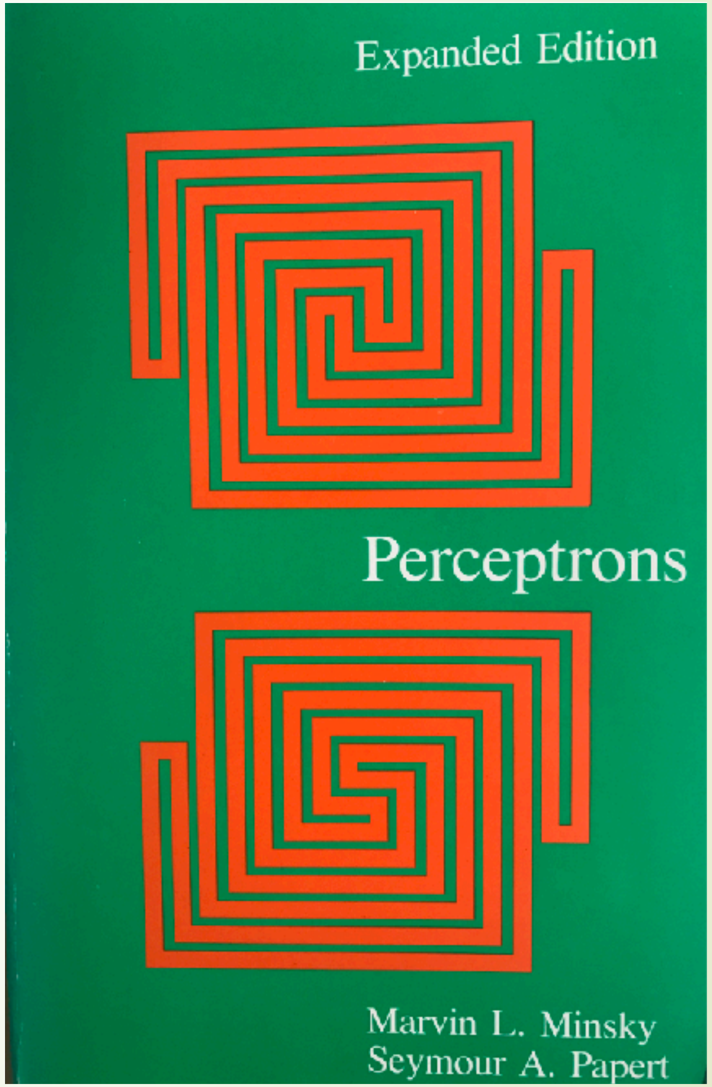
Locality in space?



Marvin Minsky



Seymour Papert



Convolutional Neural Networks: shift invariances and locality for images

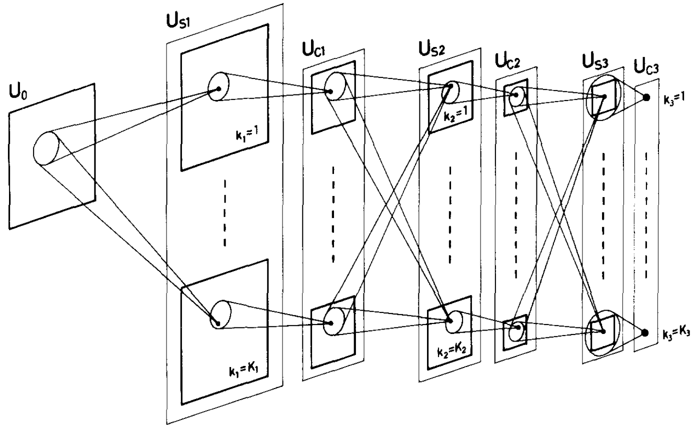
- Can be traced to *Neocognitron* of Kunihiko Fukushima (1979)
- Yann LeCun combined convolutional neural networks with back propagation (1989)
- Imposes **shift invariance** and **locality** on the weights
- Forward pass remains similar
- Backpropagation slightly changes – need to sum over the gradients from all spatial positions

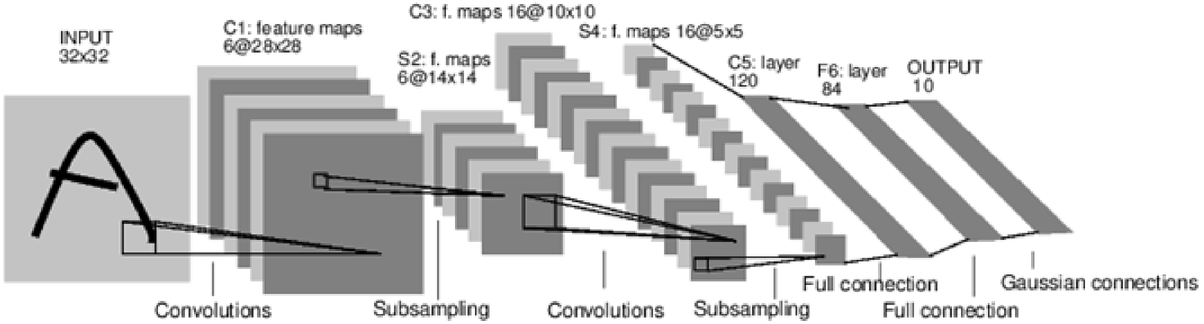
Biol. Cybernetics 36, 193–202 (1980)

Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position

Kunihiko Fukushima

NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan



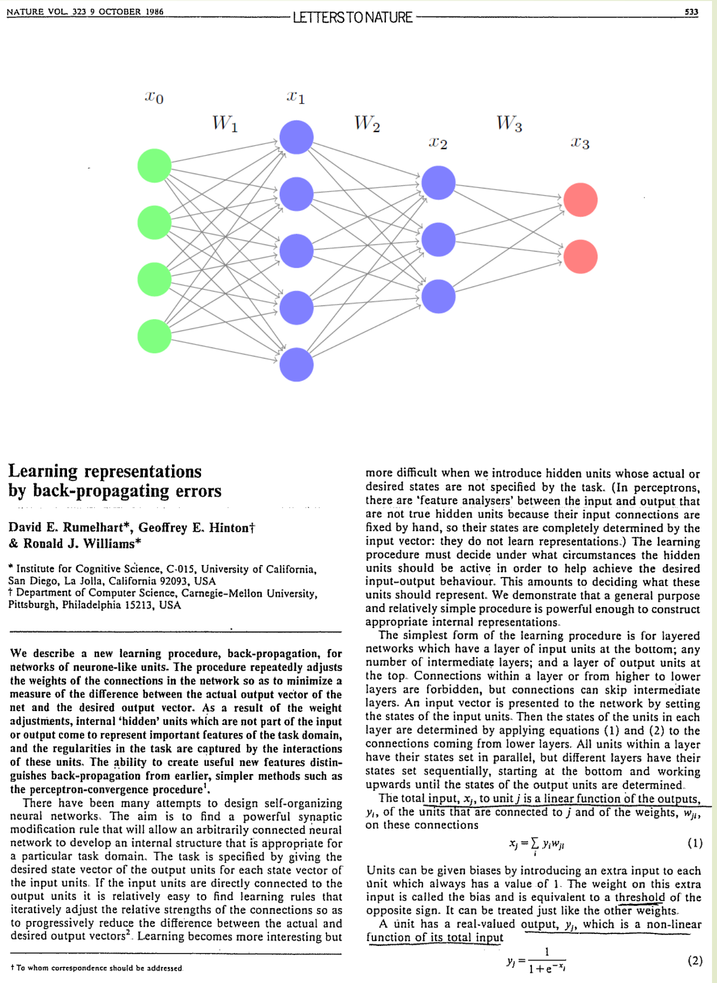


Multilayer Perceptrons (MLP) and Back-Propagation (BP) Algorithms

Rumelhart, Hinton, Williams (1986)
Learning representations by back-propagating errors, Nature, 323(9): 533-536

BP algorithms as **stochastic gradient descent** algorithms (**Robbins–Monro 1950; Kiefer-Wolfowitz 1951**) with Chain rules of Gradient maps

MLP classifies **XOR**, but the global hurdle on topology (connectivity) computation still exists



NATURE VOL. 323 9 OCTOBER 1986 LETTERS TO NATURE 533

Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are 'feature analysers' between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \tag{1}$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

$$y_j = \frac{1}{1+e^{-x_j}} \tag{2}$$

† To whom correspondence should be addressed.

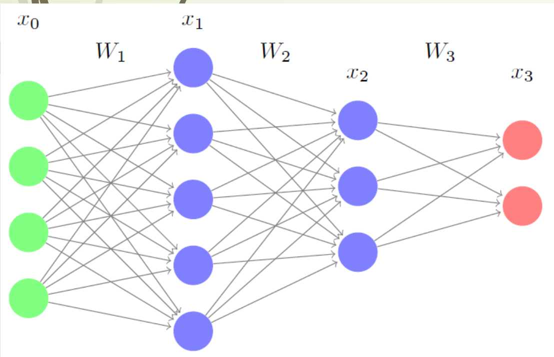
BP Algorithm: Forward Pass

- Cascade of repeated [linear operation followed by coordinatewise nonlinearity]'s
- Nonlinearities: sigmoid, hyperbolic tangent, (recently) ReLU.

Algorithm 1 Forward pass

Input: x_0
Output: x_L

1: **for** $\ell = 1$ to L **do**
2: $\quad x_\ell = f_\ell(W_\ell x_{\ell-1} + b_\ell)$
3: **end for**



BP algorithm = Gradient Descent Method

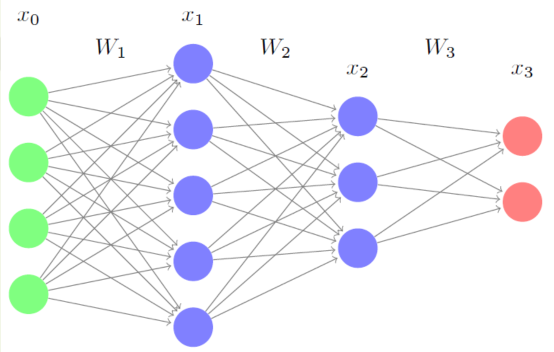
- Training examples $\{x_0^i\}_{i=1}^n$ and labels $\{y^i\}_{i=1}^n$
- Output of the network $\{x_L^i\}_{i=1}^m$
- Objective

$$J(\{W_l\},\{b_l\}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}\|y^i - x_L^i\|_2^2 \tag{1}$$

Other losses include cross-entropy, logistic loss, exponential loss, etc.

- Gradient descent

$$W_l = W_l - \eta\frac{\partial J}{\partial W_l}$$
$$b_l = b_l - \eta\frac{\partial J}{\partial b_l}$$



In practice: use Stochastic Gradient Descent (SGD)

Derivation of BP: Lagrangian Multiplier

LeCun et al. 1988

Given n training examples $(I_i, y_i) \equiv$ (input,target) and L layers

- Constrained optimization

$$\min_{W,x} \quad \sum_{i=1}^{n} \|x_i(L) - y_i\|_2$$

subject to $x_i(\ell) = f_\ell\Big[W_\ell x_i\,(\ell - 1)\Big],$

$$i = 1, \ldots, n, \quad \ell = 1, \ldots, L, \; x_i(0) = I_i$$

- Lagrangian formulation (Unconstrained)

$$\min_{W,x,B} \mathcal{L}(W, x, B)$$

$$\mathcal{L}(W, x, B) = \sum_{i=1}^{n} \Big\{ \|x_i(L) - y_i\|_2^2 + \sum_{\ell=1}^{L} B_i(\ell)^T \Big(x_i(\ell) - f_\ell\Big[W_\ell x_i\,(\ell - 1)\Big]\Big)\Big\}$$

http://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf

back-propagation – derivation

- $\frac{\partial \mathcal{L}}{\partial B}$

Forward pass

$$x_i(\ell) = f_\ell\left[\underbrace{W_\ell x_i\,(\ell-1)}_{A_i(\ell)}\right] \quad \ell = 1,\ldots,L, \quad i = 1,\ldots,n$$

- $\frac{\partial \mathcal{L}}{\partial x}$, $z_\ell = [\nabla f_\ell]B(\ell)$

Backward (adjoint) pass

$$z(L) = 2\nabla f_L\left[A_i(L)\right](y_i - x_i(L))$$

$$z_i(\ell) = \nabla f_\ell\left[A_i(\ell)\right]W_{\ell+1}^T z_i(\ell+1) \quad \ell = 0,\ldots,L-1$$

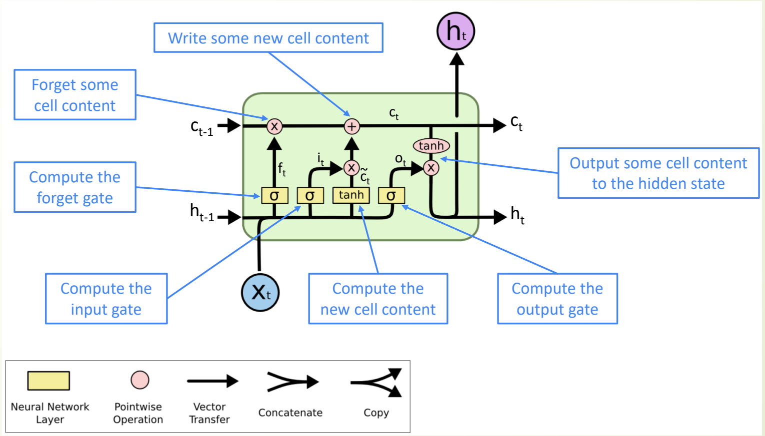
- $W \leftarrow W + \lambda \frac{\partial \mathcal{L}}{\partial W}$

Weight update

$$W_\ell \leftarrow W_\ell + \lambda \sum_{i=1}^n z_i(\ell)x_i^T(\ell-1)$$

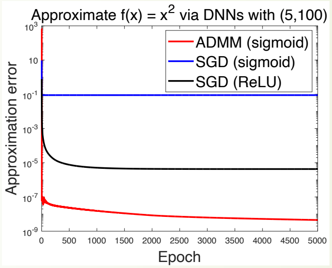
Long-Short-Term-Memory (LSTM, 1997)

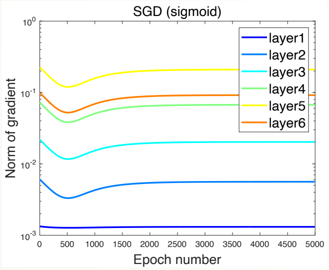
- Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". *Neural Computation*. **9** (8): 1735–1780. (https://www.bioinf.jku.at/publications/older/2604.pdf)
- BP can not train deep networks due to gradient vanishing problem etc.
- Introduction of **short path** to train deep networks without vanishing gradient problem.
- This idea will come back to Convolutional Networks as **ResNet** in 2015.



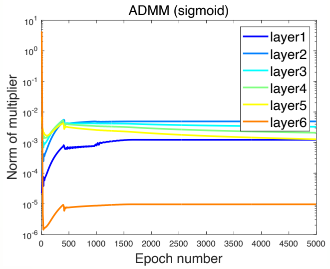
SGD vs. ADMM/BCD

- Stochastic Gradient Descent (SGD) suffers from the well-known *gradient vanishing issue* in deep learning
- ADMM/BCD may alleviate gradient vanishing

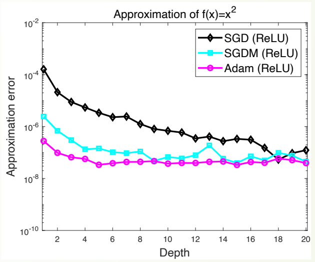




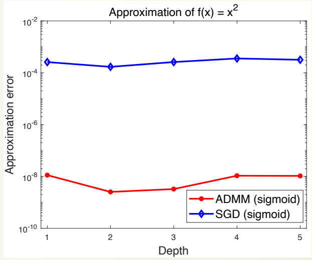
(a) Gradient vanishing of SGD



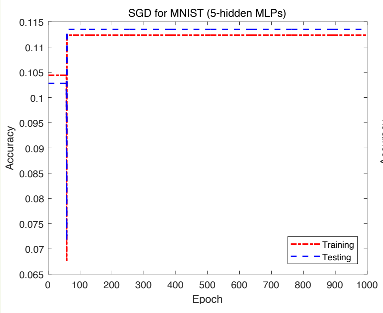
(b) Saturation-avoidance of ADMM

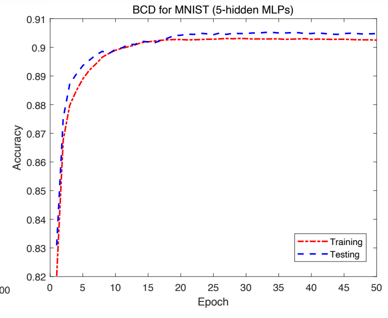


(a) Deep ReLU nets

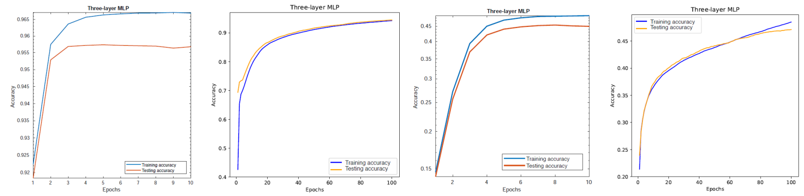


(b) Deep sigmoid nets





- High epoch efficiency of BCD at early stage



(a) MNIST (BCD) (b) MNIST (SGD) (c) CIFAR-10 (BCD) (d) CIFAR-10 (SGD)

Zeng-Lau-Lin-Y., ICML 2019
Zeng-Lin-Y.-Zhou, JMLR 2021

Notes on Algorithms

- Gradient descent (back propagation) can be derived via Lagrangian multiplier method [LeCun 1988, http://yann.lecun.com/exdb/publis/pdf/lecun-88.pdf]
- **ADMM** (Alternating Direction Method of Multipliers) is alternative primal-dual method via **Augmented Lagrangian** multipliers [Zeng-Lin-Y.-Zhou, JMLR 2021]
- **BCD** (Block-Coordinate-Descent) ***drops*** the dual update in Augmented Lagrangian multipliers [Zeng-Lau-Lin-Y., ICML 2019]
- *Global convergence to KKT points from arbitrary initialization* can be established with the aid of ***Kurdyka-Łojasiewicz*** framework.

$$\begin{aligned}
&\underset{\mathcal{W},\mathcal{V}}{\text{minimize}} \ \frac{1}{2}\|V_N - Y\|_F^2 + \frac{\lambda}{2}\sum_{i=1}^{N}\|W_i\|_F^2 \\
&\text{subject to} \quad V_i = \sigma(W_i V_{i-1}),\ i = 1,\ldots,N-1, \quad V_N = W_N V_{N-1},
\end{aligned}$$

Augmented Lagrangian function:

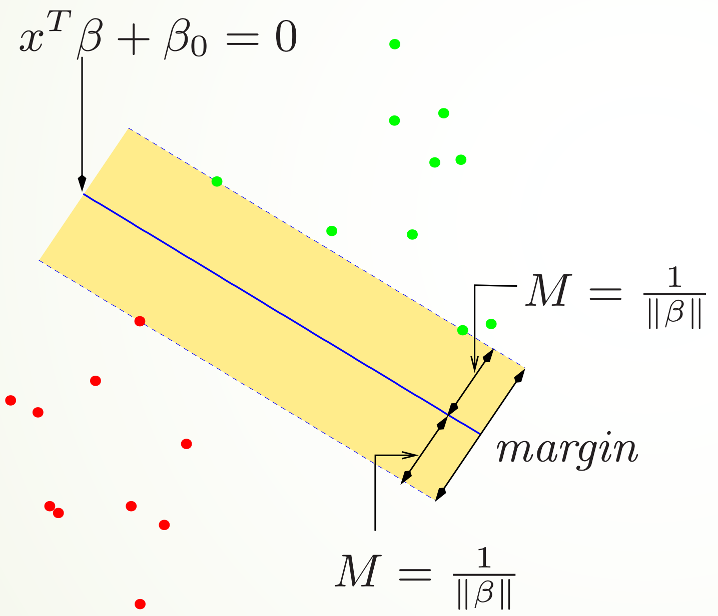
$$\begin{aligned}
\mathcal{L}(\mathcal{W},\mathcal{V},\{\Lambda_i\}_{i=1}^N) := &\frac{1}{2}\|V_N - Y\|_F^2 + \frac{\lambda}{2}\sum_{i=1}^{N}\|W_i\|_F^2 \\
&+ \sum_{i=1}^{N-1}\left(\frac{\beta_i}{2}\|\sigma(W_i V_{i-1}) - V_i\|_F^2 + \langle \Lambda_i, \sigma(W_i V_{i-1}) - V_i\rangle\right) \\
&+ \frac{\beta_N}{2}\|W_N V_{N-1} - V_N\|_F^2 + \langle \Lambda_N, W_N V_{N-1} - V_N\rangle,
\end{aligned}$$

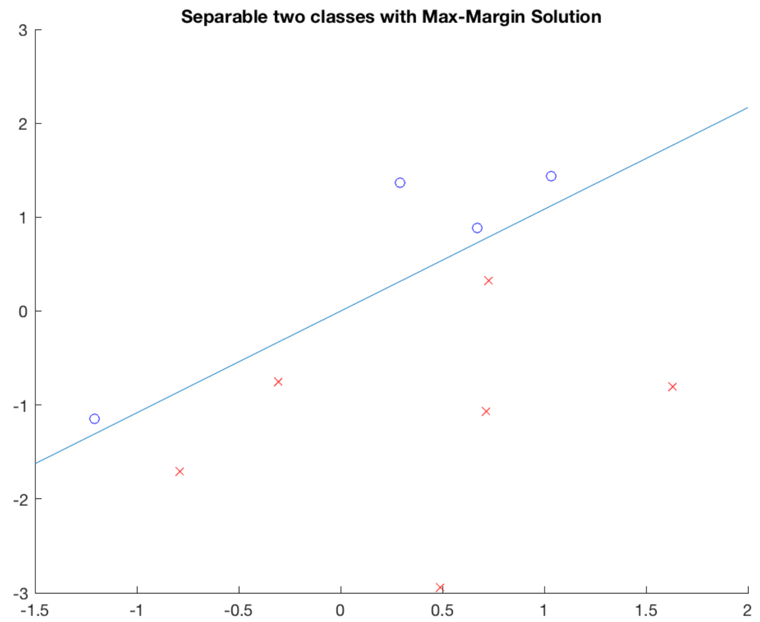
Lagrangian multiplier Λ_i

Support Vector Machine (Max-Margin Classifier)

$$\text{minimize}_{\beta_0,\beta_1,\ldots,\beta_p} \|\beta\|^2 := \sum_j \beta_j^2$$

$$\text{subject to } y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq 1 \text{ for all } i$$





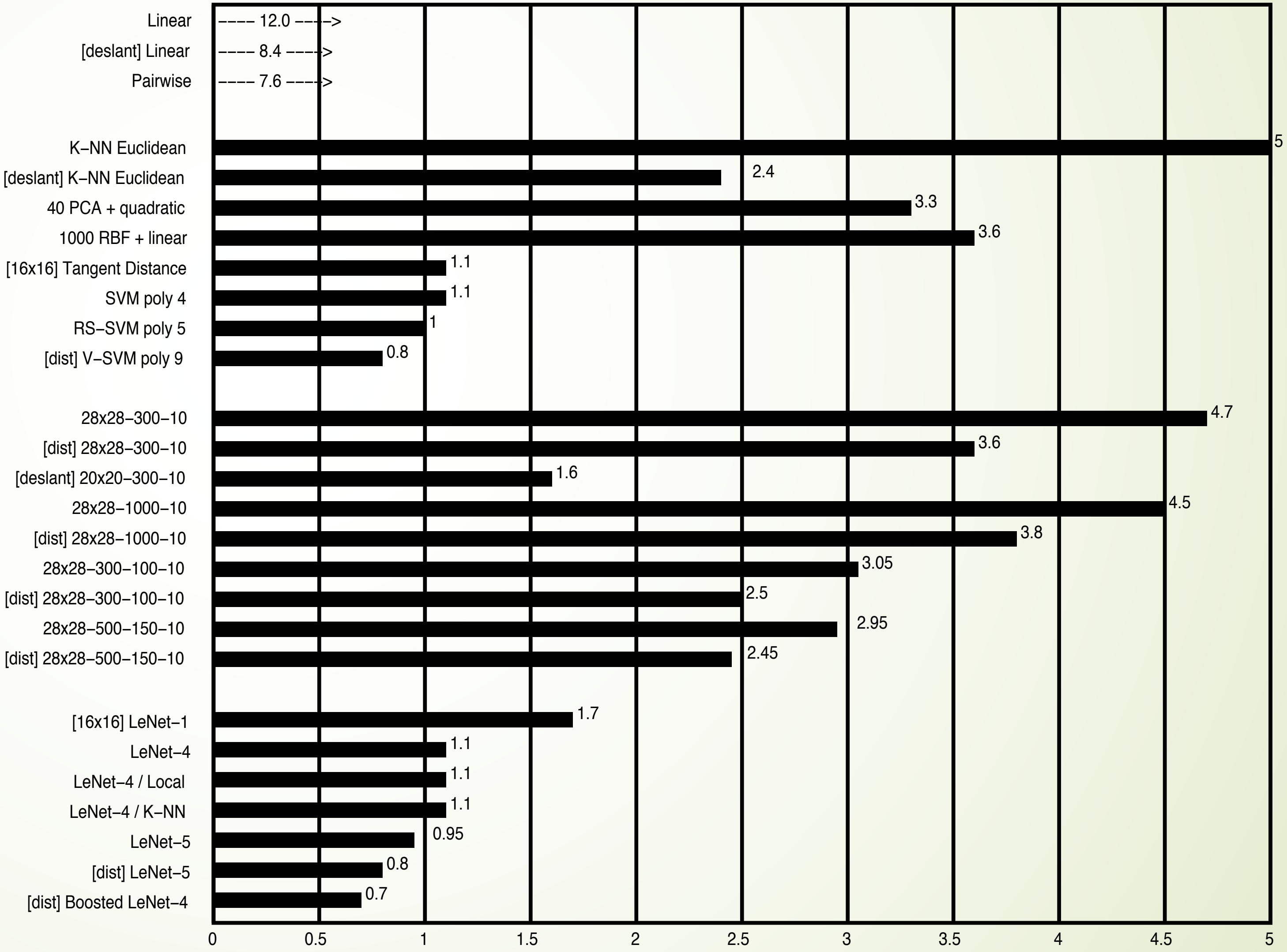
Vladmir Vapnik, 1994

Convex optimization + Reproducing Kernel Hilbert Spaces (Grace Wahba etc.)

MNIST Challenge Test Error: SVM vs. CNN LeCun et al. 1998

Simple SVM performs as well as Multilayer Convolutional Neural Networks which need careful tuning (LeNets)

Second dark era for NN: 2000s



LeNet

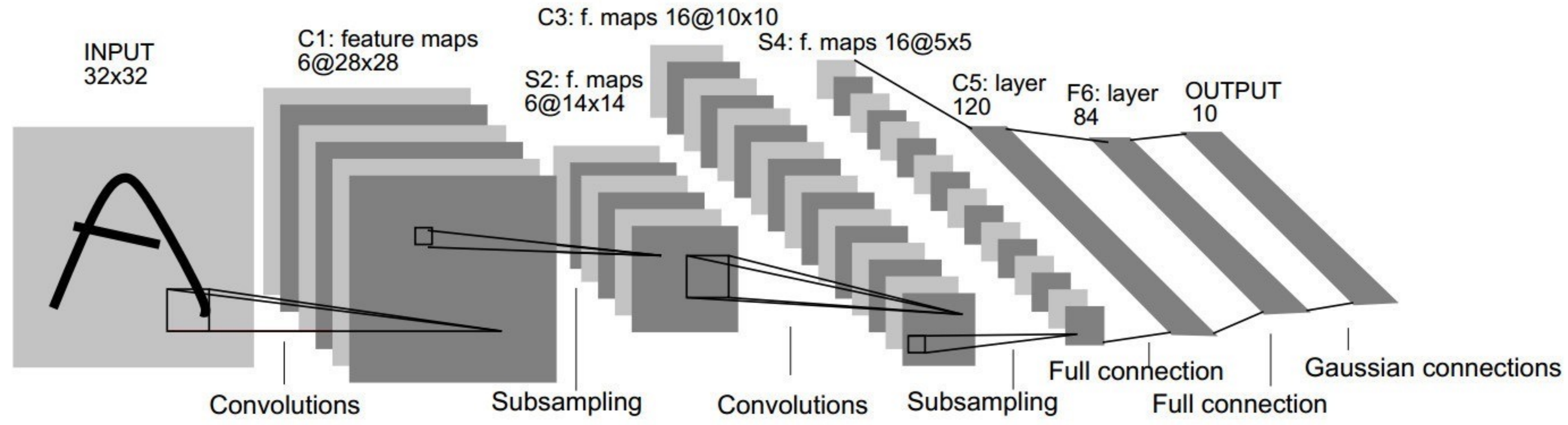
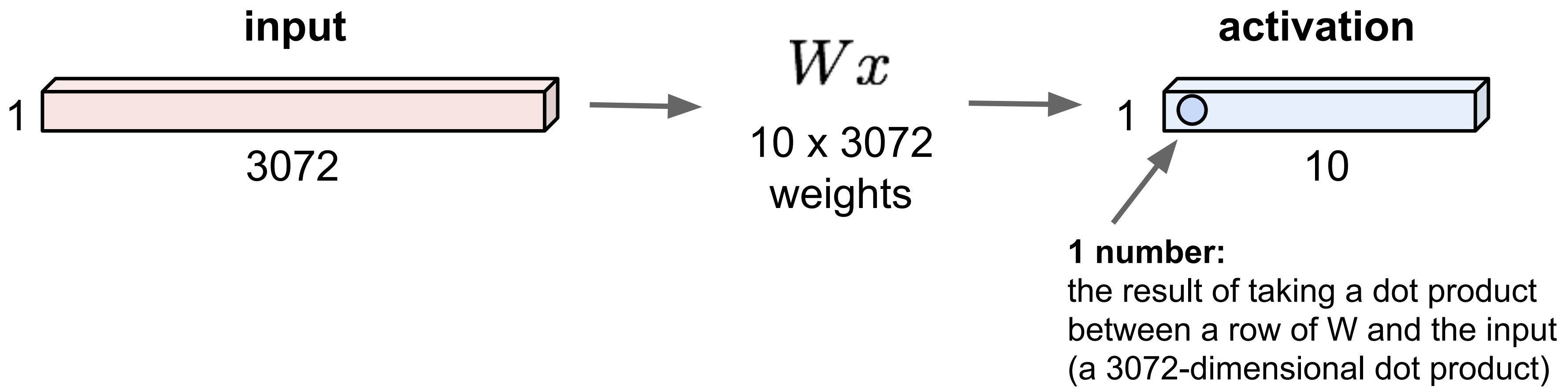


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- **Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner**. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, november 1998.

Fully Connected Layer

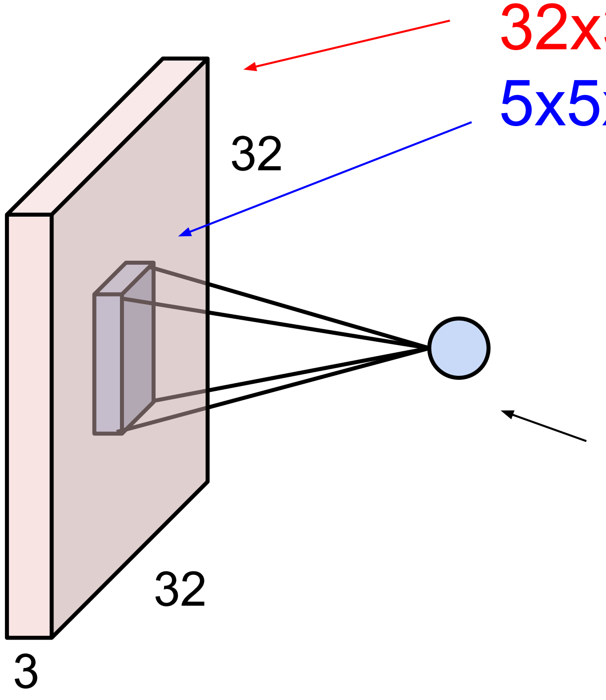
32x32x3 image -> stretch to 3072 x 1



Convolution

32x32x3 image

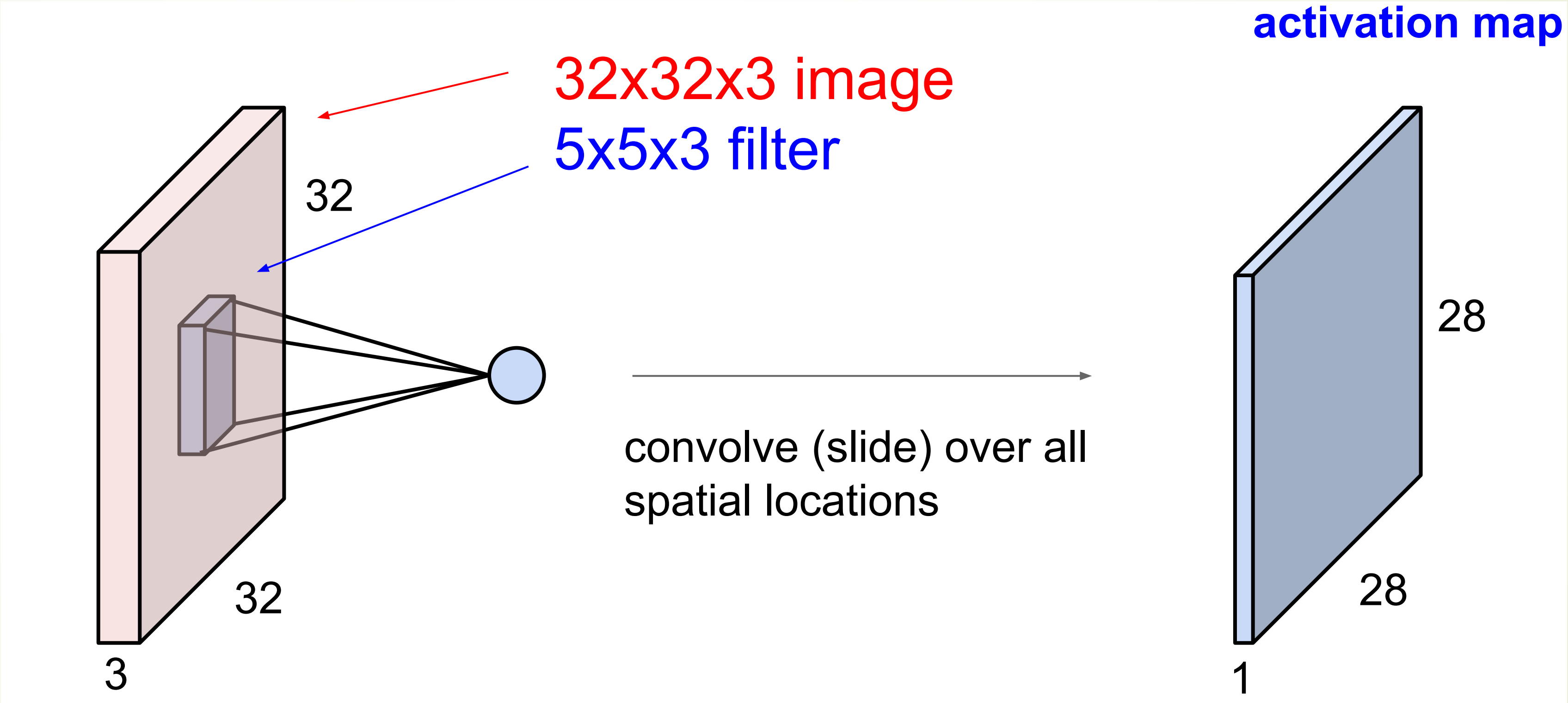
5x5x3 filter w



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

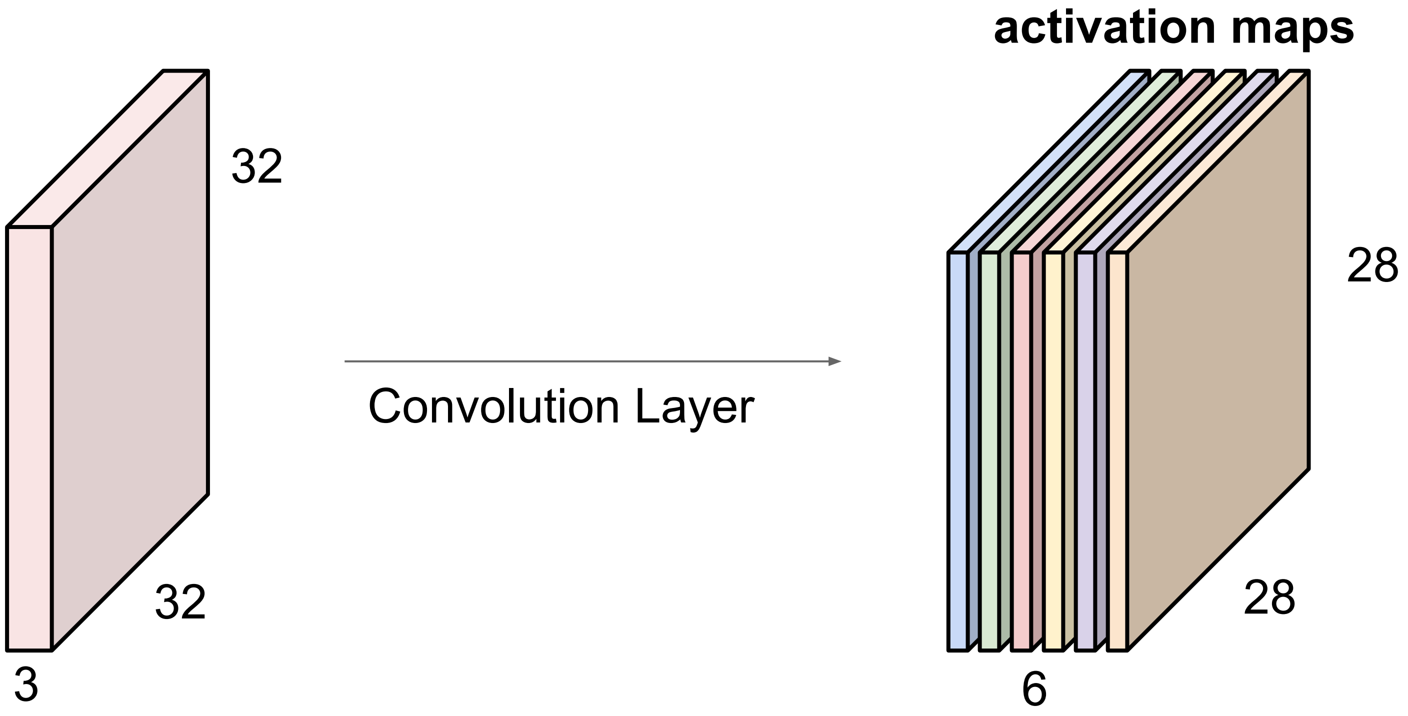
Convolution Layer: a first (blue) filter



Convolution Layer: a second (green) filter

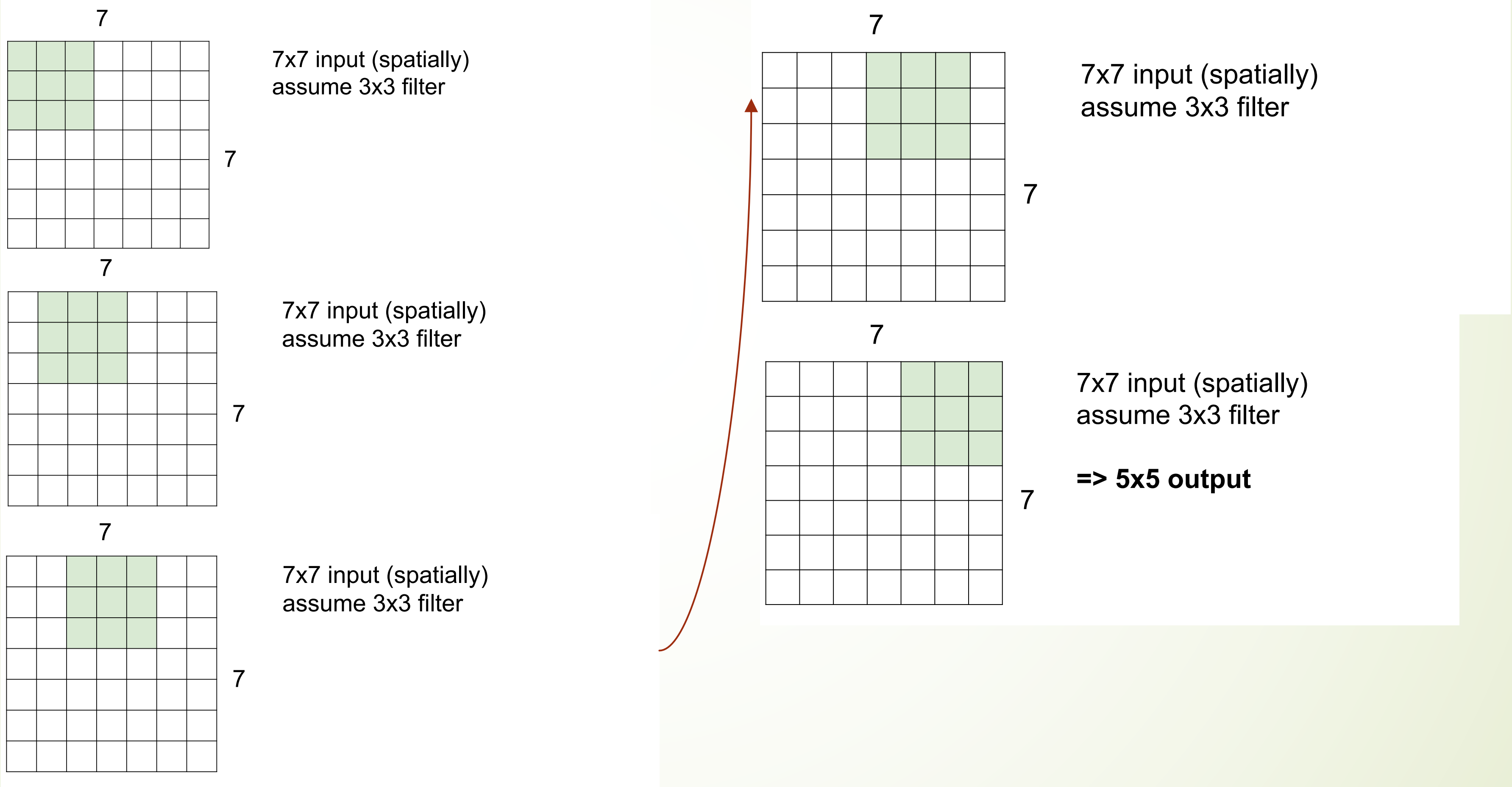
Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

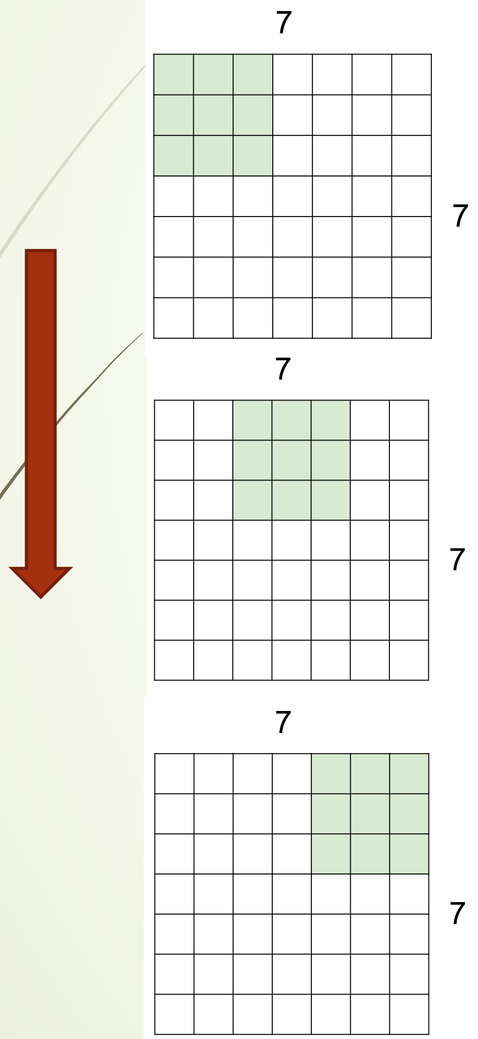


We stack these up to get a "new image" of size 28x28x6!

A Closer Look at Convolution: stride=1



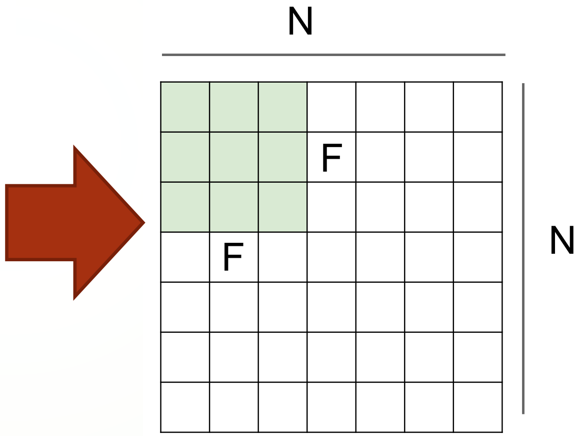
A Closer Look at Convolution: stride=2



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

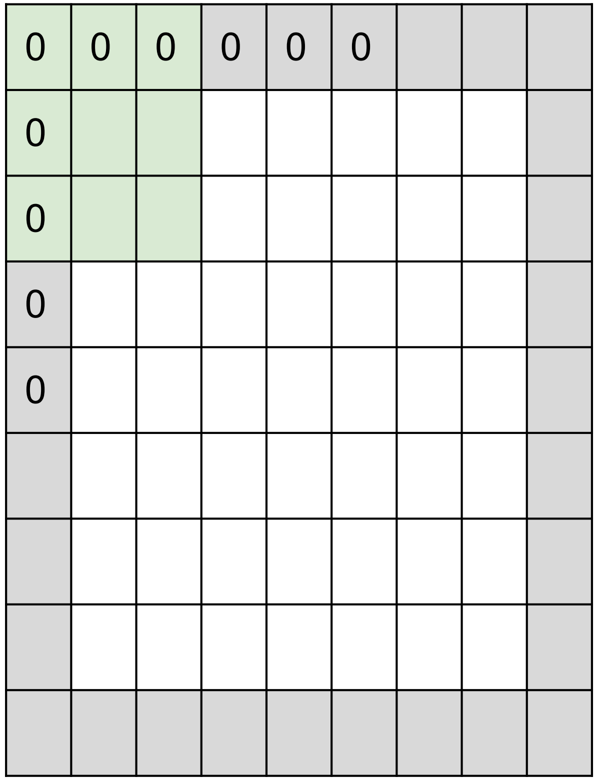
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

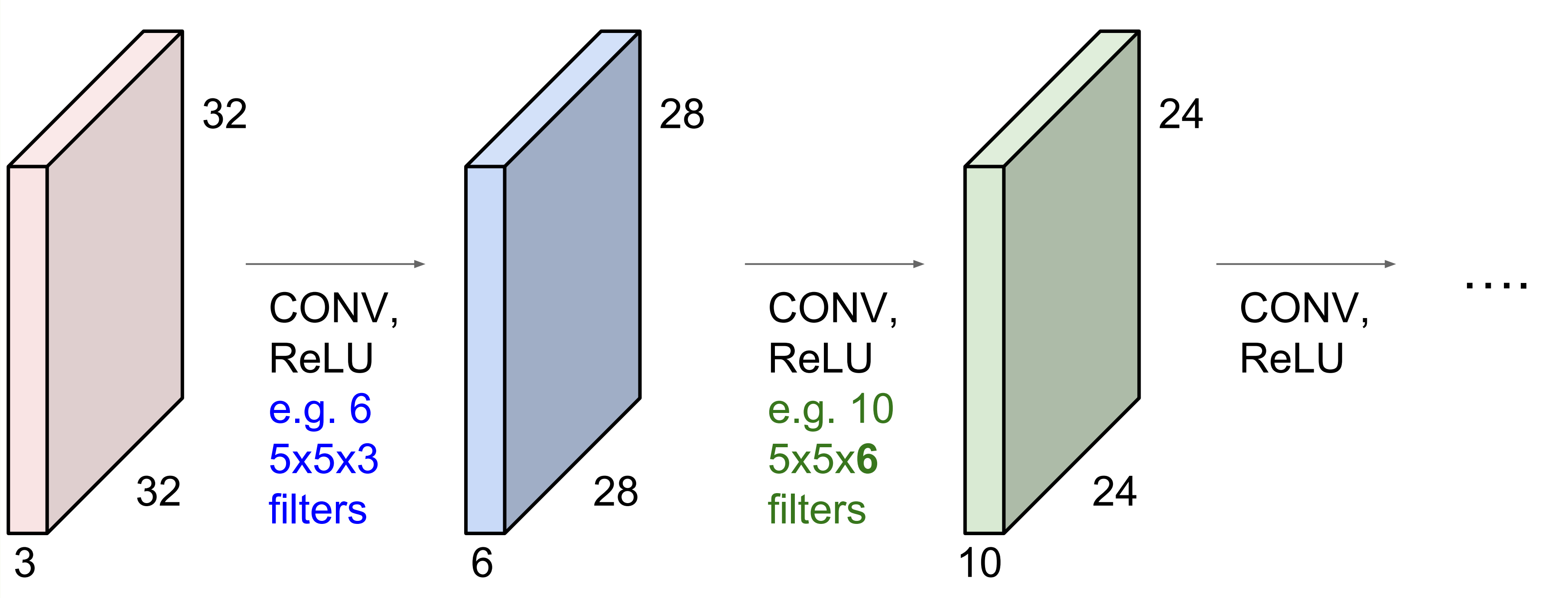
A Closer Look at Convolution: Padding



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

ConvNet:



Stride = 1
Padding = 0

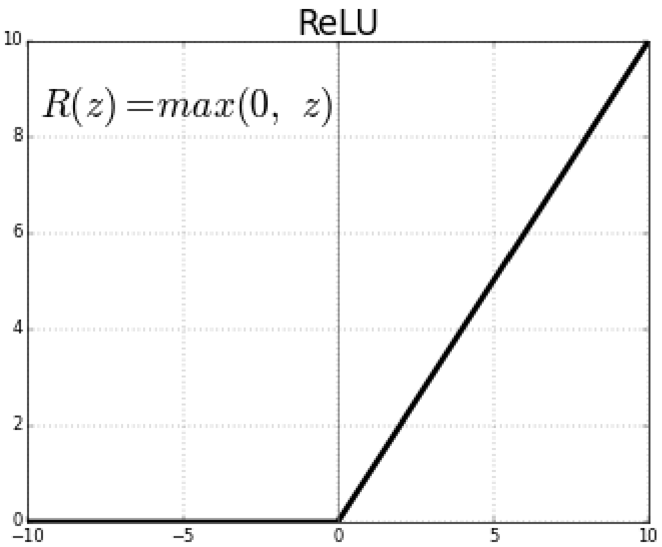
Formula: NewImageSize = floor((ImageSize – Filter + 2*Padding)/Stride + 1)

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

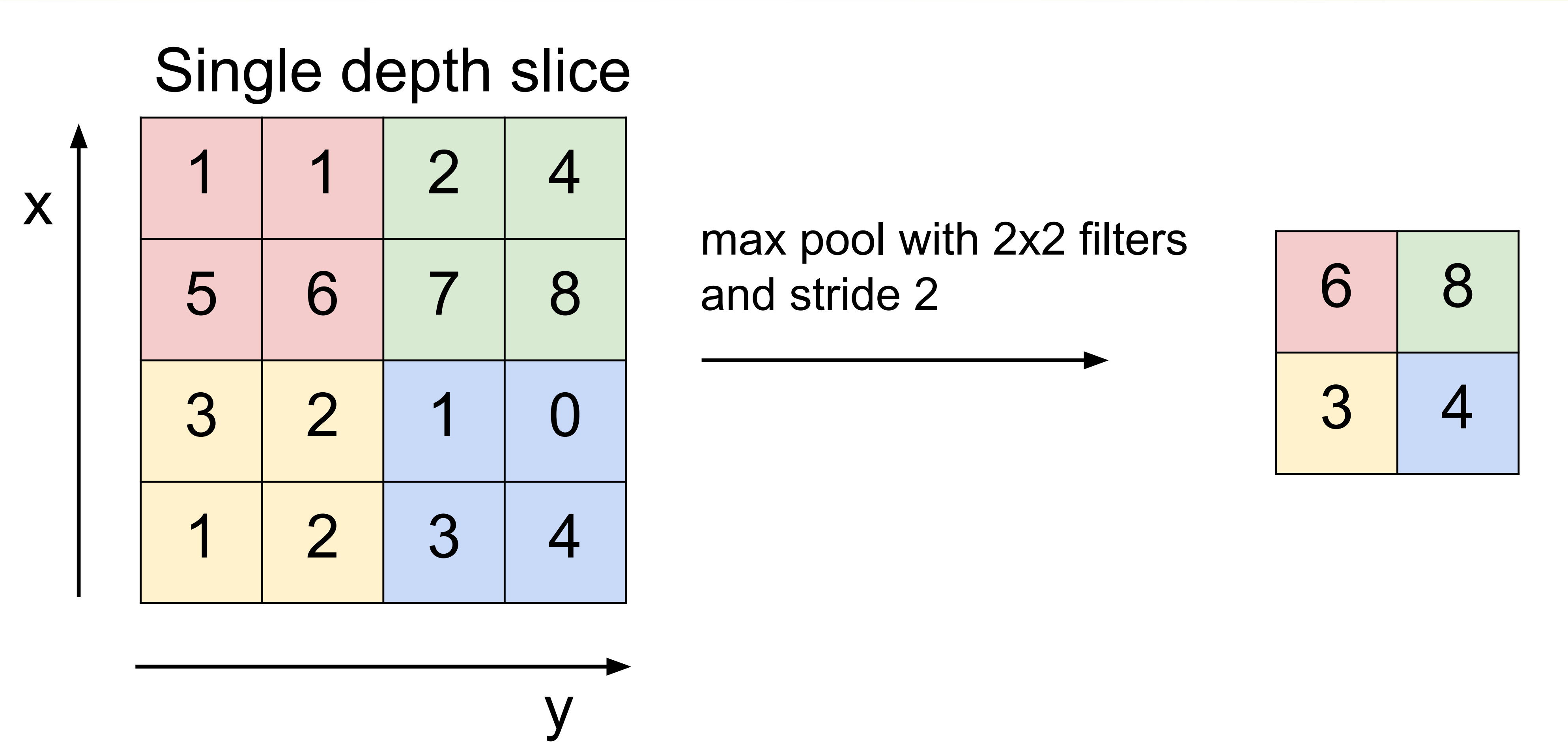
ReLU

- Non-saturating function and therefore faster convergence when compared to other nonlinearities
- Problem of dying neurons



Source: `https://ml4a.github.io/ml4a/neural_networks/`

Max Pooling

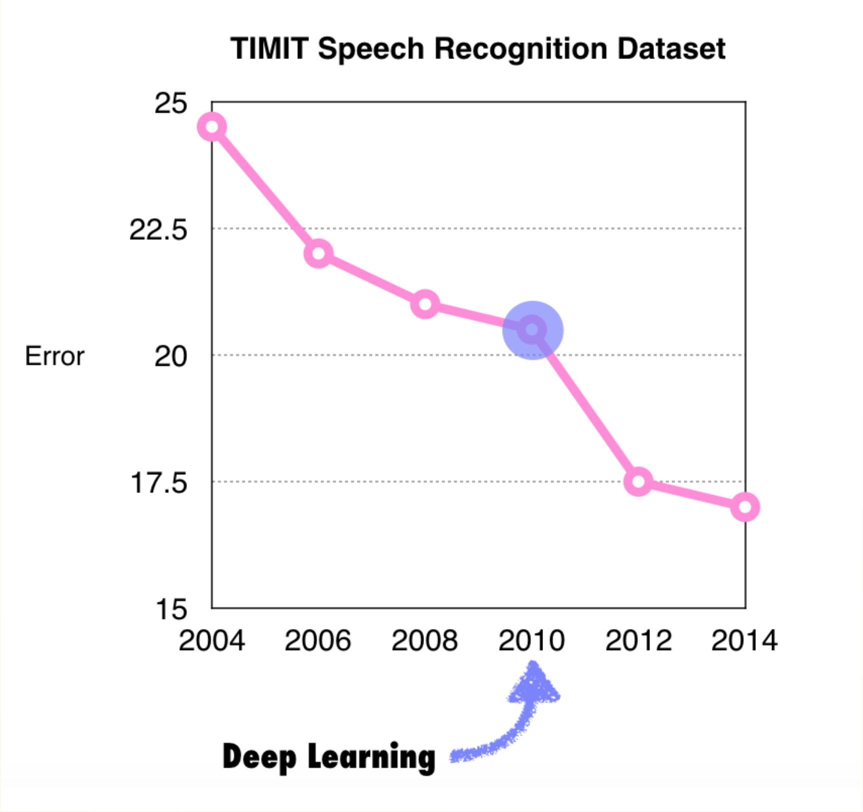


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

2000-2010: The Era of SVM, Boosting, … as nights of Neural Networks

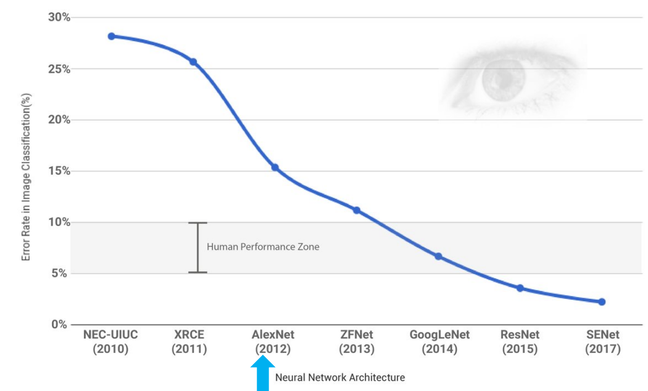
Around the year of 2012...

Speech Recognition: TIMIT



Computer Vision: ImageNet

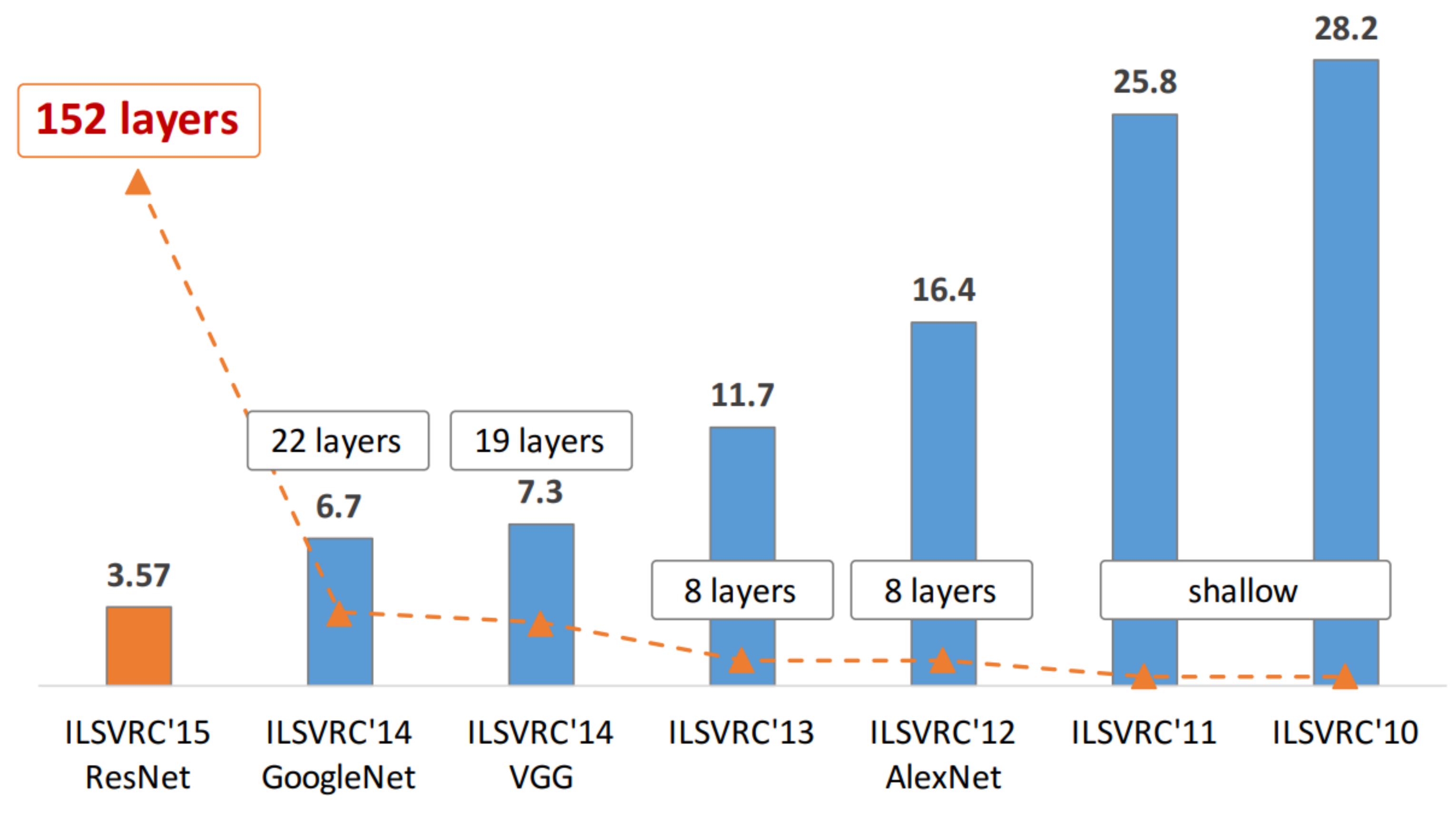
- ImageNet (subset):
  - 1.2 million training images
  - 100,000 test images
  - 1000 classes
- ImageNet large-scale visual recognition Challenge



source: https://www.linkedin.com/pulse/must-read-path-breaking-papers-image-classification-muktabh-mayank

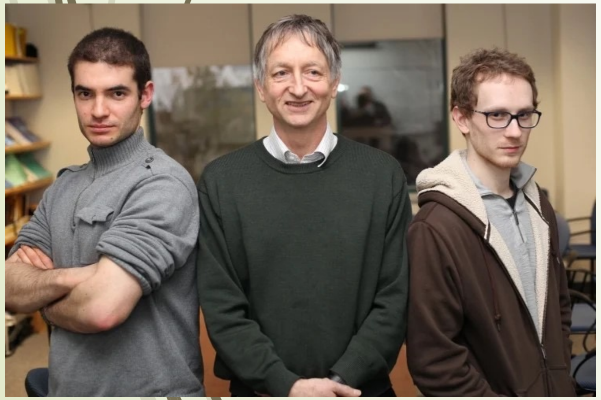
Deep Learning

Depth as function of year

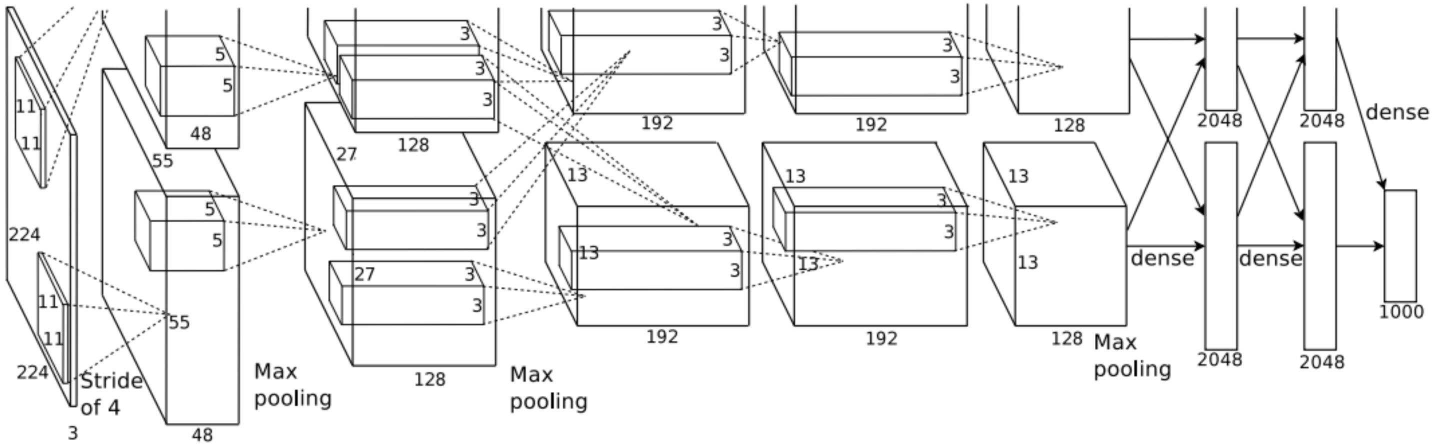


[He et al., 2016]

AlexNet (2012): Architecture



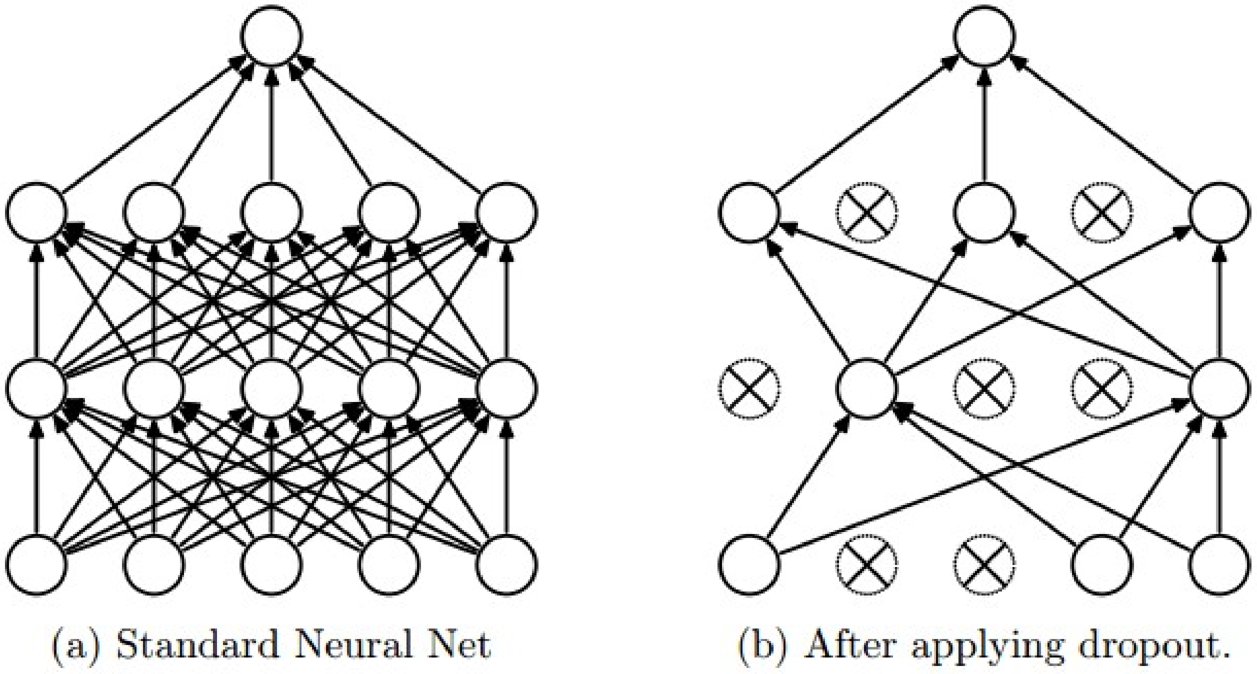
- 8 layers: first 5 convolutional, rest fully connected
- ReLU nonlinearity
- Local response normalization
- Max-pooling
- Dropout



Source: [Krizhevsky et al., 2012]

https://github.com/computerhistory/AlexNet-Source-Code

AlexNet (2012): Dropout

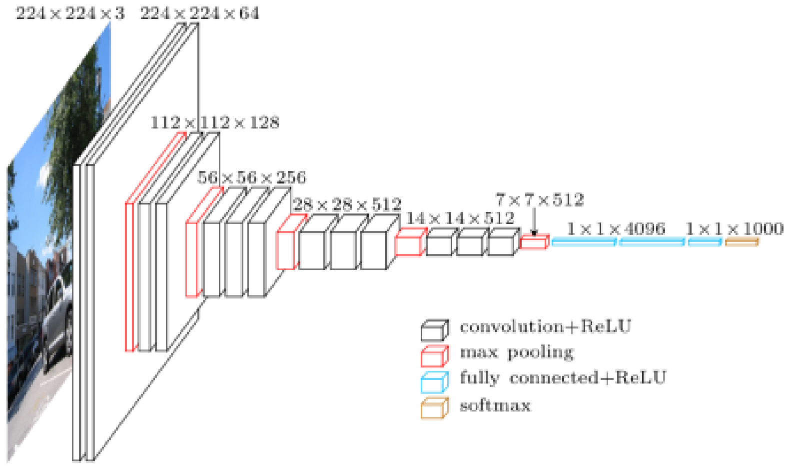


Source: [Srivastava et al., 2014]

- Zero every neuron with probability $1 - p$
- At test time, multiply every neuron by p

VGG (2014) [Simonyan-Zisserman'14]

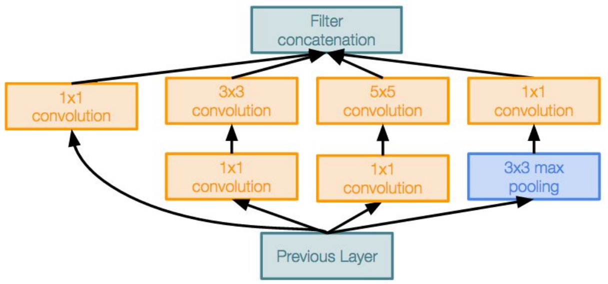
- Deeper than AlexNet: 11-19 layers versus 8
- No local response normalization
- Number of filters multiplied by two every few layers
- Spatial extent of filters 3×3 in all layers
- Instead of 7×7 filters, use three layers of 3×3 filters
  - Gain intermediate nonlinearity
  - Impose a regularization on the 7×7 filters



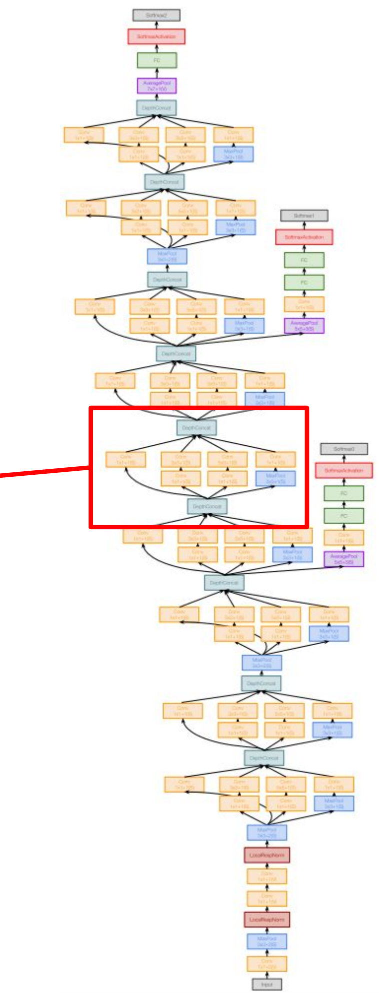
Source: `https://blog.heuritech.com/2016/02/29/`

GoogLeNet *[Szegedy et al., 2014]*

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
- 12x less than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

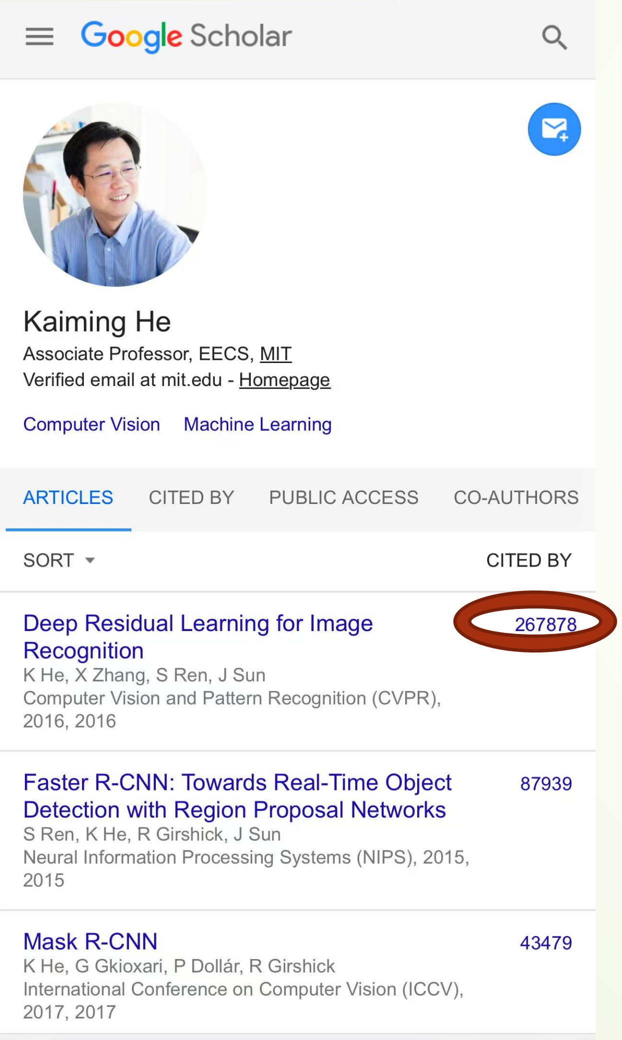


Inception module

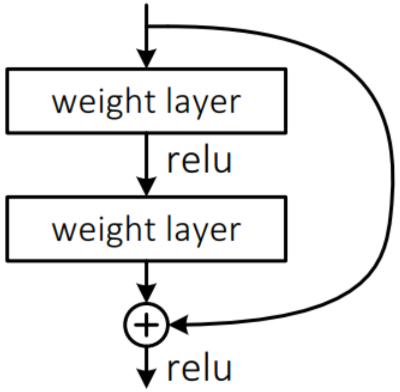


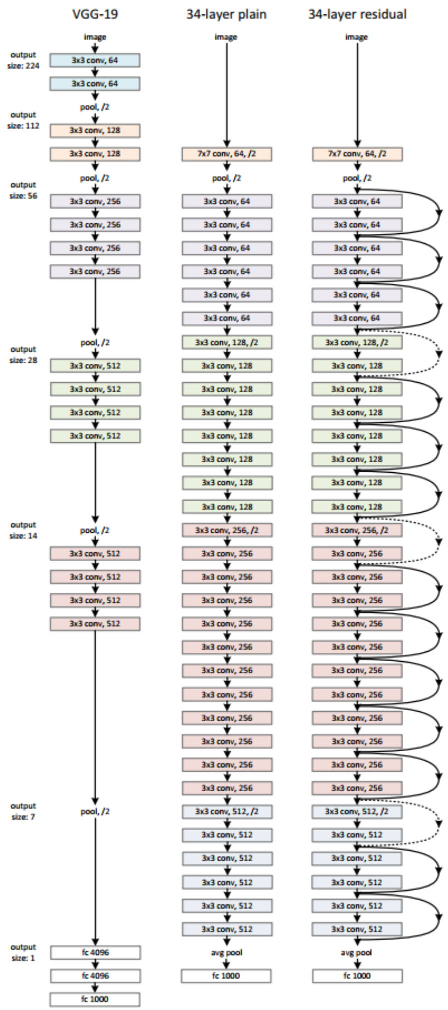
ResNet (2015) [HGRS-15]

ILSVRC'15 classification winner (3.57% top 5 error)



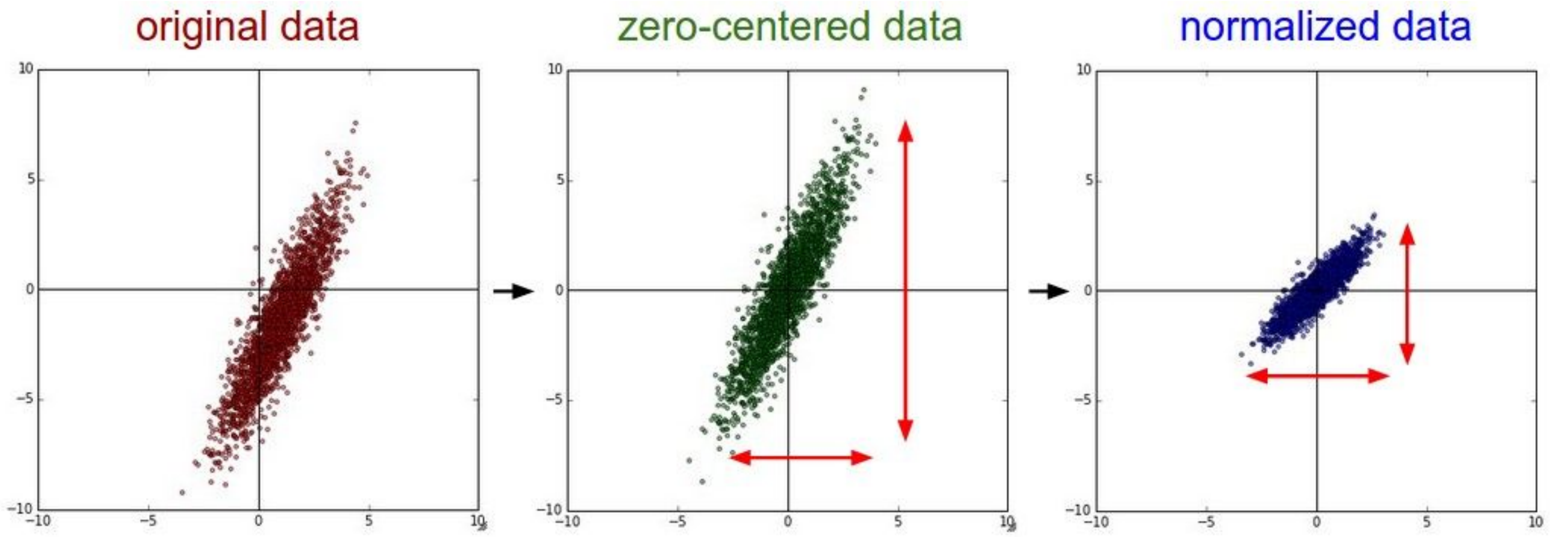
- Solves problem by adding skip connections
- Very deep: 152 layers
- No dropout
- Stride
- Batch normalization





Source: Deep Residual Learning for Image Recognition

Batch Normalization




```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

(Assume X [NxD] is data matrix, each example in a row)

Batch Normalization

Algorithm 2 Batch normalization [Ioffe and Szegedy, 2015]

Input: Values of x over minibatch $x_1 \dots x_B$, where x is a certain channel in a certain feature vector
Output: Normalized, scaled and shifted values $y_1 \dots y_B$

1: $\mu = \frac{1}{B} \sum_{b=1}^{B} x_b$
2: $\sigma^2 = \frac{1}{B} \sum_{b=1}^{B} (x_b - \mu)^2$
3: $\hat{x}_b = \frac{x_b - \mu}{\sqrt{\sigma^2 + \epsilon}}$
4: $y_b = \gamma \hat{x}_b + \beta$

- Accelerates training and makes initialization less sensitive
- Zero mean and unit variance feature vectors

BatchNorm at Test

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

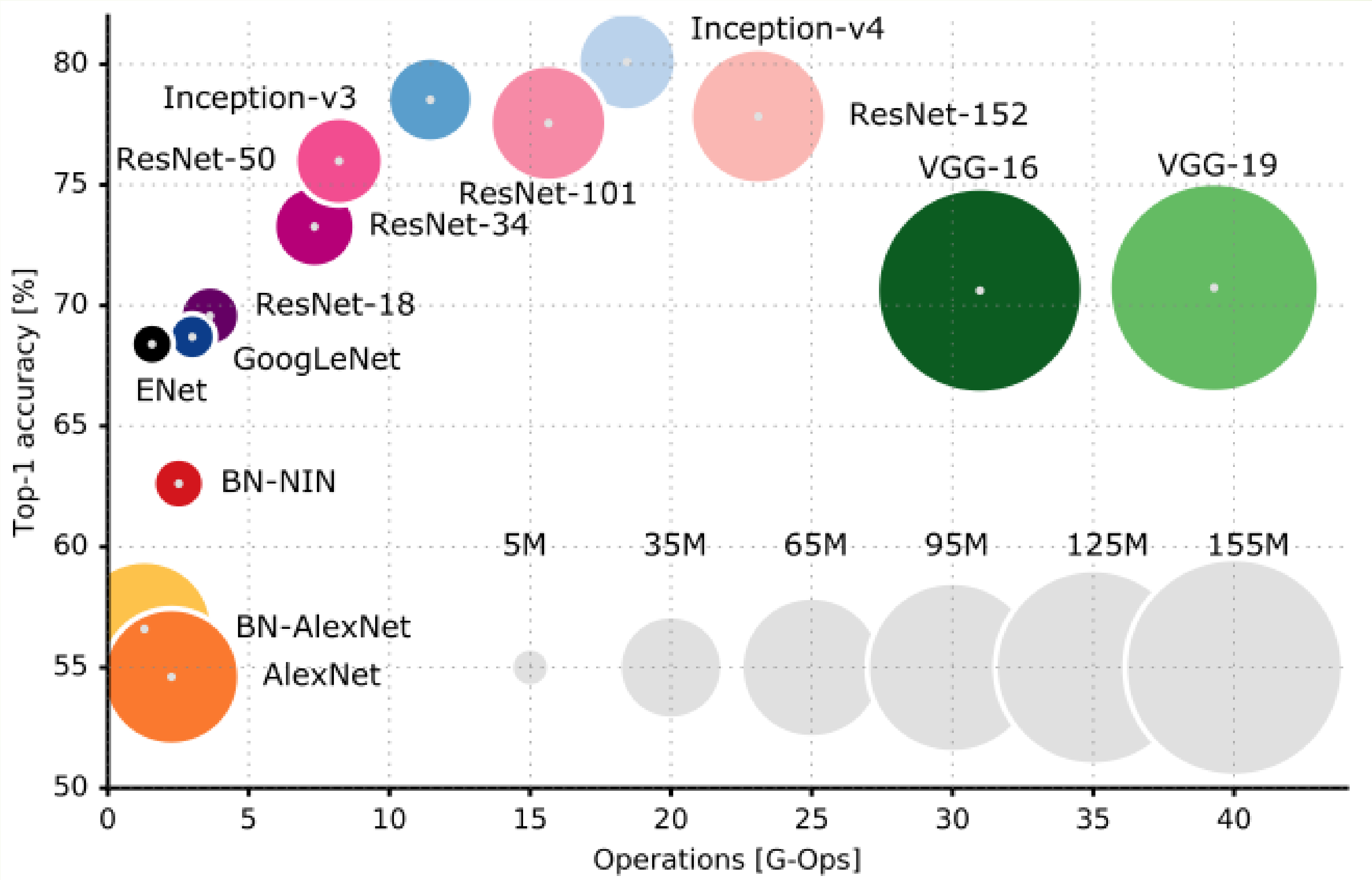
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

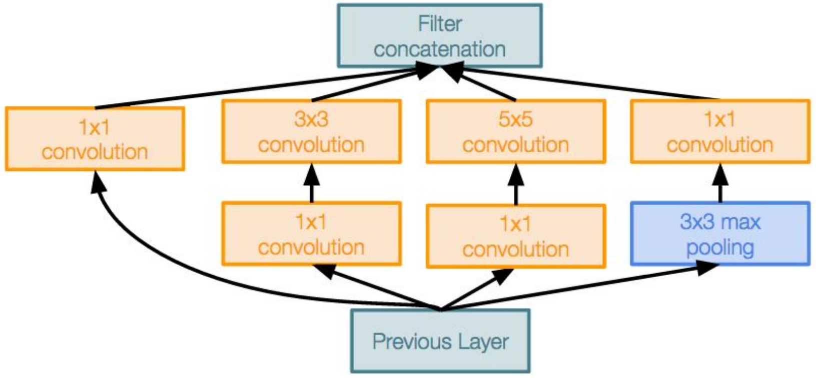
(e.g. can be estimated during training with running averages)

Complexity vs. Accuracy of Different Networks

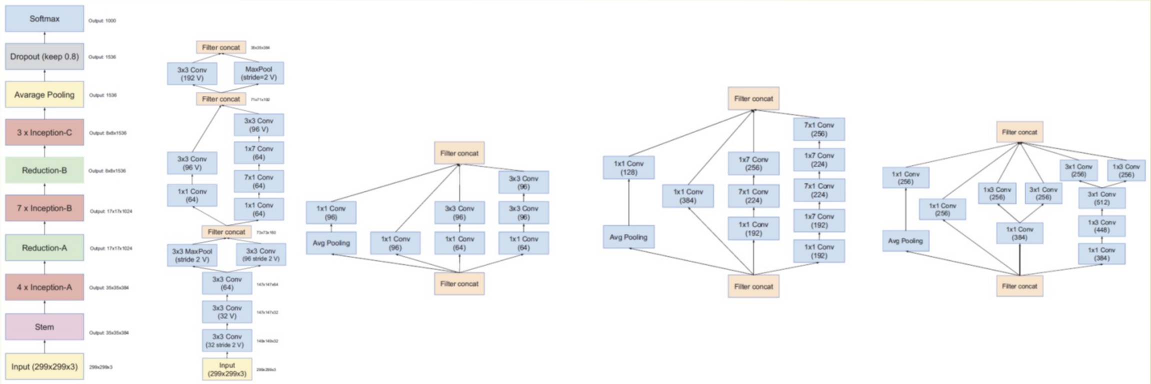


Inception-v4 = ResNet + Inception

- “**Inception**” module:
  - Introduced by *Szegedy et al., 2014* in ***GoogLeNet***
  - ILSVRC’14 classification winner (6.7% top 5 error)
- Apply parallel filter operations on the input from previous layer:
  - Dimensionality reduction (1x1 conv)
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise



Inception module



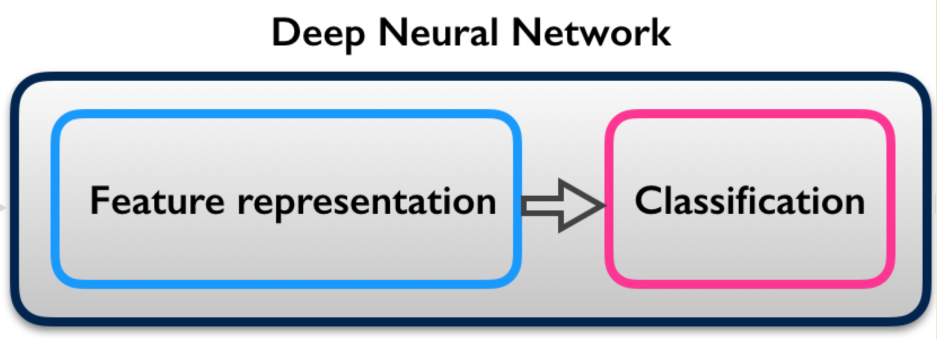
Deep Learning Softwares

- **Pytorch** (developed by Yann LeCun and Facebook):
  - http://pytorch.org/tutorials/
- Tensorflow (developed by Google based on Caffe)
  - https://www.tensorflow.org/tutorials/
- Theano (developed by Yoshua Bengio)
  - http://deeplearning.net/software/theano/tutorial/
- **Keras (based on Tensorflow or Pytorch)**
  - https://www.manning.com/books/deep-learning-with-python?a_aid=keras&a_bid=76564dff

Show some examples by jupyter notebooks…

Transfer Learning:
Feature Extraction and Fine Tuning

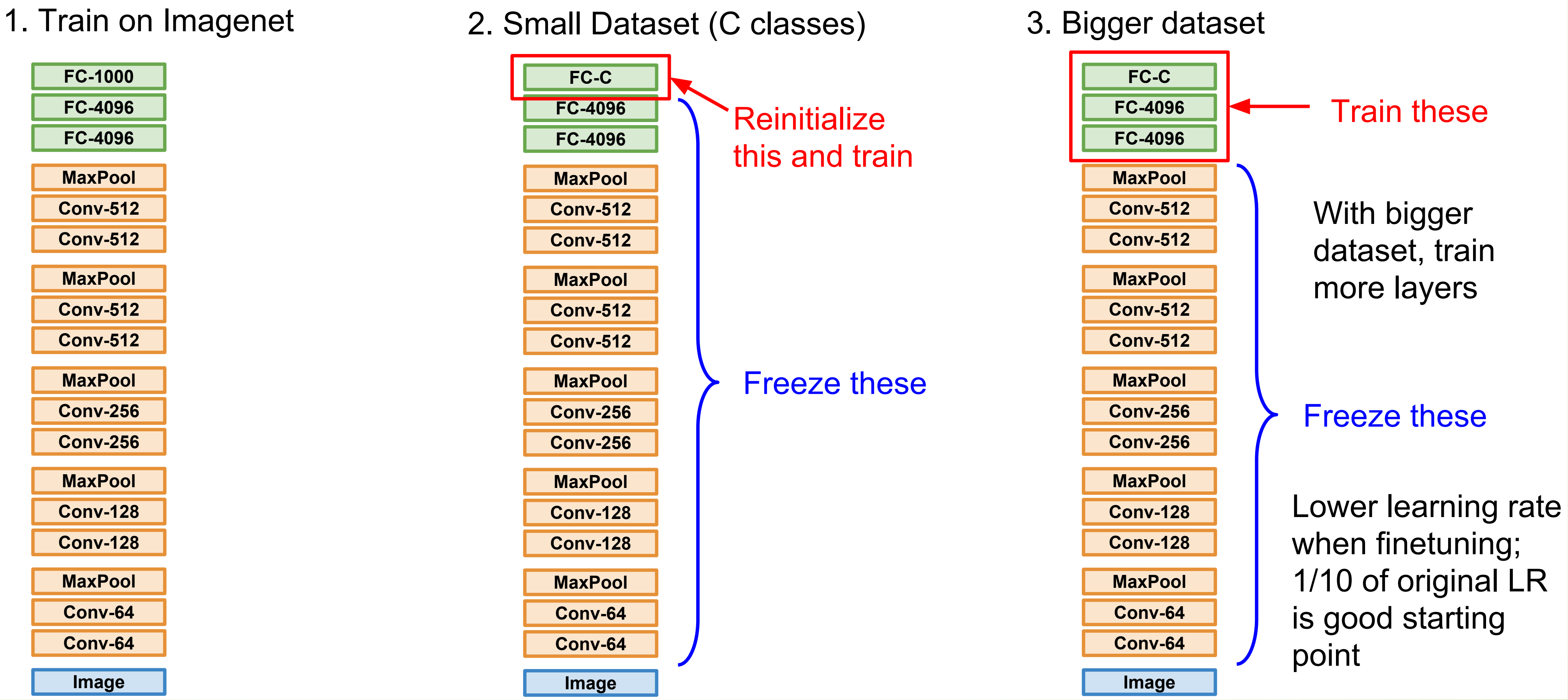
Transfer Learning?

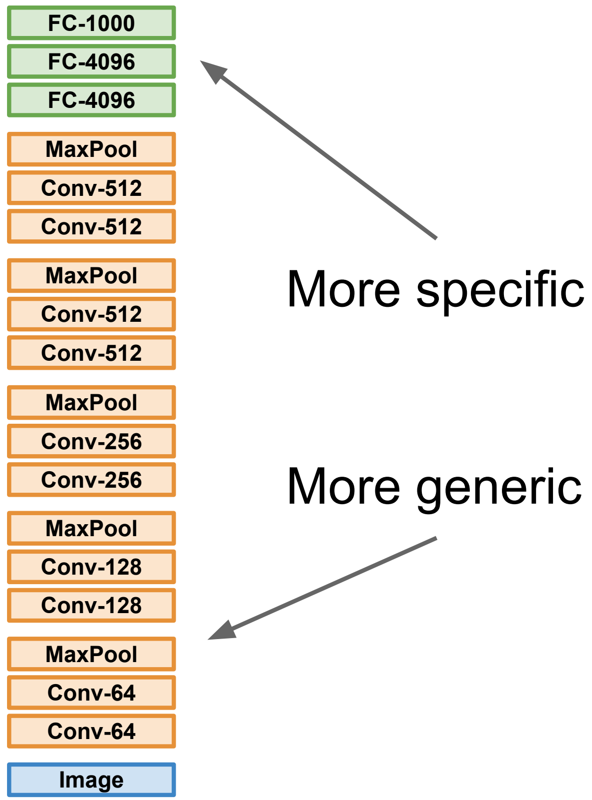


- Filters learned in first layers of a network are transferable from one task to another
- When solving another problem, no need to retrain the lower layers, just fine tune upper ones
- Is this simply due to the large amount of images in ImageNet?
- Does solving many classification problems simultaneously result in features that are more easily transferable?
- Does this imply filters can be learned in unsupervised manner?
- Can we characterize filters mathematically?

Transfer Learning with CNNs

Donahue et al, “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition”, ICML 2014
Razavian et al, “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014





	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

Summary

- Feature Extraction vs. Fine-Tuning:
  - Feature extraction usually refers to freeze the bottom (early layers) and retrain the top (last) layer
  - Fine-Tuning usually refers to retrain the last few layers or the whole network ninialized from pretrained parameters
  - They are both called transfer learning
- Jupyter notebook examples with pytorch:
  - https://github.com/aifin-hkust/aifin-hkust.github.io/blob/master/2020/notebook/finetuning_resnet.ipynb

Neural Collapse: in the zero-training loss phase

Papyan, Han, and Donoho (2020), PNAS. arXiv:2008.08186

Prevalence of neural collapse during the terminal phase of deep learning training

Vardan Papyan[a,1], X. Y. Han[b,1], and David L. Donoho[a,2]

[a]Department of Statistics, Stanford University, Stanford, CA 94305-4065; and [b]School of Operations Research and Information Engineering, Cornell University, Ithaca, NY 14850

Contributed by David L. Donoho, August 18, 2020 (sent for review July 22, 2020; reviewed by Helmut Boelsckei and Stéphane Mallat)

Modern practice for training classification deepnets involves a terminal phase of training (TPT), which begins at the epoch where training error first vanishes. During TPT, the training error stays effectively zero, while training loss is pushed toward zero. Direct measurements of TPT, for three prototypical deepnet architectures and across seven canonical classification datasets, expose a pervasive inductive bias we call *neural collapse* (NC), involving four deeply interconnected phenomena. (NC1) Cross-example within-class variability of last-layer training activations collapses to zero, as the individual activations themselves collapse to their class means. (NC2) The class means collapse to the vertices of a simplex equiangular tight frame (ETF). (NC3) Up to rescaling, the last-layer classifiers collapse to the class means or in other words, to the simplex ETF (i.e., to a self-dual configuration). (NC4) For a given activation, the classifier's decision collapses to simply choosing whichever class has the closest train class mean (i.e., the nearest class center [NCC] decision rule). The symmetric and very simple geometry induced by the TPT confers important benefits, including better generalization performance, better robustness, and better interpretability.

deep learning | inductive bias | adversarial robustness | simplex equiangular tight frame | nearest class center

1. Introduction

Over the last decade, deep learning systems have steadily advanced the state of the art in benchmark competitions, culminating in superhuman performance in tasks ranging from image classification to language translation to game play. One might expect the trained networks to exhibit many particularities—making it impossible to find any empirical regularities across a wide range of datasets and architectures. On the contrary, in this article we present extensive measurements across image classification datasets and architectures, exposing a common empirical pattern.

Our observations focus on today's standard training paradigm in deep learning, an accretion of several fundamental ingredients that developed over time. Networks are trained beyond zero misclassification error, approaching negligible cross-entropy loss and interpolating the in-sample training data; networks are overparameterized, making such memorization possible; and these parameters are layered in ever-growing depth, allowing for sophisticated feature engineering. A series of recent works (1–5) highlighted the paradigmatic nature of the practice of training well beyond zero error, seeking zero loss. We call the postzero-error phase the terminal phase of training (TPT).

A scientist with standard preparation in mathematical statistics might anticipate that the linear classifier resulting from this paradigm, being a by-product of such training, would be quite arbitrary and vary wildly—from instance to instance, dataset to dataset, and architecture to architecture—thereby displaying no underlying cross-situational invariant structure. The scientist might further expect that the configuration of the fully trained decision boundaries—and the underlying linear classifier defining those boundaries—would be quite arbitrary and vary chaotically from situation to situation. Such expectations might be supported by appealing to the overparameterized nature of the model, and to standard arguments whereby any noise in the data propagates during overparameterized training to generate disproportionate changes in the parameters being fit.

Defeating such expectations, we show here that TPT frequently induces an underlying mathematical simplicity to the trained deepnet model—and specifically to the classifier and last-layer activations—across many situations now considered canonical in deep learning. Moreover, the identified structure naturally suggests performance benefits. Additionally, indeed, we show that convergence to this rigid structure tends to occur simultaneously with improvements in the network's generalization performance as well as adversarial robustness.

We call this process **neural collapse** (NC) and characterize it by four manifestations in the classifier and last-layer activations:

(NC1) **Variability collapse:** as training progresses, the within-class variation of the activations becomes negligible as these activations collapse to their class means.

(NC2) **Convergence to simplex equiangular tight frame (ETF):** the vectors of the class means (after centering by their global mean) converge to having equal length, forming equal-sized angles between any given pair, and being the maximally pairwise-distanced configuration constrained to the previous two properties. This configuration is identical to a previously studied configuration in the

Significance

Modern deep neural networks for image classification have achieved superhuman performance. Yet, the complex details of trained networks have forced most practitioners and researchers to regard them as black boxes with little that could be understood. This paper considers in detail a now-standard training methodology: driving the cross-entropy loss to zero, continuing long after the classification error is already zero. Applying this methodology to an authoritative collection of standard deepnets and datasets, we observe the emergence of a simple and highly symmetric geometry of the deepnet features and of the deepnet classifier, and we document important benefits that the geometry conveys—thereby helping us understand an important component of the modern deep learning training paradigm.

Author contributions: V.P., X.Y.H., and D.L.D. designed research, performed research, analyzed data, provided mathematical analysis, and wrote the paper.

Reviewers: H.B., ETH Zurich; and S.M., Collège de France.

The authors declare no competing interest.

See online for related content such as Commentaries.

[1]V.P. and X.Y.H. contributed equally to this work.

[2]To whom correspondence may be addressed. Email: donoho@stanford.edu.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2015509117/-/DCSupplemental.

First published September 21, 2020.

Neural Collapse phenomena, in post-zero-training-error phase

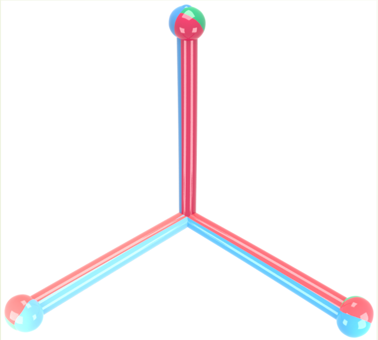
- **(NC1) Variability collapse:** As training progresses, the within-class variation of the activations becomes negligible as these activations collapse to their class-means.
- **(NC2) Convergence to Simplex ETF:** The vectors of the class-means (after centering by their global-mean) converge to having equal length, forming equal-sized angles between any given pair, and being the maximally pairwise-distanced configuration constrained to the previous two properties. This configuration is identical to a previously studied configuration in the mathematical sciences known as Simplex **Equiangular Tight Frame (ETF).**

- *Papyan, Han, and Donoho (2020), PNAS. arXiv:2008.0818**6***
- Visualization: https://purl.stanford.edu/br193mh4244

Definition 1 (**Simplex ETF**). A *standard* Simplex ETF is a collection of points in $\mathbb{R}^C$ specified by the columns of

$$\boldsymbol{M}^\star = \sqrt{\frac{C}{C-1}}\left(\boldsymbol{I} - \frac{1}{C}\mathbb{1}\mathbb{1}^\top\right), \qquad [1]$$

where $\boldsymbol{I} \in \mathbb{R}^{C\times C}$ is the identity matrix, and $\mathbb{1}_C \in \mathbb{R}^C$ is the ones vector. In this paper, we allow other poses, as well as rescaling, so the *general* Simplex ETF consists of the points specified by the columns of $\boldsymbol{M} = \alpha\boldsymbol{U}\boldsymbol{M}^\star \in \mathbb{R}^{p\times C}$, where $\alpha \in \mathbb{R}_+$ is a scale factor, and $\boldsymbol{U} \in \mathbb{R}^{p\times C}$ $(p \geq C)$ is a partial orthogonal matrix $(\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I})$.



Be careful!

- For imbalanced (long tail) classifications, **minority** classes may **collapse** and be absorbed by the majority classes
- *Fang, He, Long, Su, PNAS 2021, 118(43):e2103091118*

Exploring deep neural networks via layer-peeled model: Minority collapse in imbalanced training

Cong Fang[a,1], Hangfeng He[a], Qi Long[b], and Weijie J. Su[c,2]

[a]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104; [b]Department of Biostatistics, Epidemiology, and Informatics, University of Pennsylvania, Philadelphia, PA 19104; and [c]Department of Statistics and Data Science, University of Pennsylvania, Philadelphia, PA 19104

Edited by David L. Donoho, Stanford University, Stanford, CA, and approved August 30, 2021 (received for review February 15, 2021)

In this paper, we introduce the *Layer-Peeled Model*, a nonconvex, yet analytically tractable, optimization program, in a quest to better understand deep neural networks that are trained for a sufficiently long time. As the name suggests, this model is derived by isolating the topmost layer from the remainder of the neural network, followed by imposing certain constraints separately on the two parts of the network. We demonstrate that the Layer-Peeled Model, albeit simple, inherits many characteristics of well-trained neural networks, thereby offering an effective tool for explaining and predicting common empirical patterns of deep-learning training. First, when working on class-balanced datasets, we prove that any solution to this model forms a simplex equiangular tight frame, which, in part, explains the recently discovered phenomenon of neural collapse [V. Papyan, X. Y. Han, D. L. Donoho, *Proc. Natl. Acad. Sci. U.S.A.* 117, 24652–24663 (2020)]. More importantly, when moving to the imbalanced case, our analysis of the Layer-Peeled Model reveals a hitherto-unknown phenomenon that we term *Minority Collapse*, which fundamentally limits the performance of deep-learning models on the minority classes. In addition, we use the Layer-Peeled Model to gain insights into how to mitigate Minority Collapse. Interestingly, this phenomenon is first predicted by the Layer-Peeled Model before being confirmed by our computational experiments.

deep learning | neural collapse | class imbalance

Introduction

In the past decade, deep learning has achieved remarkable performance across a range of scientific and engineering domains (1–3). Interestingly, these impressive accomplishments were mostly achieved by heuristics and tricks, though often plausible, without much principled guidance from a theoretical perspective. On the flip side, however, this reality suggests the great potential a theory could have for advancing the development of deep-learning methodologies in the coming decade.

Unfortunately, it is not easy to develop a theoretical foundation for deep learning. Perhaps the most difficult hurdle lies in the nonconvexity of the optimization problem for training neural networks, which, loosely speaking, stems from the interaction between different layers of neural networks. To be more precise, consider a neural network for K-class classification (in logits), which in its simplest form reads*

$$f(x; W_{\mathrm{full}}) = b_L + W_L\sigma(b_{L-1} + W_{L-1}\sigma(\cdots\sigma(b_1 + W_1 x)\cdots)).$$

Here, $W_{\mathrm{full}} := \{W_1, W_2, \ldots, W_L\}$ denotes the weights of the L layers, $\{b_1, b_2, \ldots, b_L\}$ denotes the biases, and $\sigma(\cdot)$ is a nonlinear activation function such as the rectified linear unit (ReLU). Owing to the complex and nonlinear interaction between the L layers, when applying stochastic gradient descent to the optimization problem

$$\min_{W_{\mathrm{full}}} \frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{n_k}\mathcal{L}(f(x_{k,i}; W_{\mathrm{full}}), y_k) + \frac{\lambda}{2}\|W_{\mathrm{full}}\|^2, \qquad [1]$$

with a loss function $\mathcal{L}$ for training the neural network, it becomes very difficult to pinpoint how a given layer influences the output f (above, $\{x_{k,i}\}_{i=1}^{n_k}$ denotes the training examples in the k-th class, with label y_k, $N = n_1 + \cdots + n_K$ is the total number of training examples, $\lambda > 0$ is the weight decay parameter, and $\|\cdot\|$ throughout the paper is the ℓ_2 norm). Worse, this difficulty in analyzing deep-learning models is compounded by an ever-growing number of layers.

Therefore, any attempt to develop a tractable and comprehensive theory for demystifying deep learning would presumably first need to simplify the interaction between a large number of layers. Following this intuition, in this paper, we introduce the following optimization program as a *surrogate* model for Eq. **1** with the goal of unveiling quantitative patterns of deep neural networks:

$$\min_{W_L, H} \frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{n_k}\mathcal{L}(W_L h_{k,i}, y_k)$$
$$\text{s.t. } \frac{1}{K}\sum_{k=1}^{K}\|w_k\|^2 \le E_W, \quad \frac{1}{K}\sum_{k=1}^{K}\frac{1}{n_k}\sum_{i=1}^{n_k}\|h_{k,i}\|^2 \le E_H, \qquad [2]$$

Significance

The remarkable development of deep learning over the past decade relies heavily on sophisticated heuristics and tricks. To better exploit its potential in the coming decade, perhaps a rigorous framework for reasoning about deep learning is needed, which, however, is not easy to build due to the intricate details of neural networks. For near-term purposes, a practical alternative is to develop a mathematically tractable surrogate model, yet maintaining many characteristics of neural networks. This paper proposes a model of this kind that we term the Layer-Peeled Model. The effectiveness of this model is evidenced by, among others, its ability to reproduce a known empirical pattern and to predict a hitherto-unknown phenomenon when training deep-learning models on imbalanced datasets.

Author contributions: C.F., H.H., Q.L., and W.J.S. designed research; C.F., H.H., and W.J.S. performed research; C.F., H.H., and W.J.S. contributed new reagents/analytic tools; C.F., H.H., and W.J.S. analyzed data; and C.F., H.H., Q.L., and W.J.S. wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]Present address: Department of Key Laboratory of Machine Perception, Peking University, Beijing 100871, China.

[2]To whom correspondence may be addressed. Email: suw@wharton.upenn.edu.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2103091118/-/DCSupplemental.

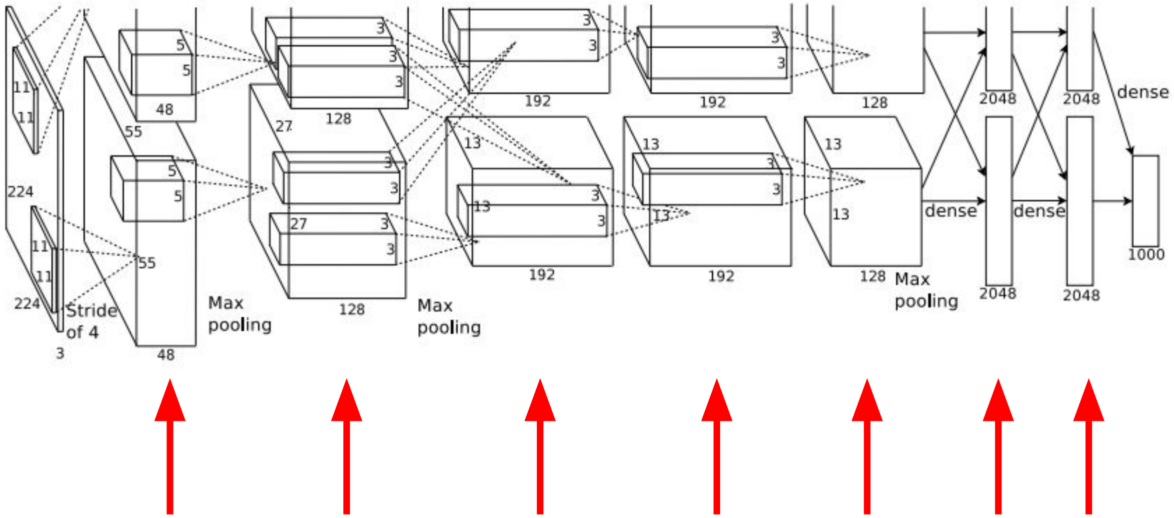
Published October 20, 2021.

*The softmax step is implicitly included in the loss function, and we omit other operations such as max-pooling for simplicity.

Visualizing Convolutional Networks

Understanding intermediate neurons?

This image is CC0 public domain



Input Image:
3 x 224 x 224

Class Scores:
1000 numbers

What are the intermediate features looking for?

Visualizing CNN Features: Gradient Ascent

- **Gradient ascent:** Generate a synthetic image that maximally activates a neuron

$$I^* = \arg\max_I \boxed{f(I)} + \boxed{R(I)}$$

Neuron value

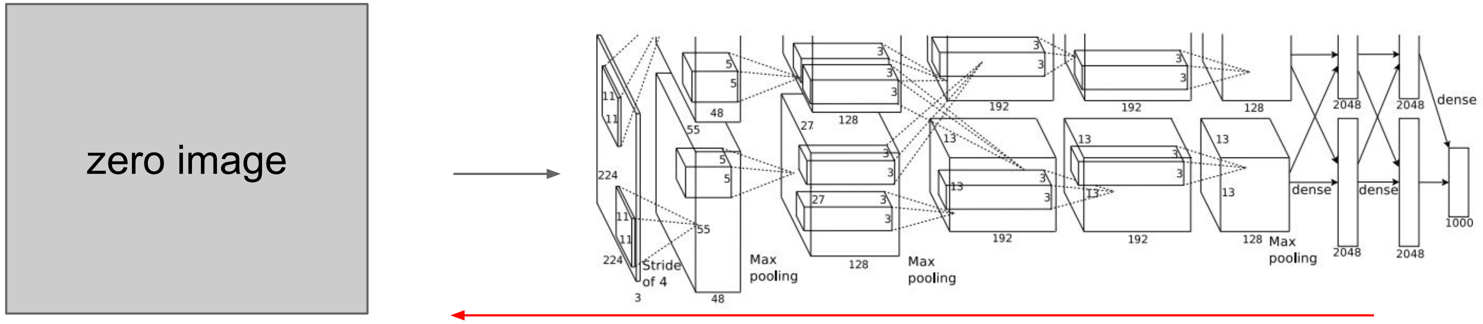
Natural image regularizer

Example: Class Visualizion of CNN via Gradient Ascent

$$\arg\max_{I} \boxed{S_c(I)} - \lambda \|I\|_2^2$$

score for class c (before Softmax)

1. Initialize image to zeros



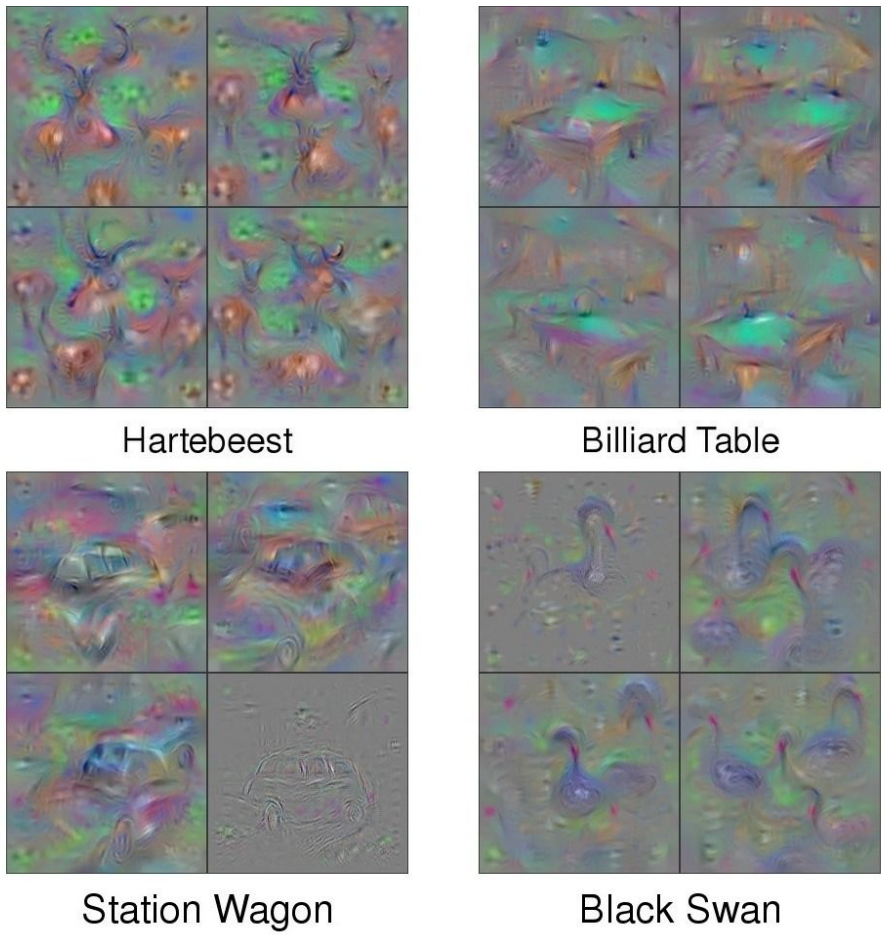
Repeat:
2. Forward image to compute current scores
3. Backprop to get gradient of neuron value with respect to image pixels
4. Make a small update to the image

Visualizing CNN Features: Gradient Ascent

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Better regularizer: Penalize L2 norm of image; also during optimization periodically

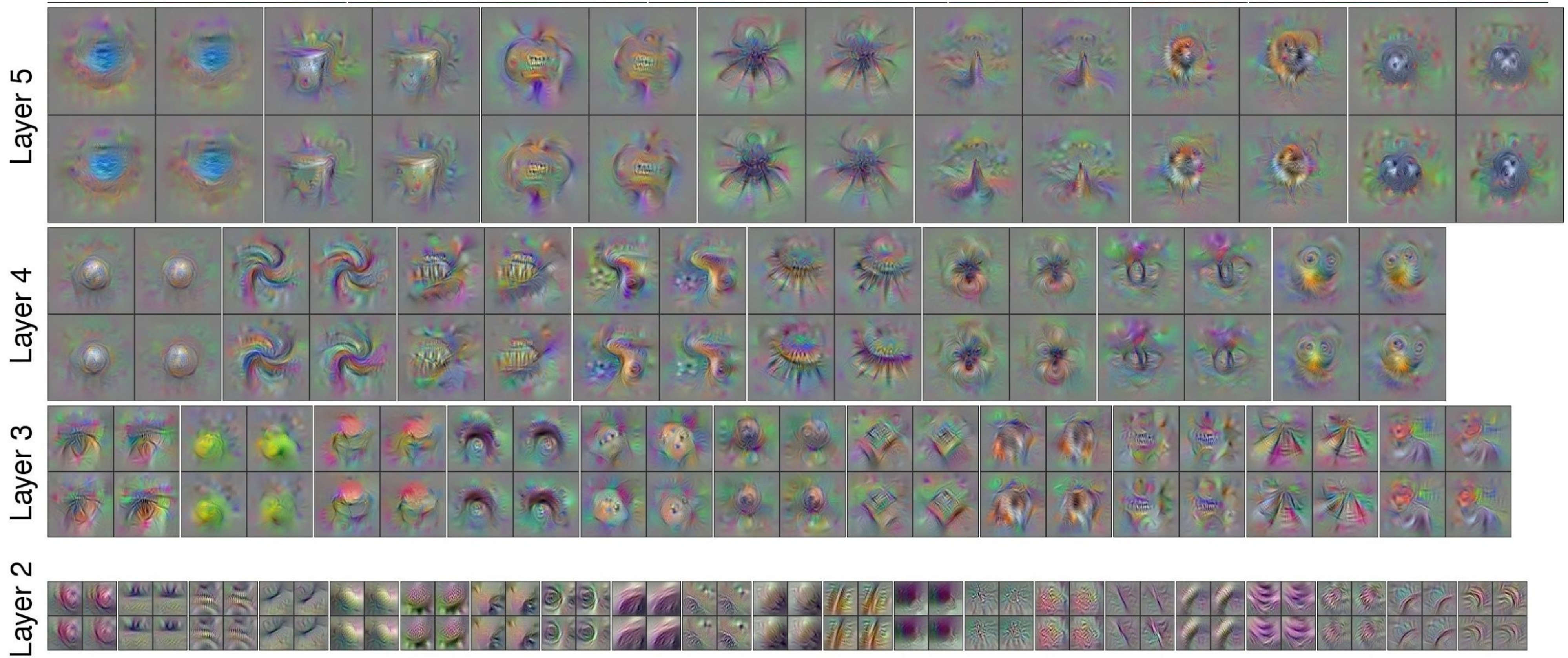
(1) Gaussian blur image
(2) Clip pixels with small values to 0
(3) Clip pixels with small gradients to 0



Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2014.

Visualizing CNN Features: Gradient Ascent

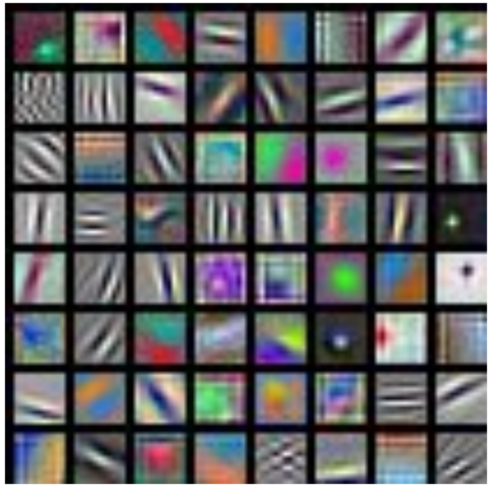
Use the same approach to visualize intermediate features



Yosinski et al, “Understanding Neural Networks Through Deep Visualization”, ICML DL Workshop 2014.

It's easy to visualize early layers

First Layer: Visualize Filters

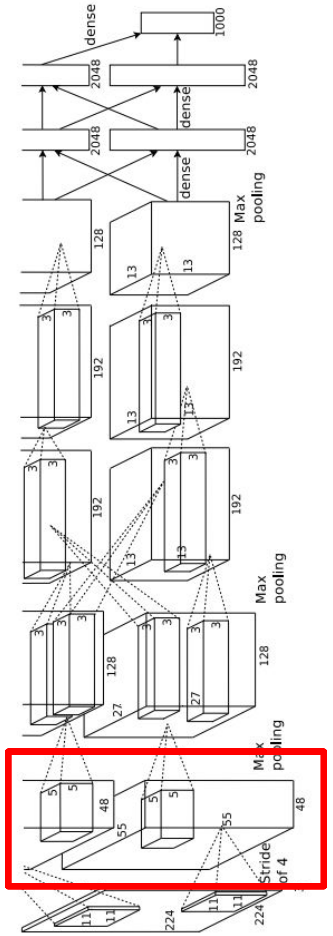


AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7



Krizhevsky, "One weird trick for parallelizing convolutional neural networks", arXiv 2014
He et al, "Deep Residual Learning for Image Recognition", CVPR 2016
Huang et al, "Densely Connected Convolutional Networks", CVPR 2017

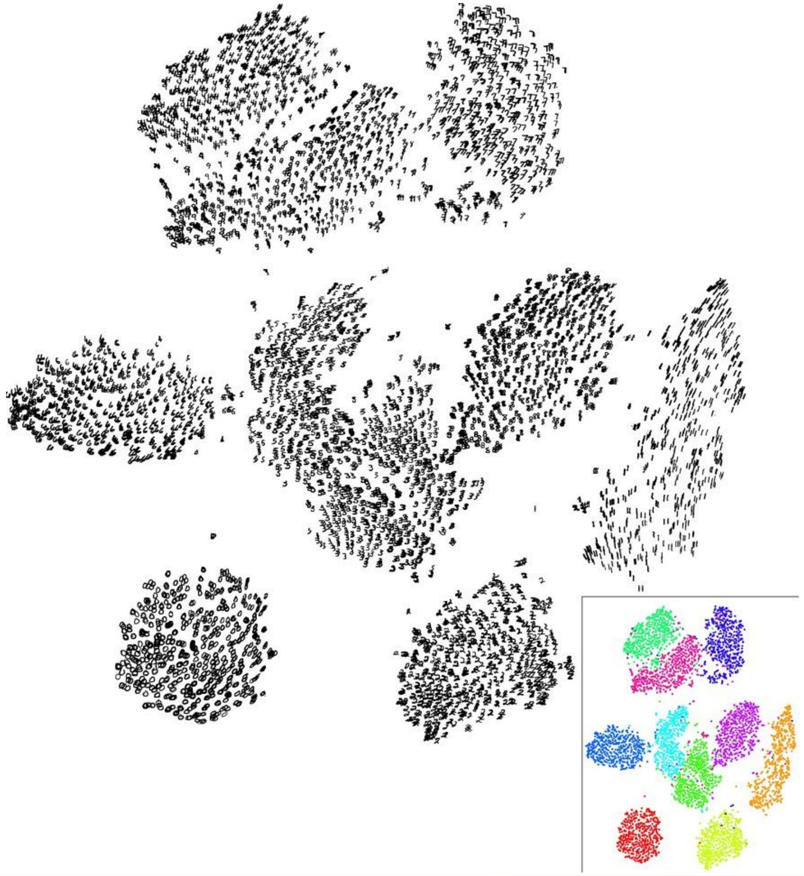
Last layers are hard to visualize

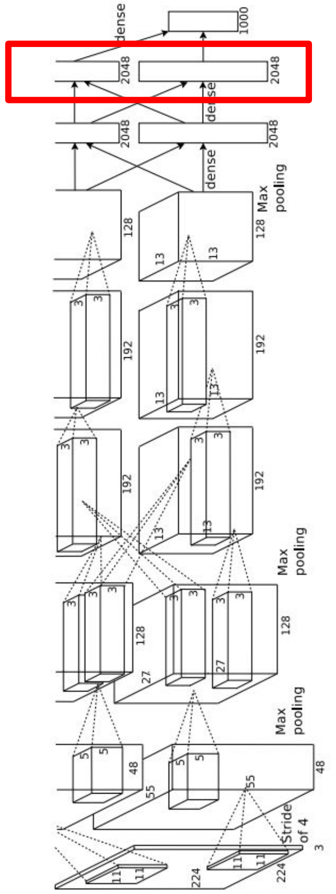
Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions

Simple algorithm: Principle Component Analysis (PCA)

More complex: **t-SNE**

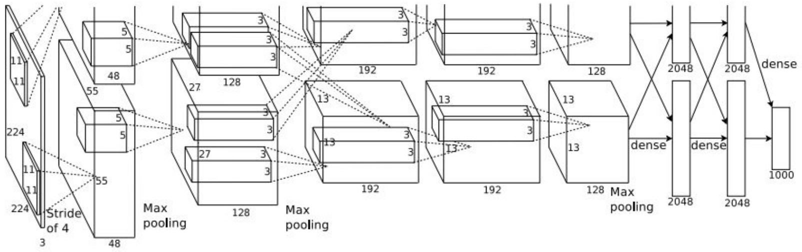




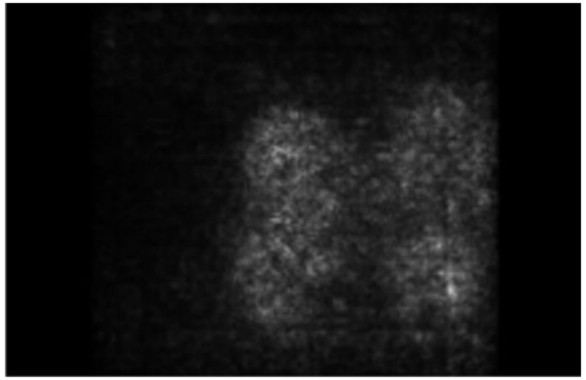
Van der Maaten and Hinton, “Visualizing Data using t-SNE”, JMLR 2008

Saliency Maps

How to tell which pixels matter for classification?



Dog

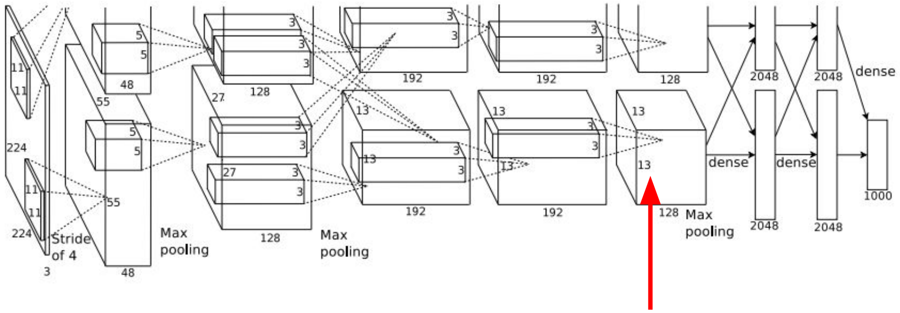


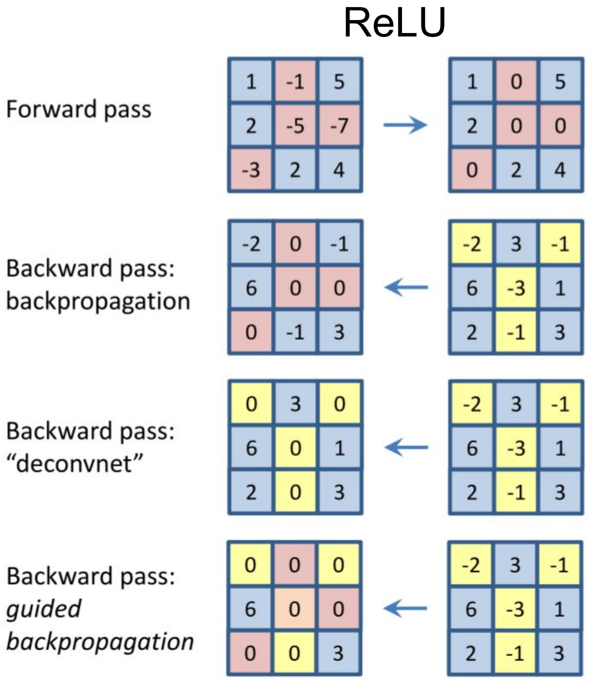
Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max over RGB channels

Simonyan, Vedaldi, and Zisserman, "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps", ICLR Workshop 2014.

Guided BP

Intermediate features via (guided) backprop





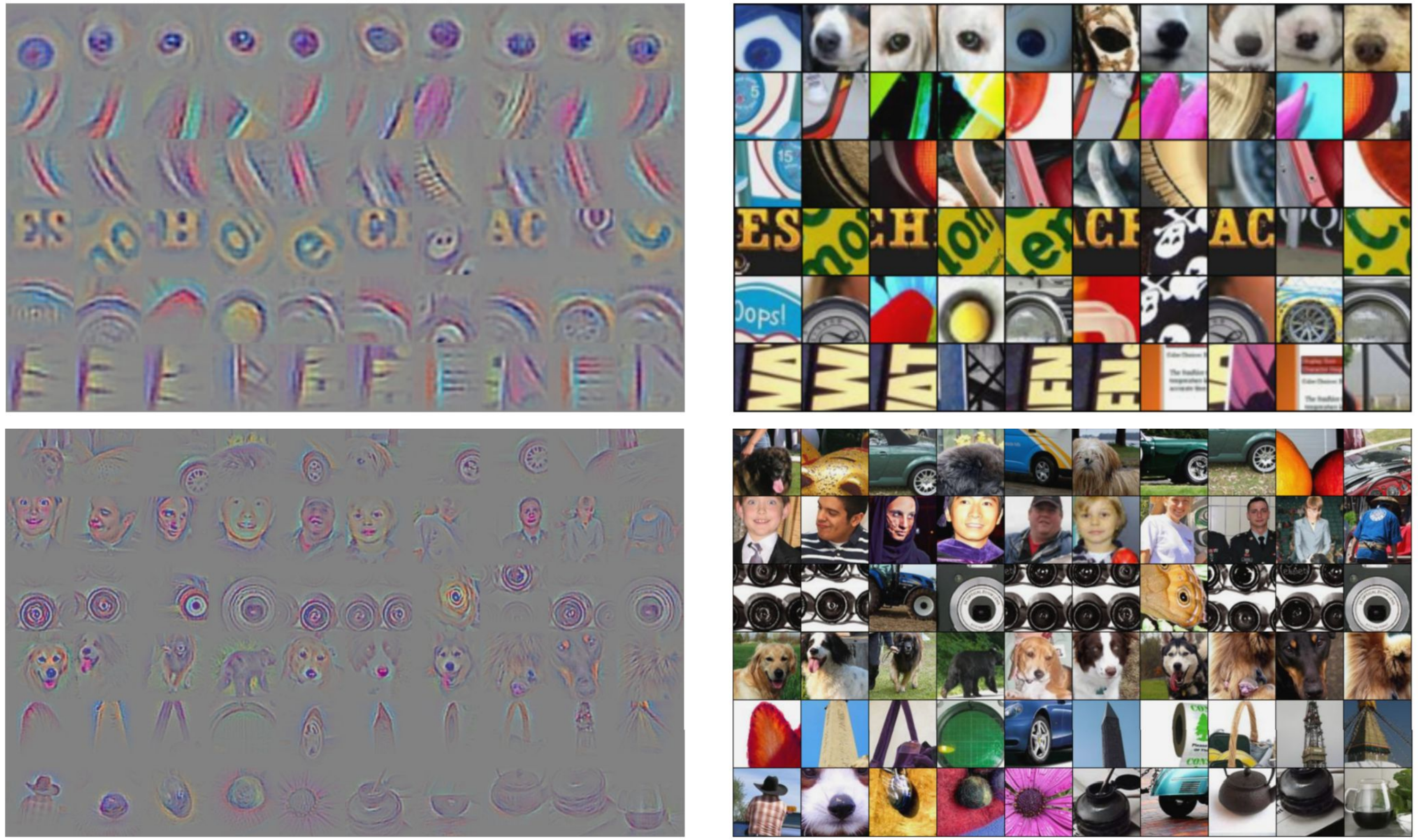
Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of neuron value with respect to image pixels

Images come out nicer if you only backprop positive gradients through each ReLU (guided backprop)

Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014
Springenberg et al, "Striving for Simplicity: The All Convolutional Net", ICLR Workshop 2015

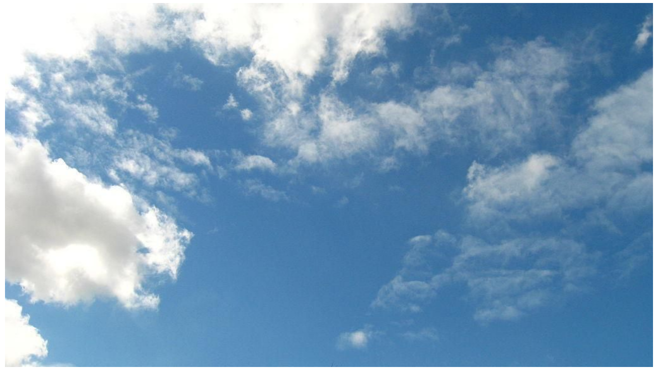
Intermediate features via Guided BP

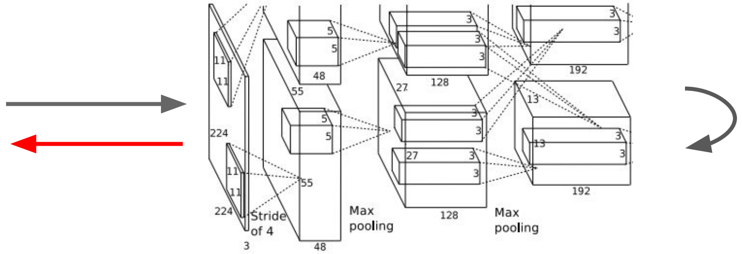


Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Springenberg et al, “Striving for Simplicity: The All Convolutional Net”, ICLR Workshop 2015

DeepDream: amplifying features

Rather than synthesizing an image to maximize a specific neuron, instead try to **amplify** the neuron activations at some layer in the network





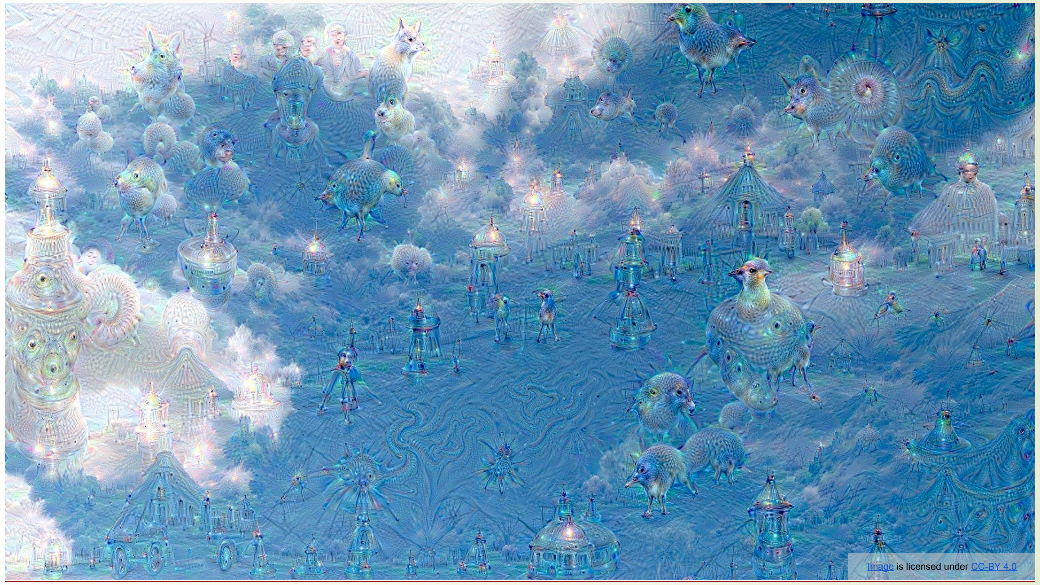
Choose an image and a layer in a CNN; repeat:

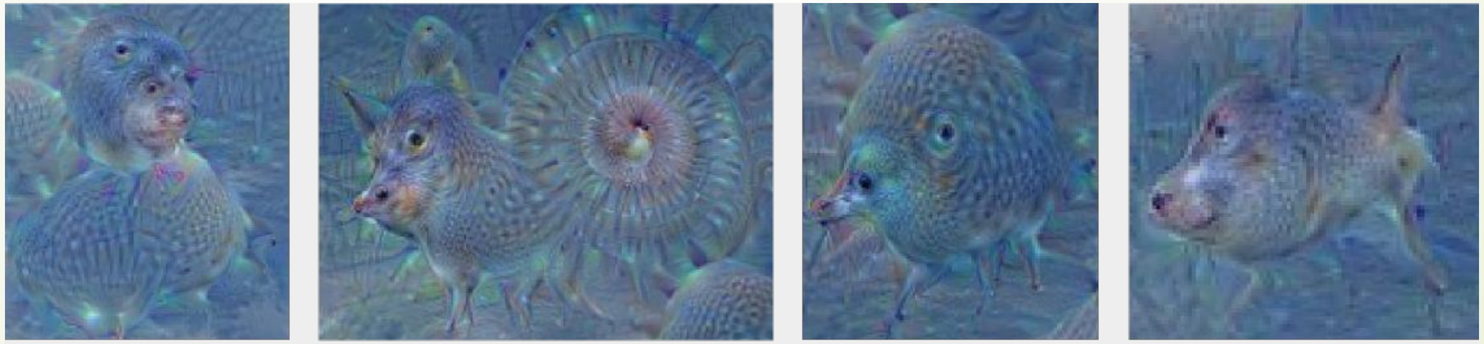
1. Forward: compute activations at chosen layer
2. Set gradient of chosen layer *equal to its activation*
3. Backward: Compute gradient on image
4. Update image

Equivalent to:

$$I^* = \arg\max_I \sum_i f_i(I)^2$$

Example: DeepDream of Sky





"Admiral Dog!" "The Pig-Snail" "The Camel-Bird" "The Dog-Fish"

More Examples

Python Notebooks

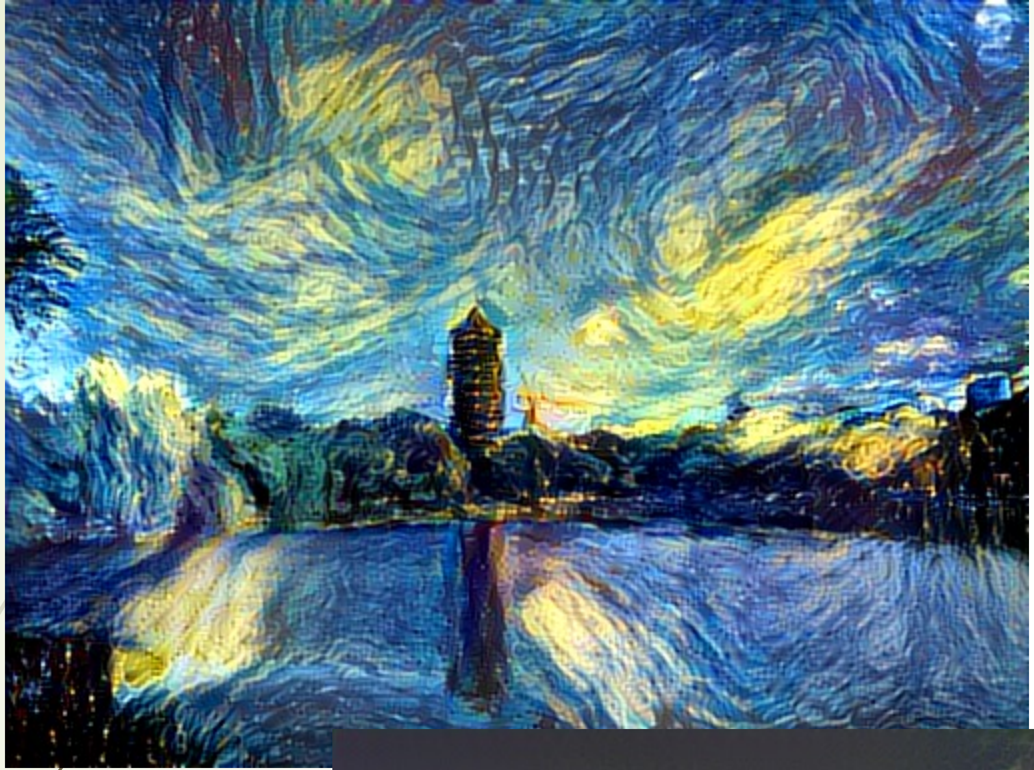
- An interesting Pytorch Implementation of these visualizatoin methods
  - https://github.com/utkuozbulak/pytorch-cnn-visualizations

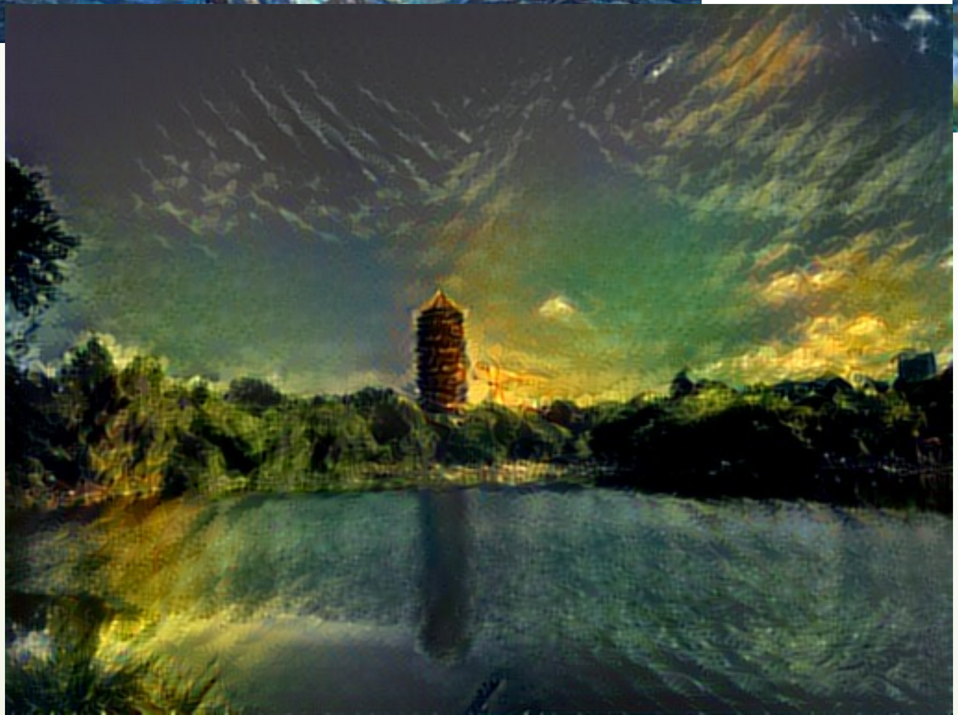
- Some examples demo:
  - https://github.com/aifin-hkust/aifin-hkust.github.io/blob/master/2020/notebook/vgg16-visualization.ipynb
  - https://github.com/aifin-hkust/aifin-hkust.github.io/blob/master/2020/notebook/vgg16-heatmap.ipynb

Neural Style

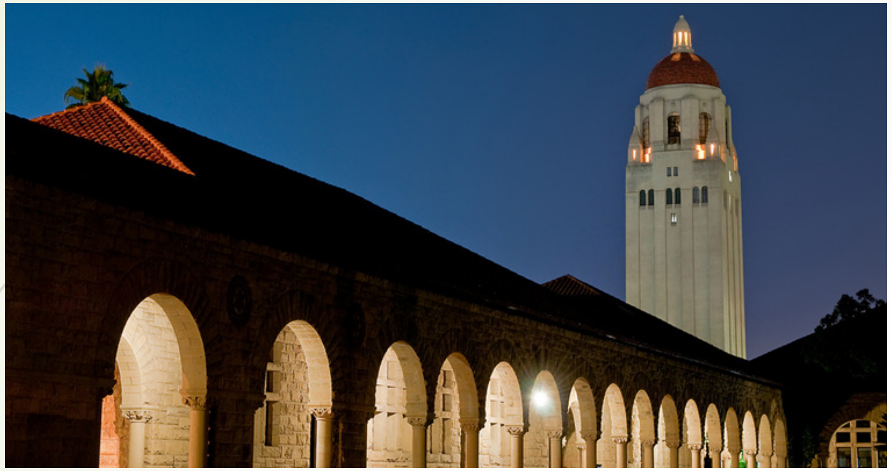
Example: The Noname Lake in PKU

Left: *Vincent Van Gogh*, Starry Night
Right: *Claude Monet*, Twilight Venice
Bottom: *William Turner*, Ship Wreck





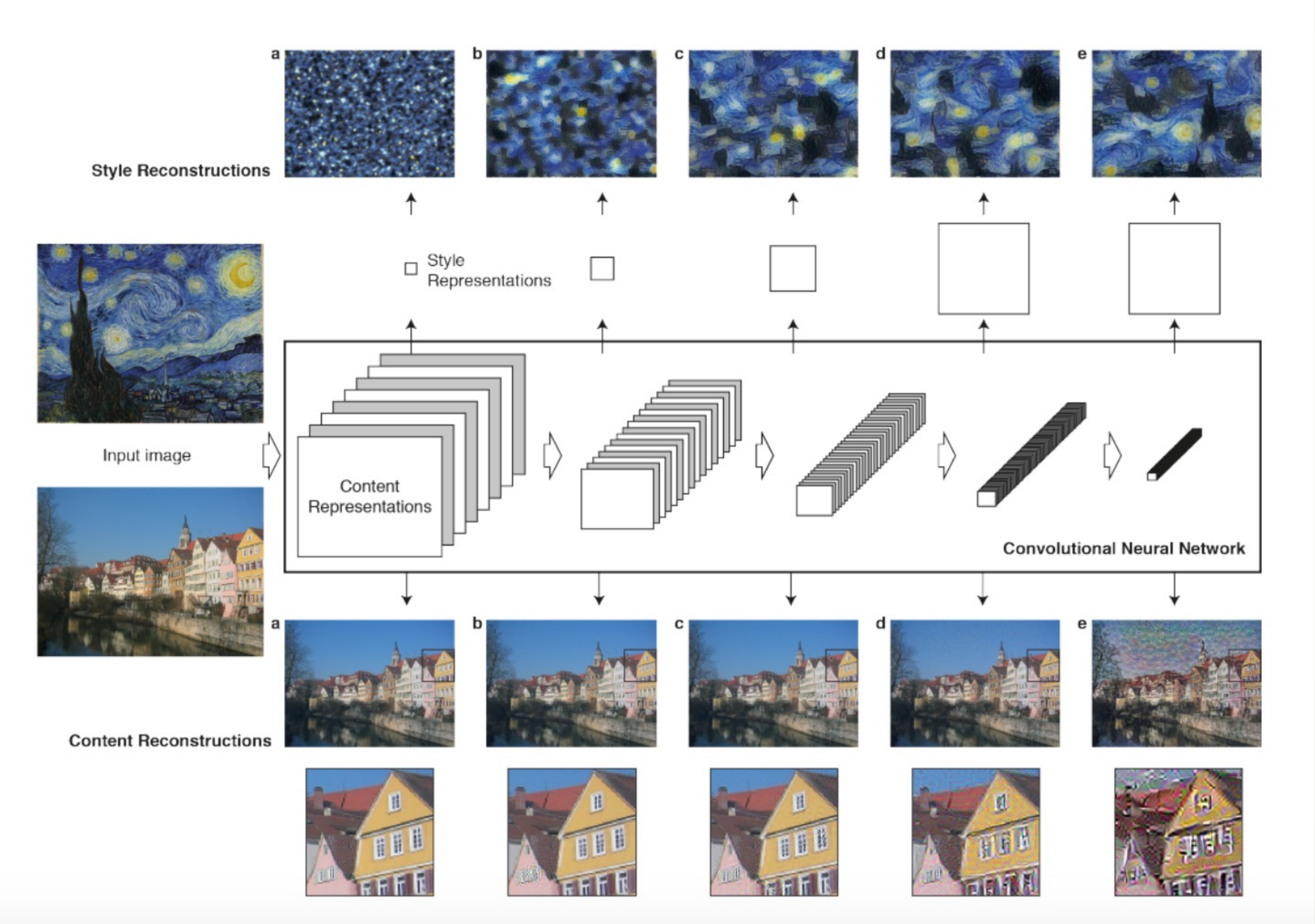
Application of Deep Learning: Content-Style synthetic pictures
By “neural-style”



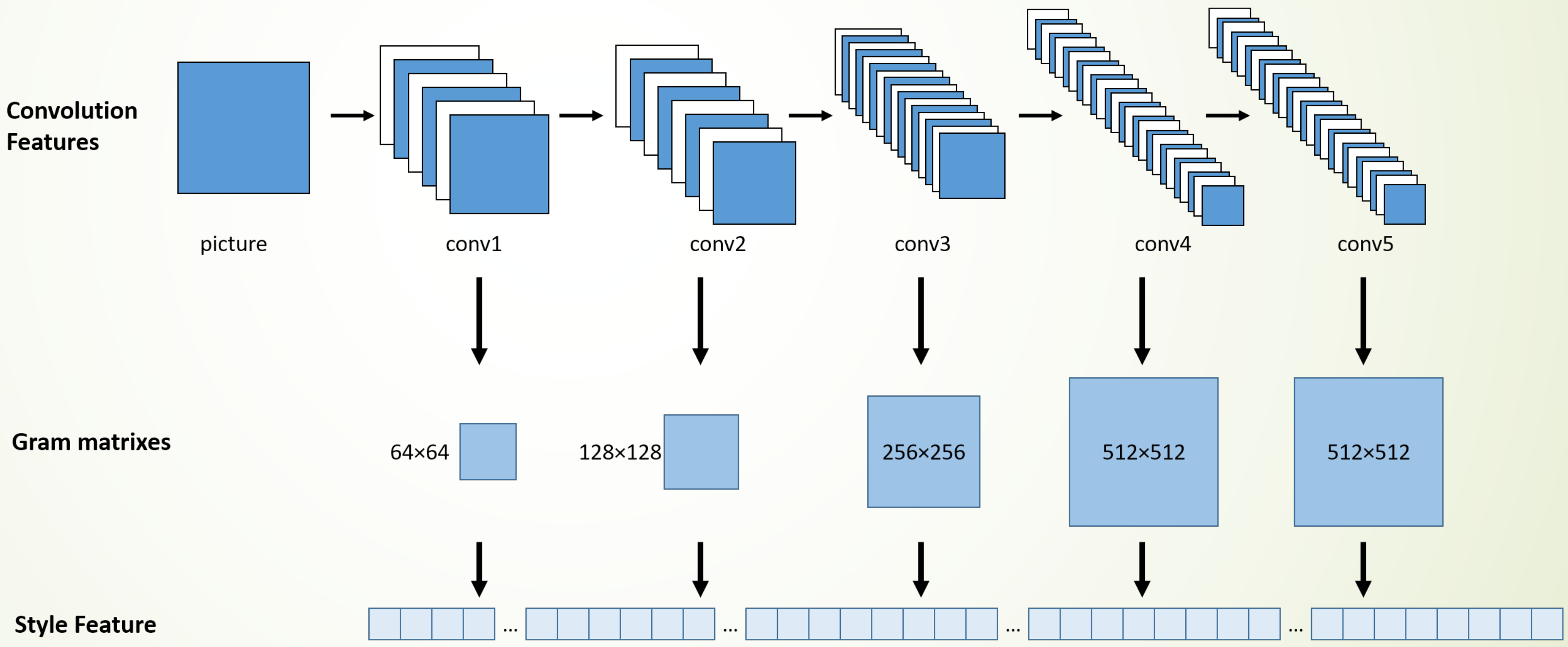
Neural Style

- J C Johnson's Website: https://github.com/jcjohnson/neural-style
- A torch implementation of the paper
  - *A Neural Algorithm of Artistic Style*,
  - by Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge.
  - http://arxiv.org/abs/1508.06576

Style-Content Feature Extraction

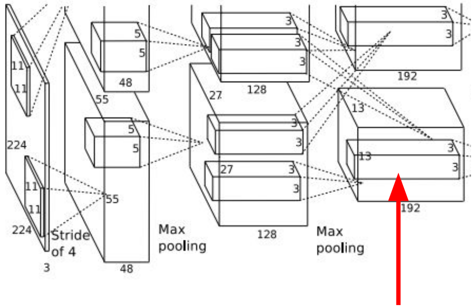


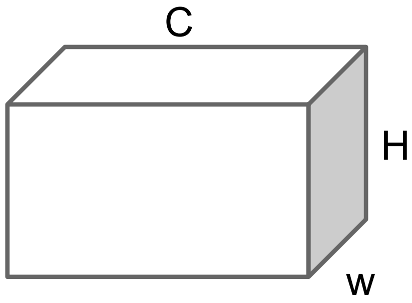
Style Features as Second Order Statistics

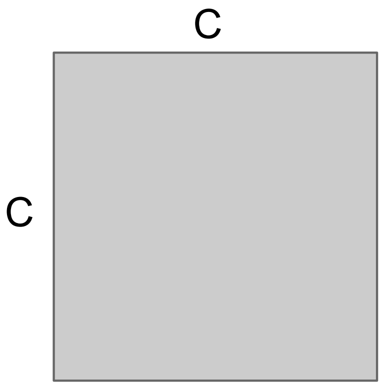


Gram Matrix as Style Features

This image is in the public domain.







Each layer of CNN gives C x H x W tensor of features; H x W grid of C-dimensional vectors

Outer product of two C-dimensional vectors gives C x C matrix measuring co-occurrence

Average over all HW pairs of vectors, giving **Gram matrix** of shape C x C

Efficient to compute; reshape features from

C x H x W to =C x HW

then compute $G = FF^T$

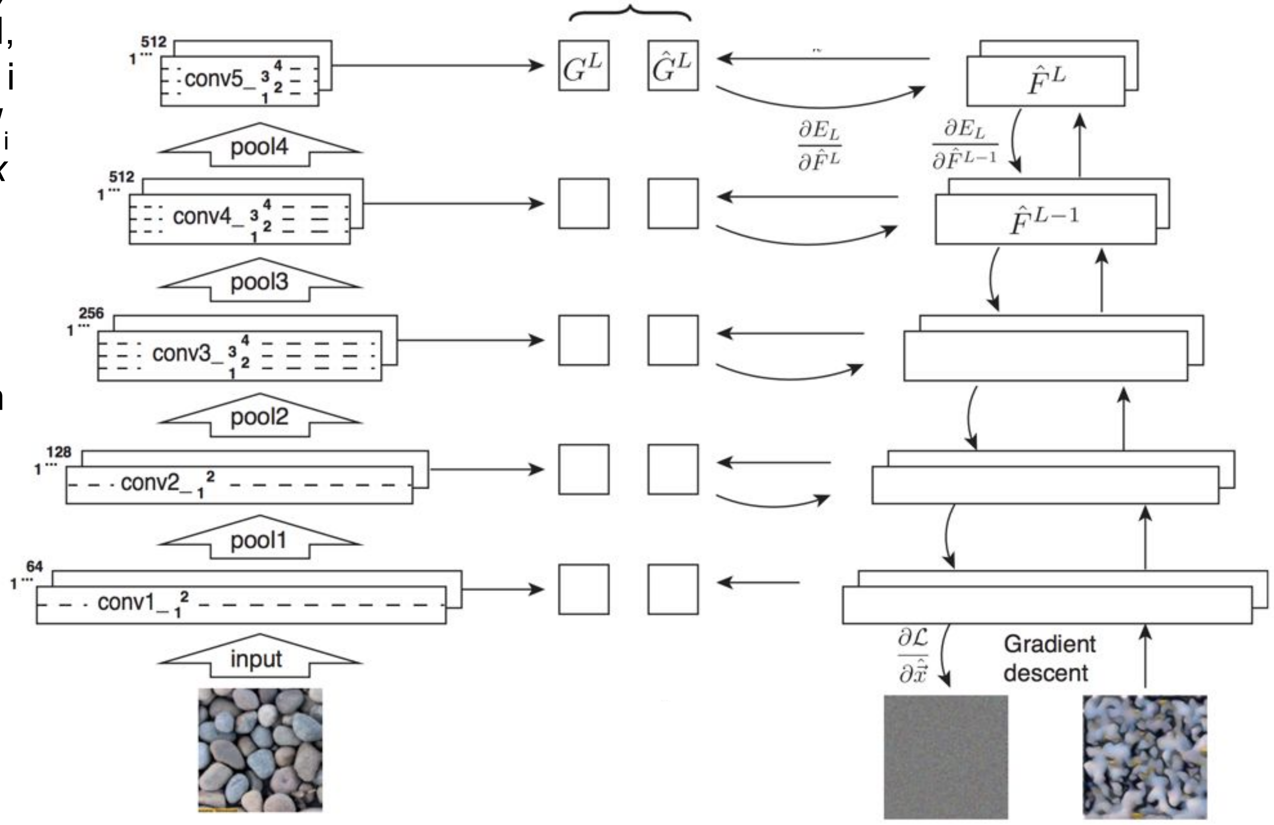
Neural Texture Synthesis

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - \hat{G}_{ij}^l\right)^2 \qquad \mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=0}^{L} w_l E_l$$

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer i gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the *Gram matrix* giving outer product of features:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{ (shape } C_i \times C_i\text{)}$$

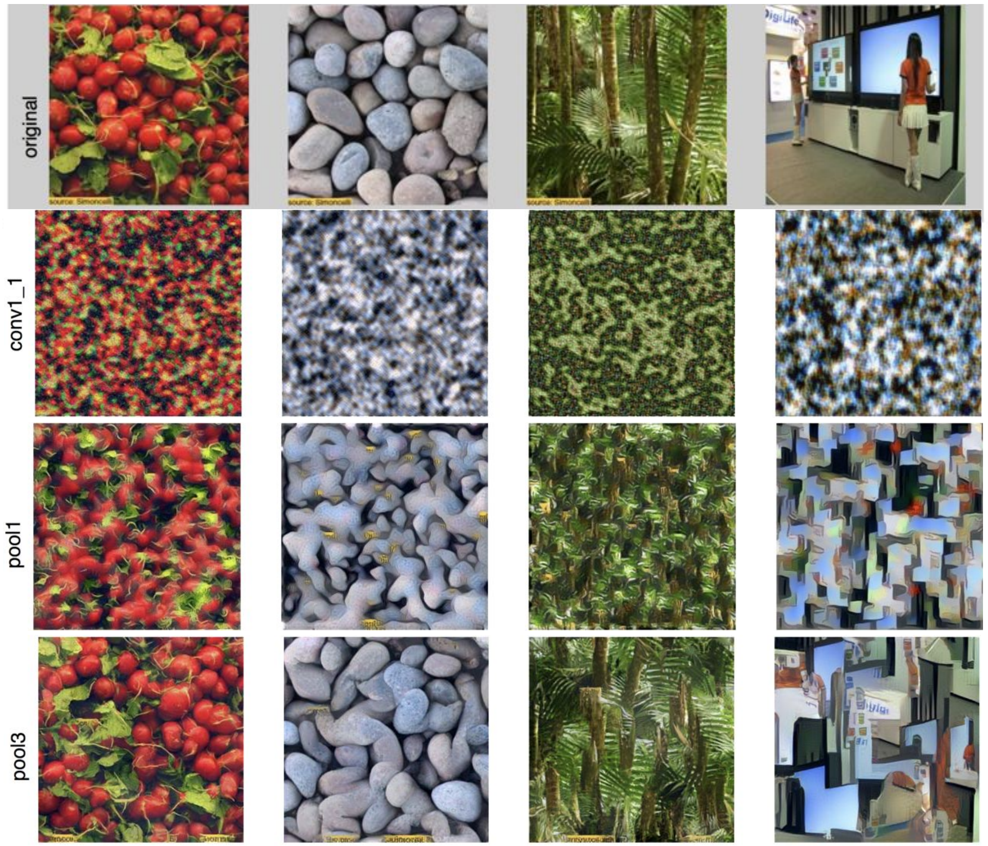
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5



Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015

Neural Texture Synthesis

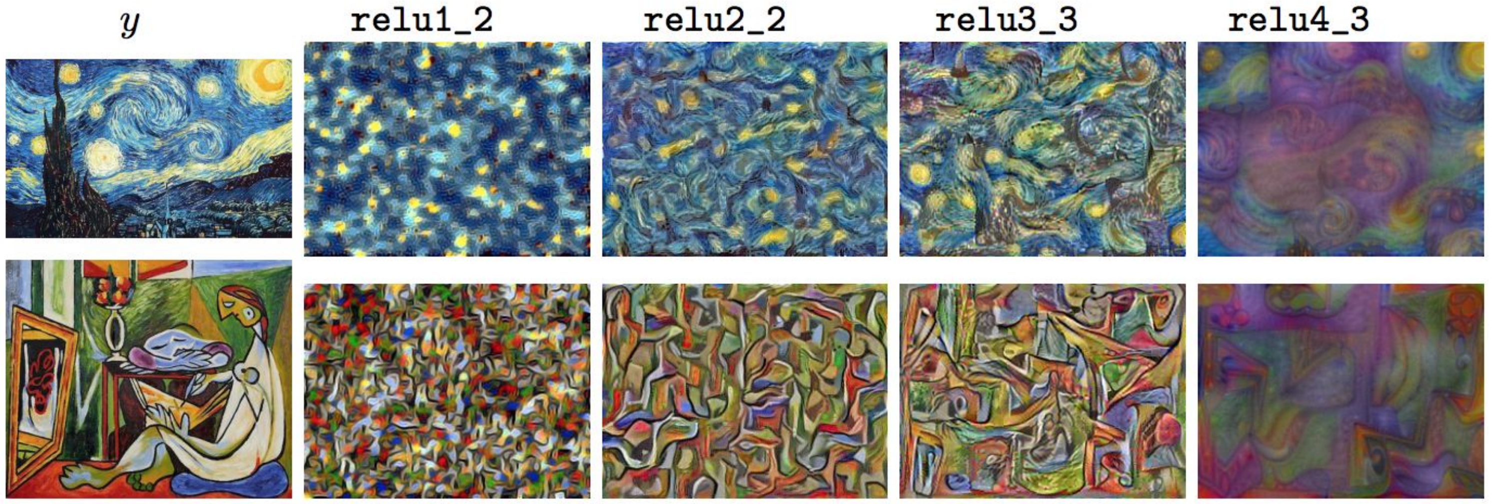
Reconstructing texture from higher layers recovers larger features from the input texture



Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Neural Networks”, NIPS 2015

Neural Texture Synthesis: Gram Reconstruction

Texture synthesis (Gram reconstruction)



Feature Inversion

Given a CNN feature vector for an image, find a new image that:

- Matches the given feature vector
- "looks natural" (image prior regularization)

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}\in\mathbb{R}^{H\times W\times C}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda\mathcal{R}(\mathbf{x})$$

Given feature vector

Features of new image

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

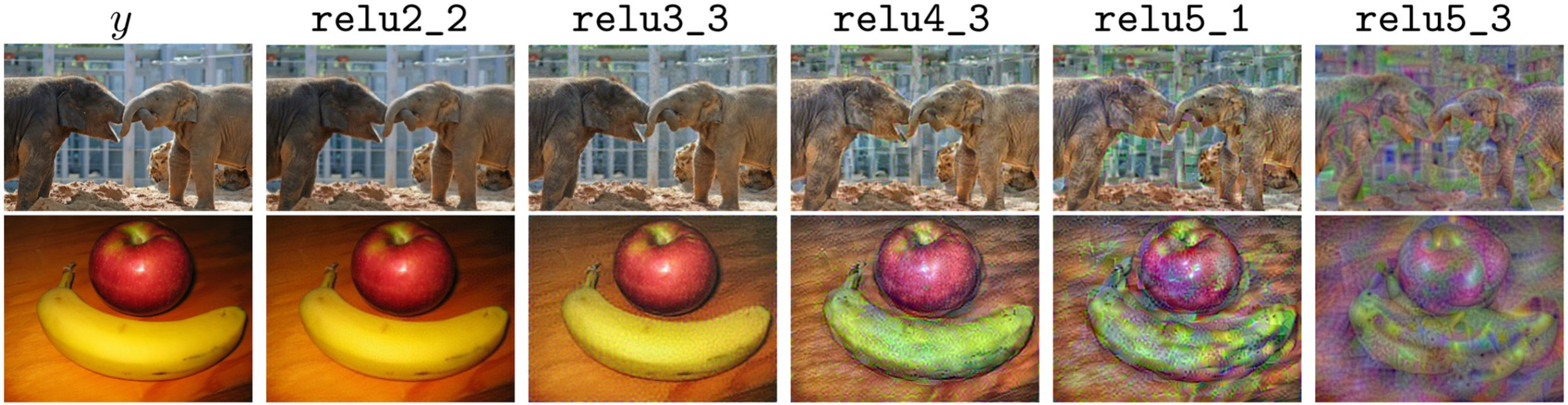
$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j}\left(\left(x_{i,j+1} - x_{ij}\right)^2 + \left(x_{i+1,j} - x_{ij}\right)^2\right)^{\frac{\beta}{2}}$$

Total Variation regularizer (encourages spatial smoothness)

Mahendran and Vedaldi, "Understanding Deep Image Representations by Inverting Them", CVPR 2015

Feature Inversion

Reconstructing from different layers of VGG-16



Mahendran and Vedaldi, “Understanding Deep Image Representations by Inverting Them”, CVPR 2015
Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016. Copyright Springer, 2016.
Reproduced for educational purposes.

Combined Loss for both Content (1st order statistics) and Style (2nd order statistics: Gram)

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^l - P_{ij}^l\right)^2 .$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l$$

where

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left(G_{ij}^l - A_{ij}^l\right)^2 \qquad G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l .$$

Neural Style Transfer

Content Image

+

Style Image

=

Style Transfer!

CNN learns texture features, not shapes!

(a) Texture image

81.4%	**Indian elephant**
10.3%	indri
8.2%	black swan

(b) Content image

71.1%	**tabby cat**
17.3%	grey fox
3.3%	Siamese cat

(c) Texture-shape cue conflict

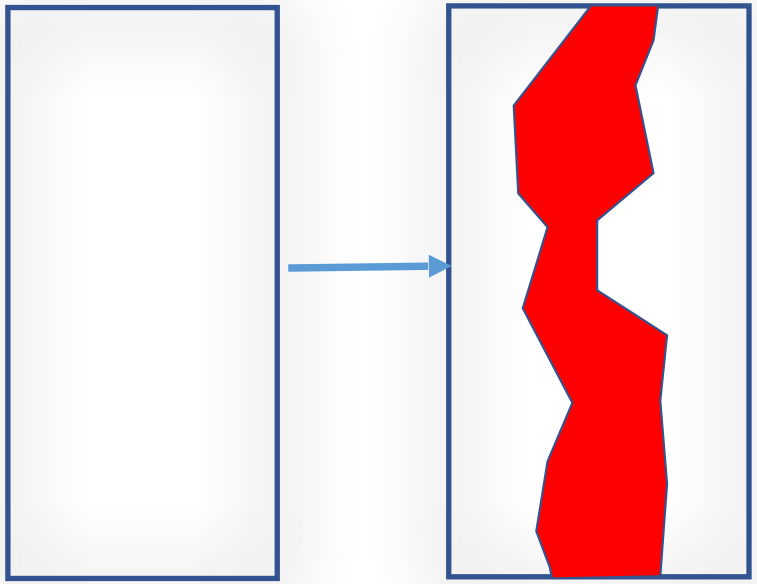
63.9%	**Indian elephant**
26.4%	indri
9.6%	black swan

Geirhos et al. ICLR 2019

https://videoken.com/embed/W2HvLBMhCJQ?tocitem=46

1:16:47

Lottery Ticket Hypothesis for Efficient Subnets in Deep Learning



Over-parameterized Networks W

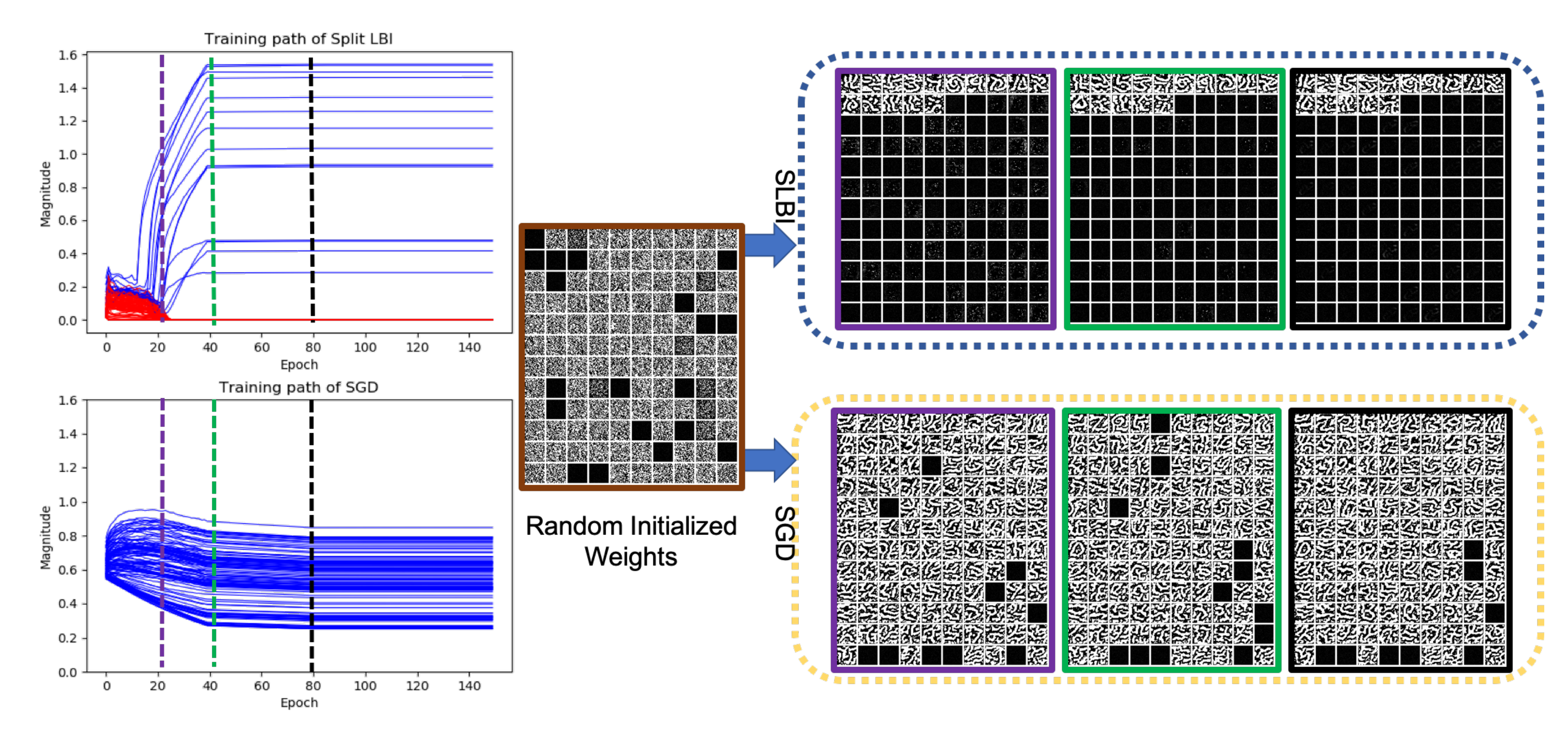
Compressive Networks W_s

Lottery Ticket Hypothesis

- *Dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that – when trained in isolation – reach test accuracy comparable to the original network in a similar number of iterations.* (Frankle & Carbin, 2019)

Rewinding the network from the initialization, and find "winning ticket" subnet

Split LBI finds efficient sparse architecture



Yanwei Fu et al. TPAMI 45(2):1749-1765, 2023.
Yanwei Fu et al. DessiLBI, ICML 2020.

Texture bias in ImageNet training

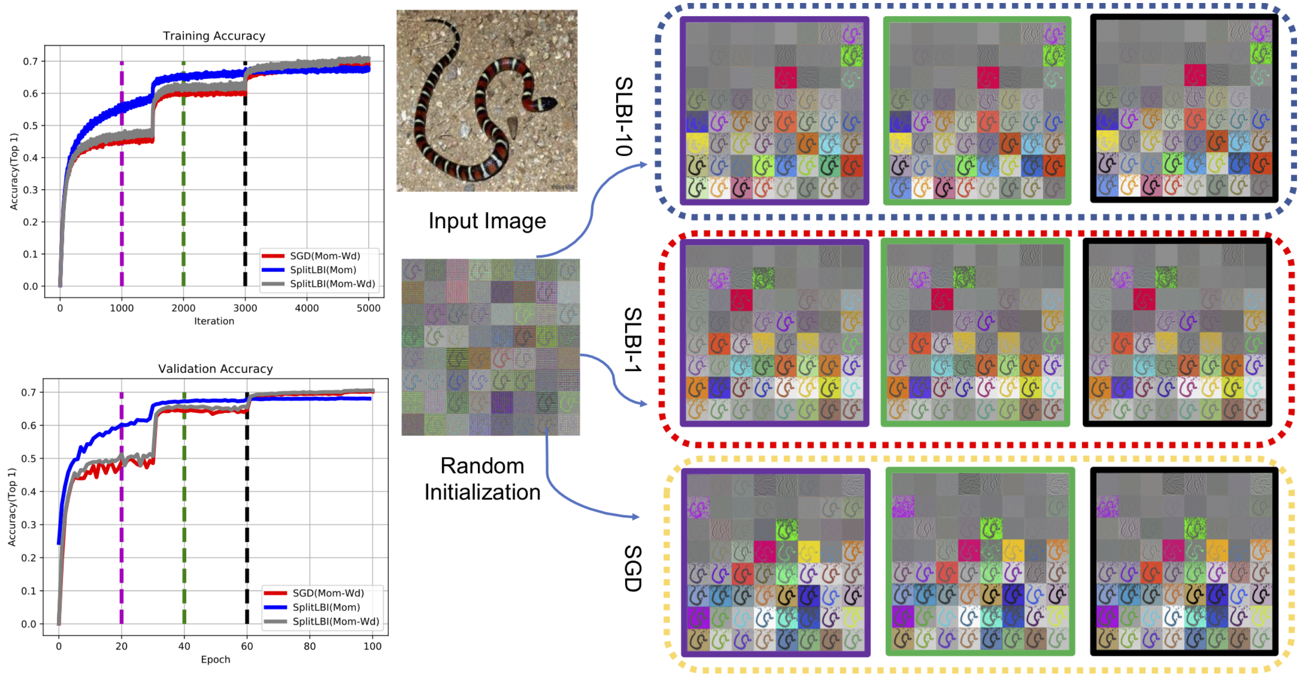
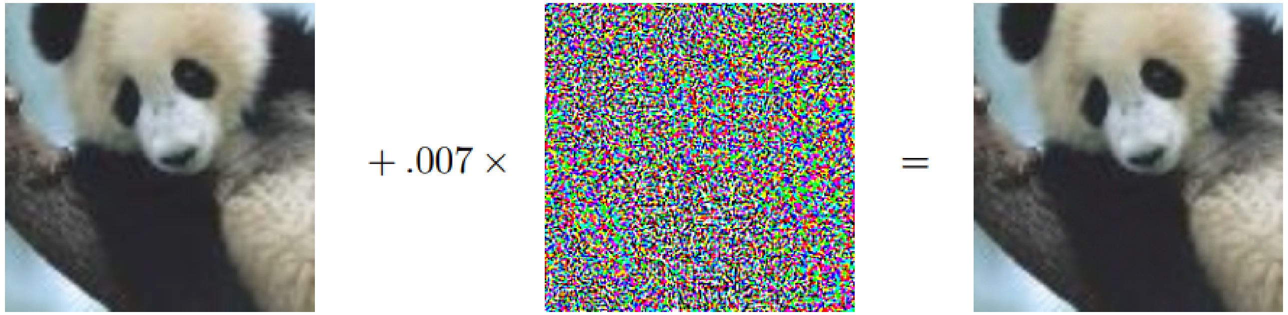


Figure: Visualization of the first convolutional layer filters of ResNet-18 trained on ImageNet-2012, where texture features are more important than colour/shapes. Given the input image and initial weights visualized in the middle, filter response gradients at 20 (purple), 40 (green), and 60 (black) epochs are visualized. SGD with Momentum (Mom) and Weight Decay (WD), is compared with SLBI.

Yanwei Fu et al. TPAMI 45(2):1749-1765, 2023.
Yanwei Fu et al. DessiLBI, ICML 2020.

Adversarial Examples and Robustness

Deep Learning may be fragile: adversarial examples



$+ .007 \times$ $=$

$\boldsymbol{x}$ — "panda" 57.7% confidence

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ — "nematode" 8.2% confidence

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$ — "gibbon" 99.3 % confidence

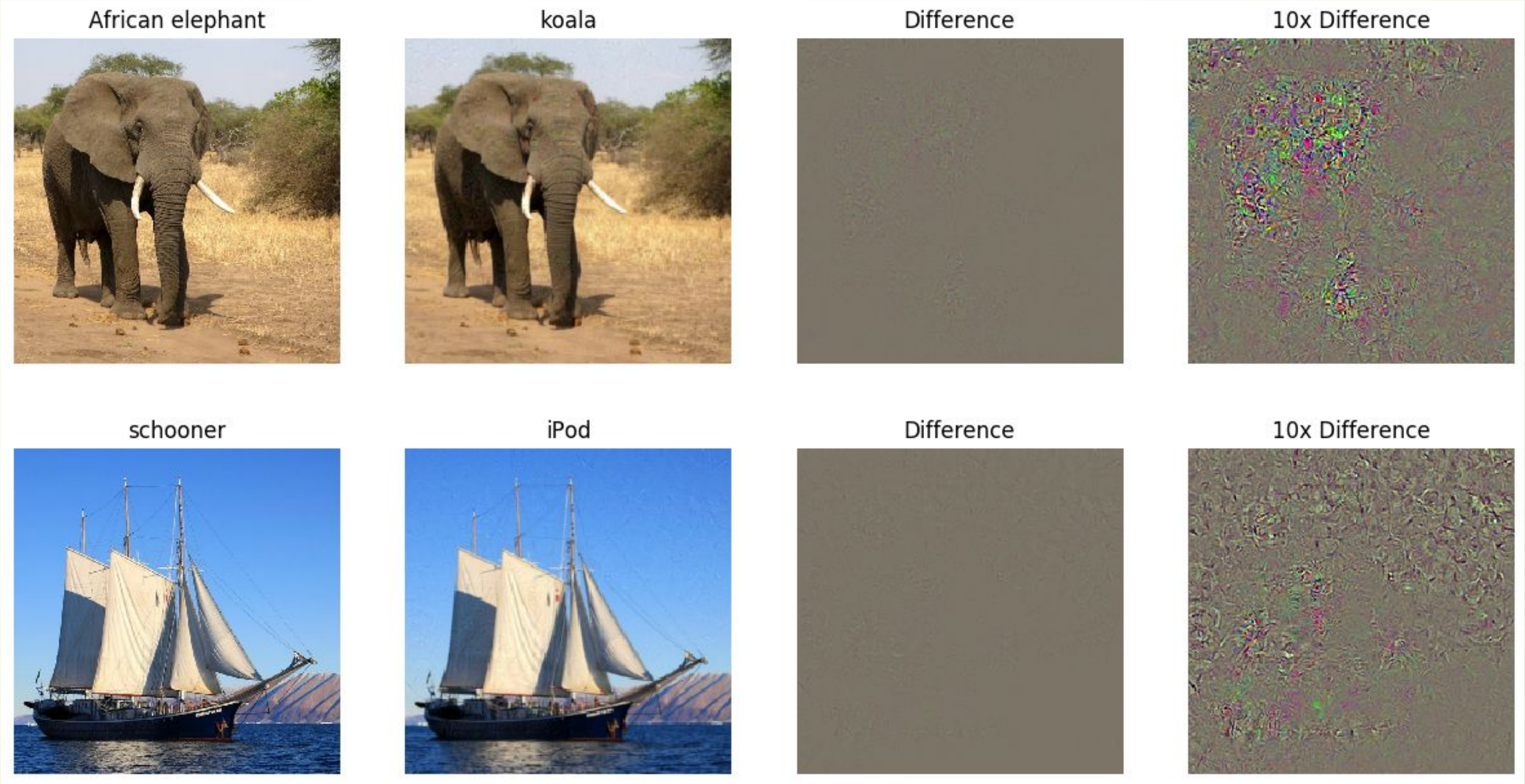
[Goodfellow et al., 2014]

- Small but malicious perturbations can result in severe misclassification
- Malicious examples generalize across different architectures
- What is source of instability?
- Can we robustify network?

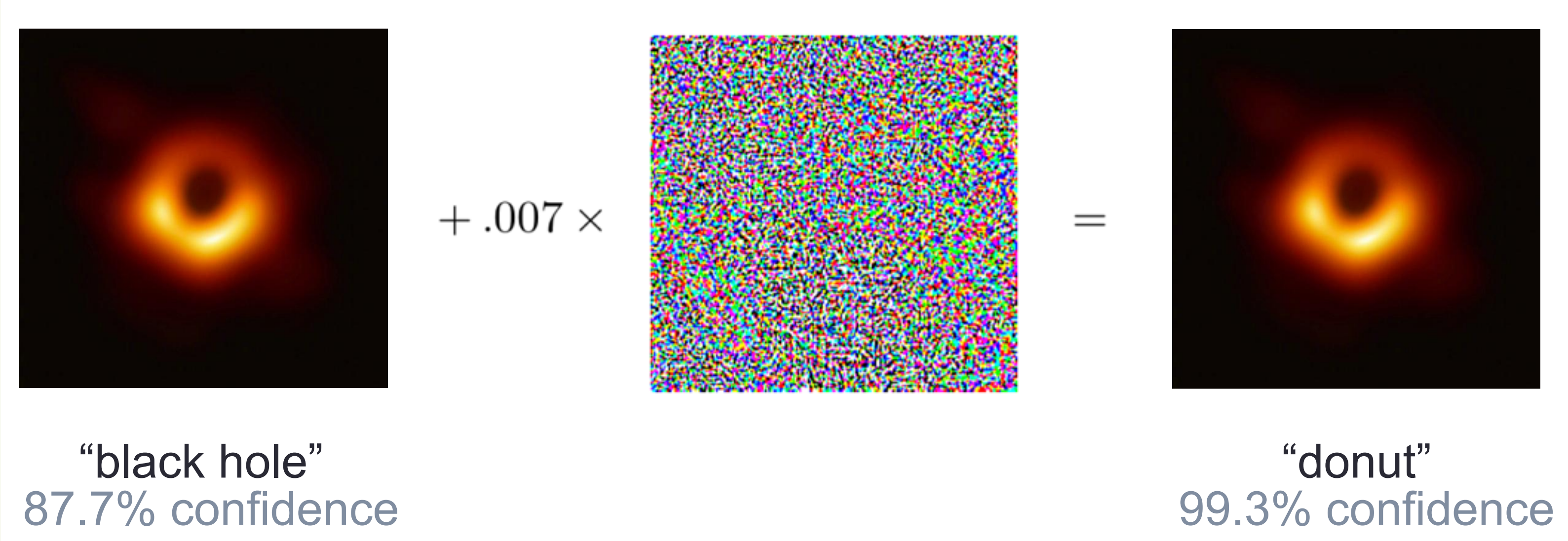
Adversarial Examples: Fooling Images

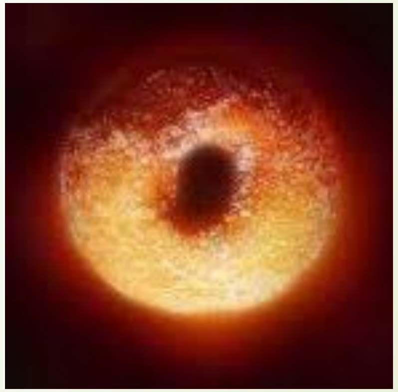
- Start from an arbitrary image
- Pick an arbitrary class
- Modify the image to maximize the class
- Repeat until network is fooled

Fooling Images/Adversarial Examples



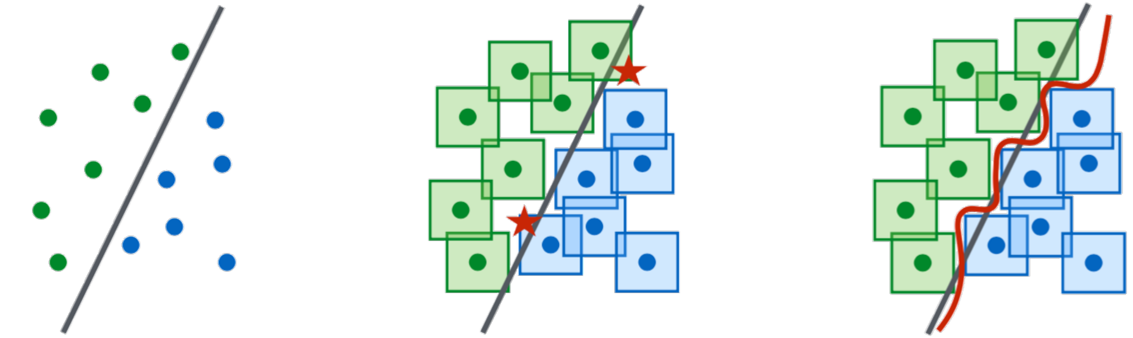
Convolutional Networks lack Robustness





Courtesy of Dr. Hongyang ZHANG.

Adversarial Robust Training



- Traditional training:

$$\min_{\theta} J_n(\theta, \mathbf{z} = (x_i, y_i)_{i=1}^{n})$$

  - e.g. square or cross-entropy loss as negative log-likelihood of logit models

- Robust optimization (Madry et al. ICLR'2018):

$$\min_{\theta} \max_{\|\epsilon_i\| \le \delta} J_n(\theta, \mathbf{z} = (x_i + \epsilon_i, y_i)_{i=1}^{n})$$

  - robust to any distributions, yet computationally hard

Extended by **Hongyang ZHANG** et al. by TRADES, 2019.

Thank you!