Convolutional Neural Networks

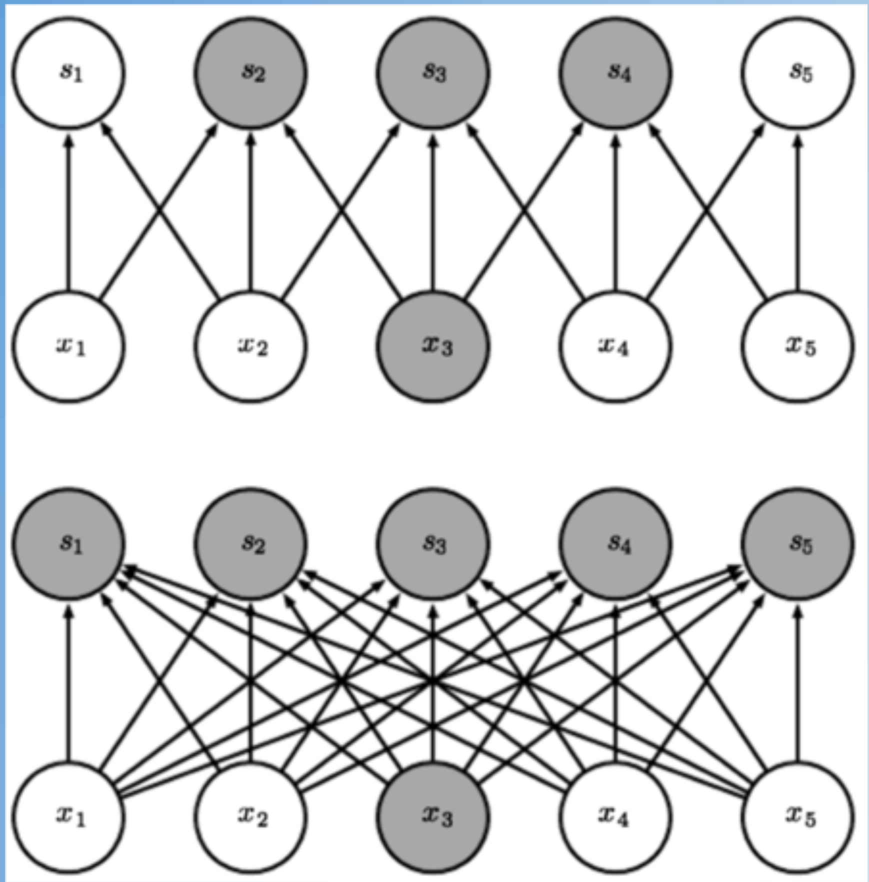
Zhanxing Zhu (朱占星)
zhanxing.zhu@pku.edu.cn
Peking University (北京大学)

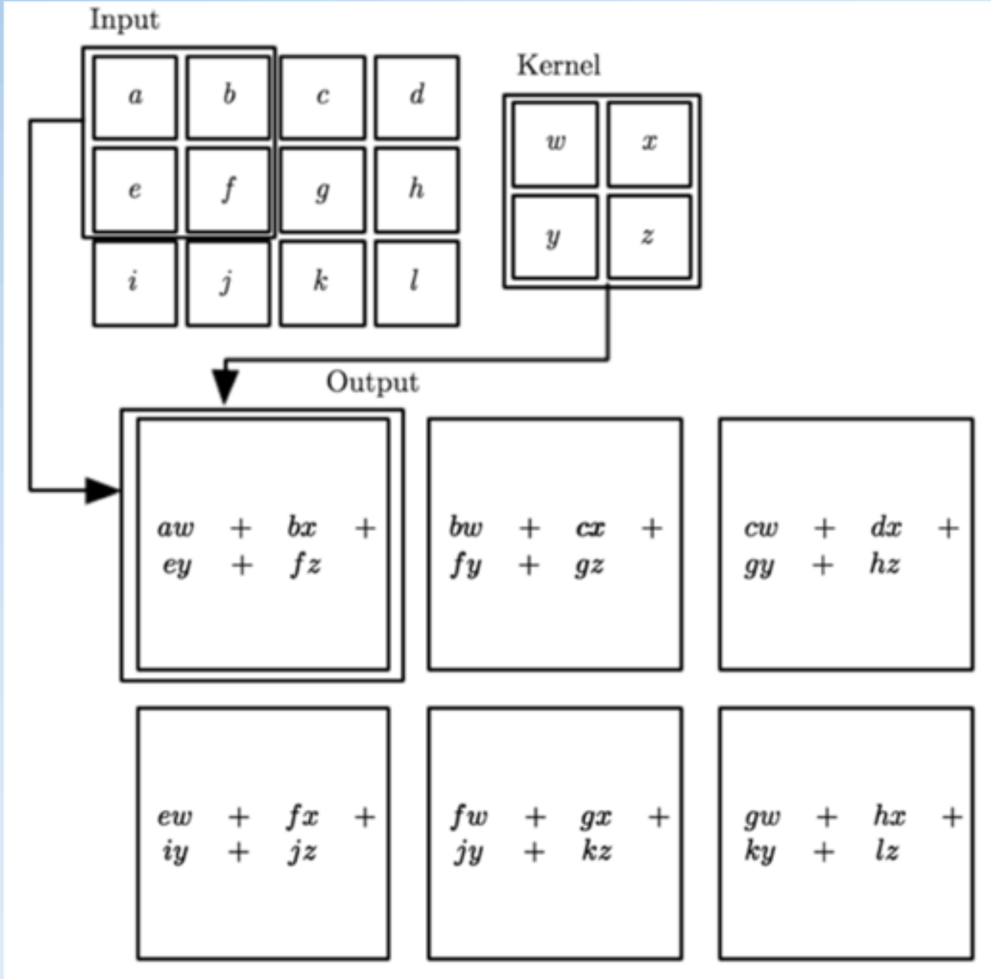
CNN (Lecun, 1989)

- Suitable for data with grid topology
  - Especially important in computer vision
    Object recognition, image classification, 2D grid
  - Time series, 1D grid
- Fully-connected not practical for images
  - Huge number of pixels: parameter explosion
  - Image size: 1000*1000, O(1,000,000) first layer
- CNN: convolution
  - Sparse connections; parameter sharing: equivariant to translation

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

Convolution

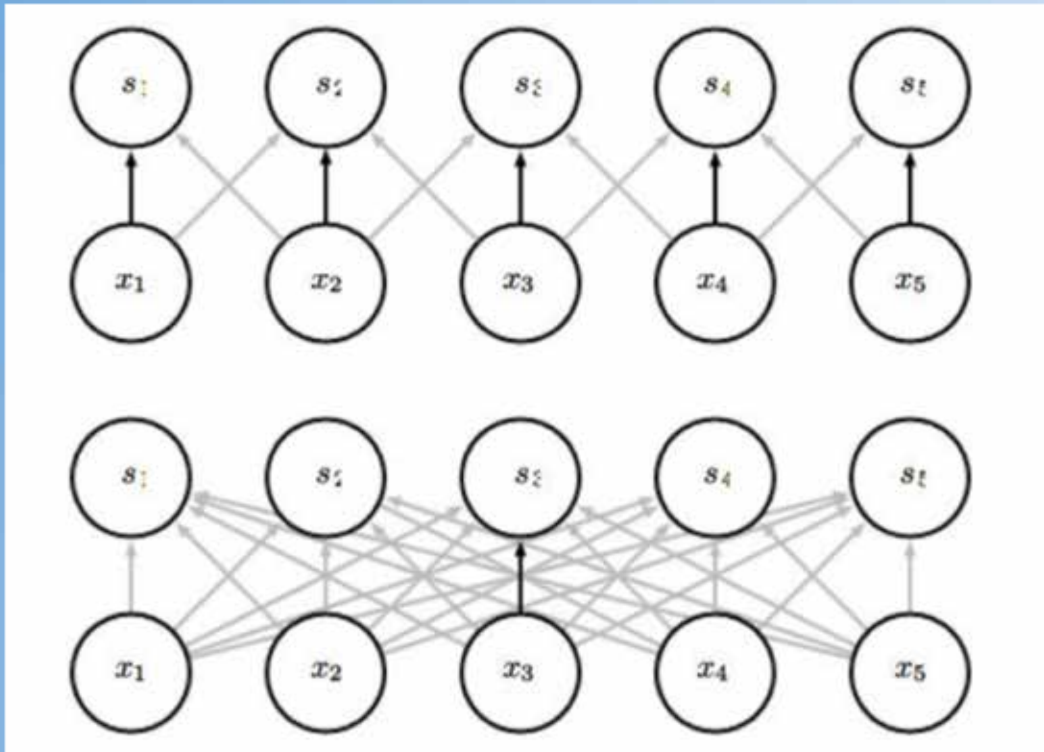




Cross-correlation

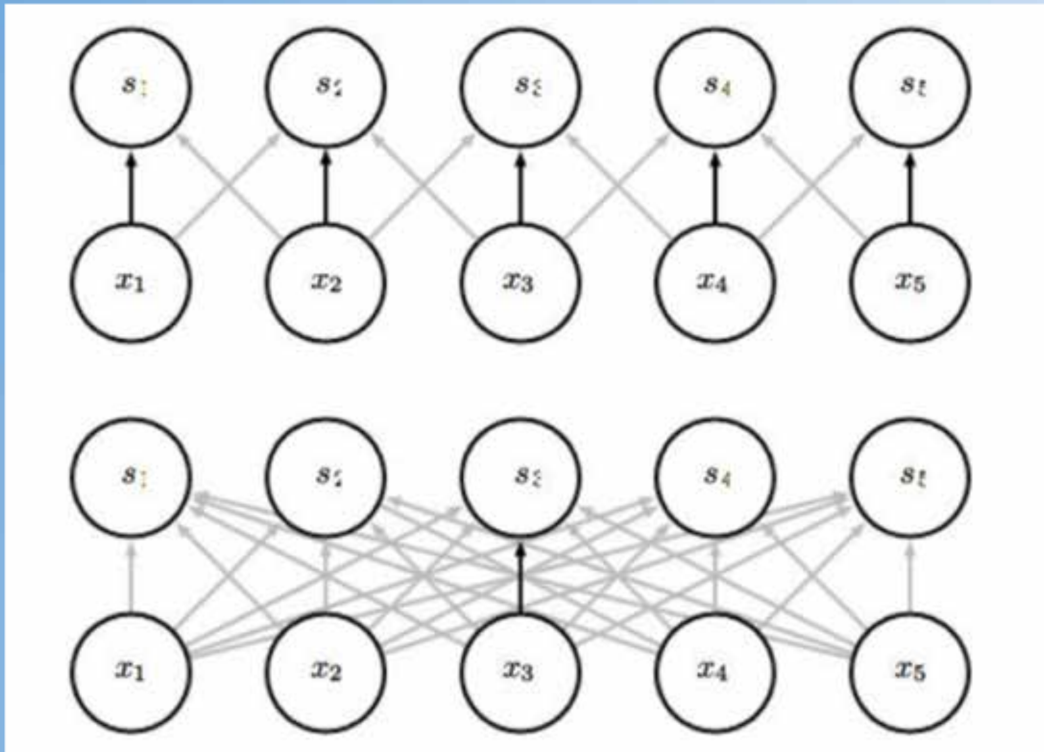
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

Motivation 1: weight sharing



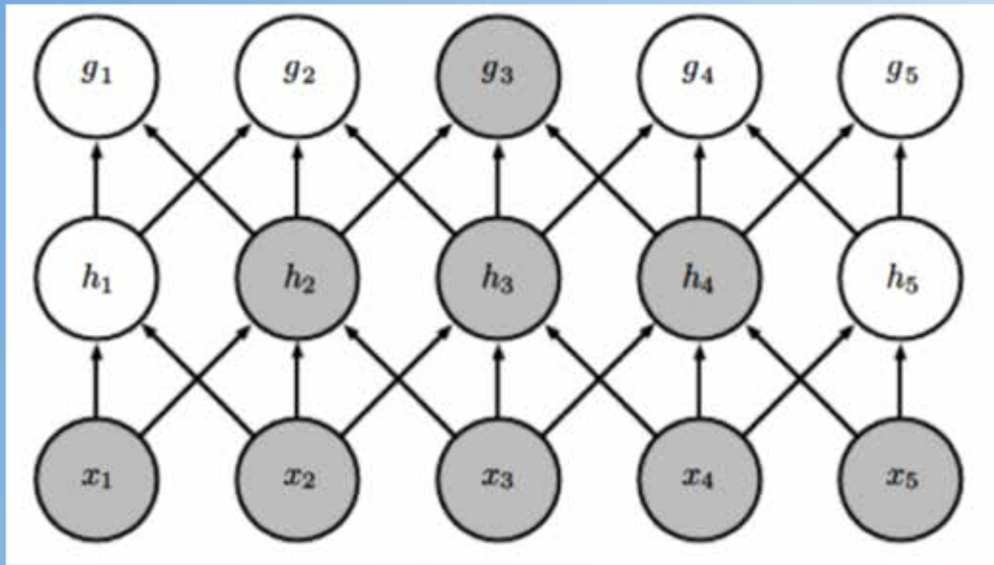
Every hidden units in the first layer detects exactly the same feature.

Motivation 2: sparse connection



Fewer parameters

Motivation 2: sparse connection

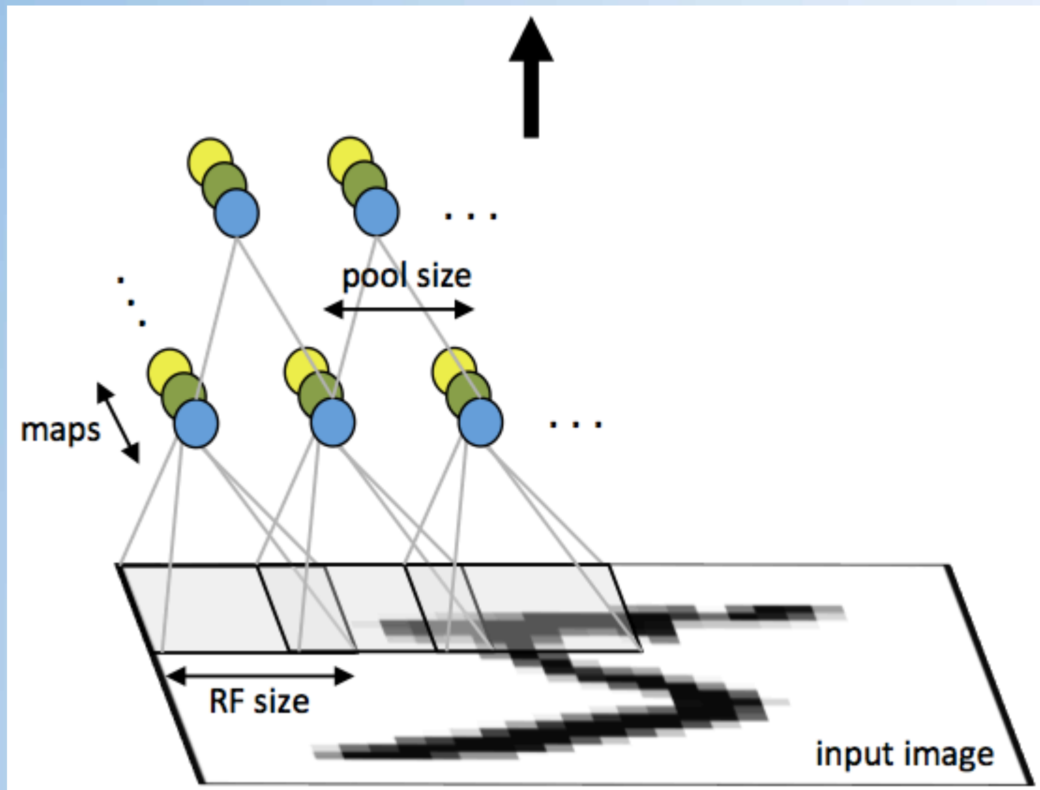


Local connection

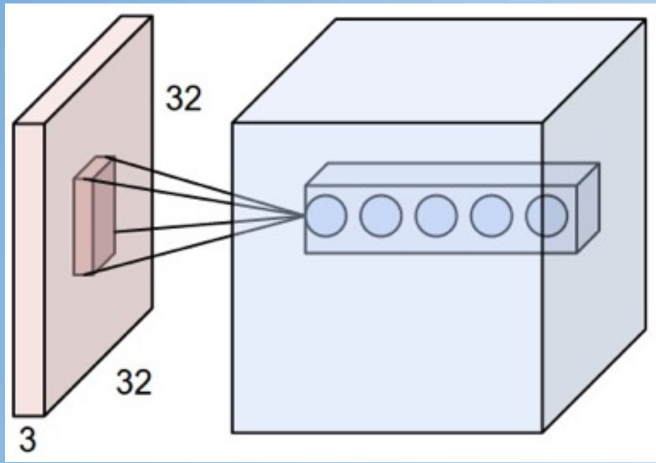
Composition to produce large receptive field.

Motivation 3: equivariant to translation

Allows features to be detected regardless of position.



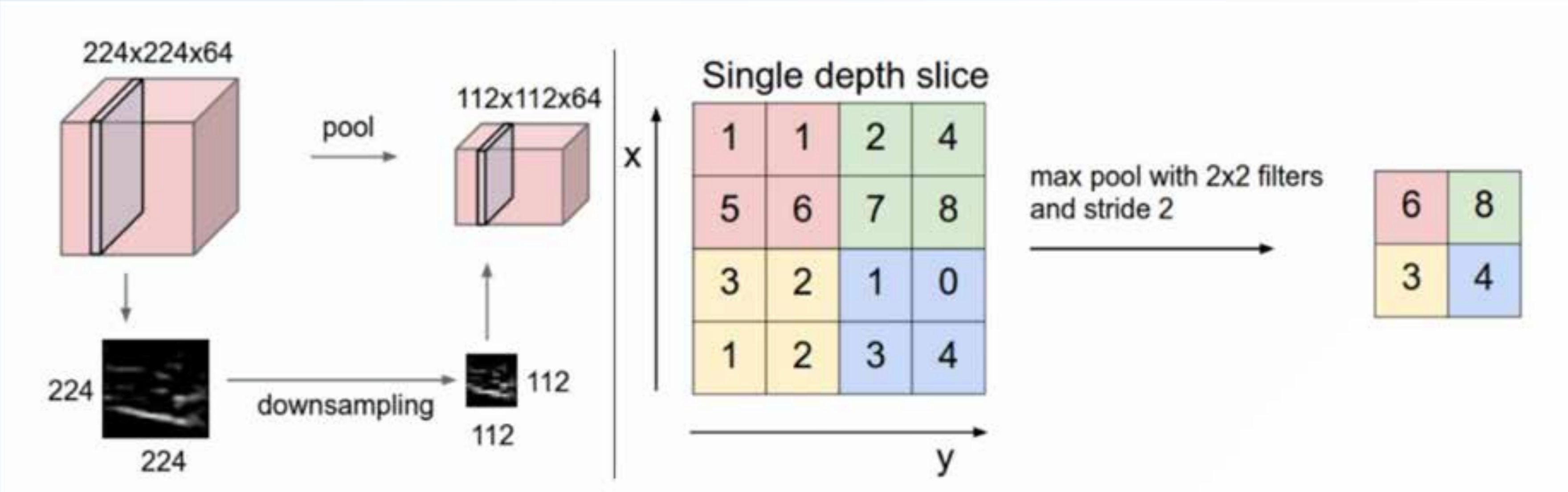
Multi-channel



Input feature map: $X \in \mathbb{R}^{M \times N \times C}$.
Convolutional kernel: $K \in \mathbb{R}^{S \times r \times r \times C}$.
s-th output feature map:

$$F_s(i,j) = \sum_{c=1}^{C} \sum_{m} \sum_{n} X(m,n,c) K_s(i-m, j-n, c).$$

Pooling

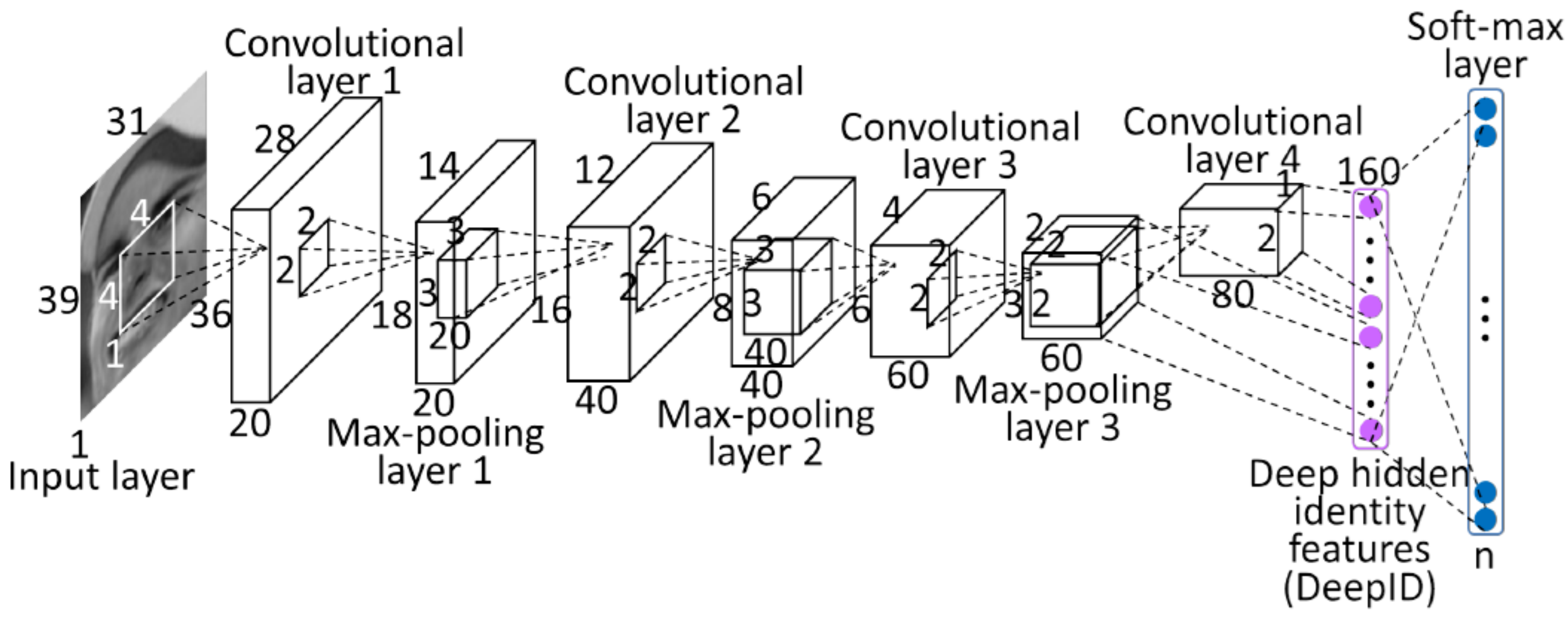


Variants

1. Max pooling. $y = \max\{x_1, \cdots, x_n\}$.
2. Average pooling. $y = \text{mean}\{x_1, \cdots, x_n\}$

CNN as a whole

Stack conv layer + pooling



Training

Again, back-propagation and SGD.

Case study: LeNet-5

Proposed in "Gradient-based learning applied to document recognition", by Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner, in Proceedings of the IEEE, 1998

- Apply convolution on 2D images (MNIST) and use back-propagation.
- Structure: 2 convolutional layers (with pooling) + 3 fully connected layers.
  - Input size: $32 \times 32 \times 1$
  - Convolutional kernel size: 5×5
  - Pooling: 2×2

Case study: LeNet-5

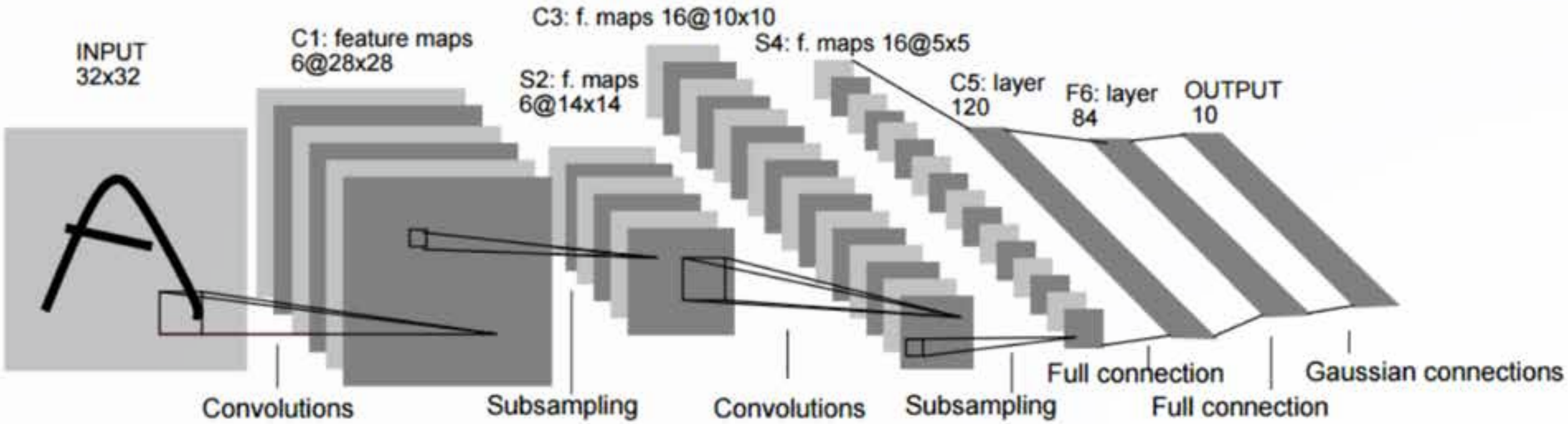


Figure from *Gradient-based learning applied to document recognition, by Y. LeCun, L. Bottou, Y. Bengio and P. Haffner*

Case study: AlexNet

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

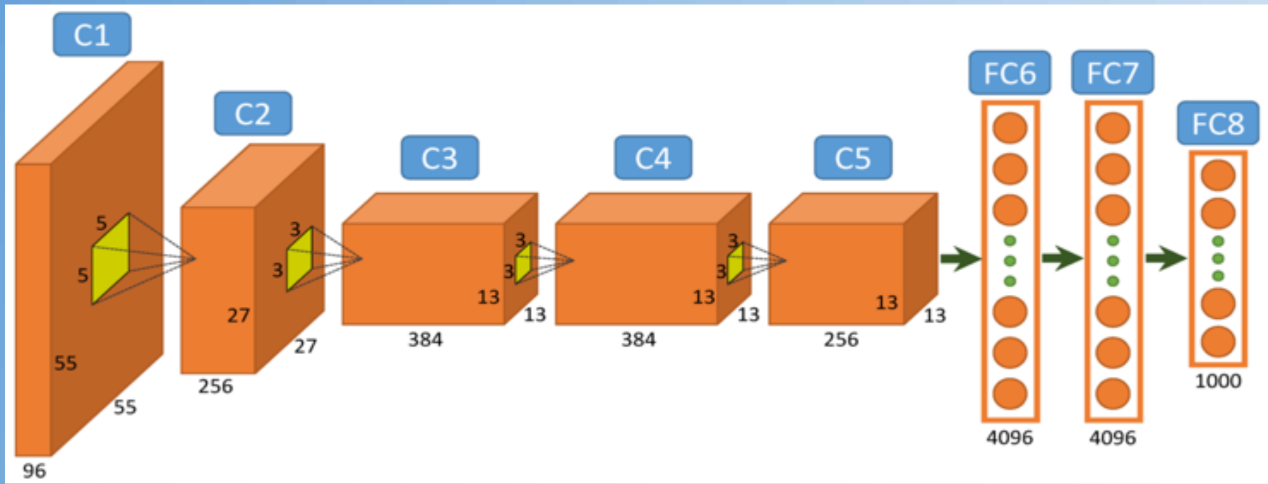
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

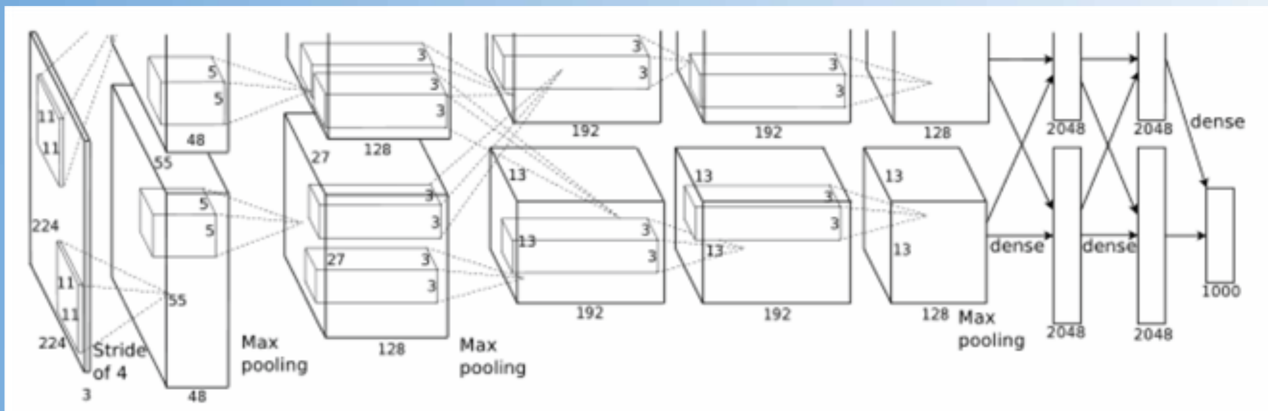
Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists

Krizhevsky et.al. “ImageNet classification with deep convolutional neural networks.” NIPS 2012.

Case study: AlexNet





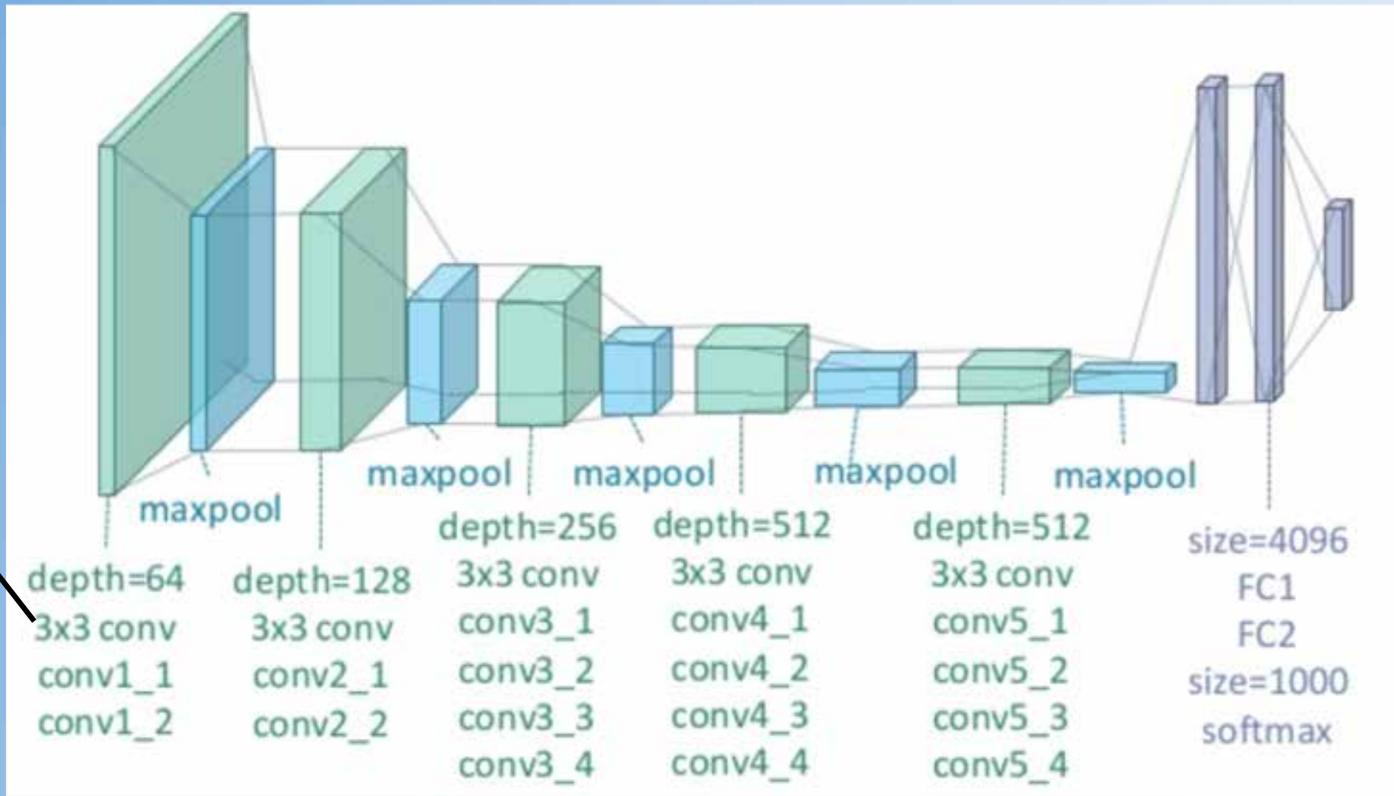
Alex et.al. "Imagenet classification with deep convolutional neural networks."

1. First ever big CNN and really effective
2. Training with GPU
3. ReLU activation
4. Dropout
5. SGD with momentum
6. Data augmentation

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

$$w_{i+1} := w_i + v_{i+1}$$

Going Deeper

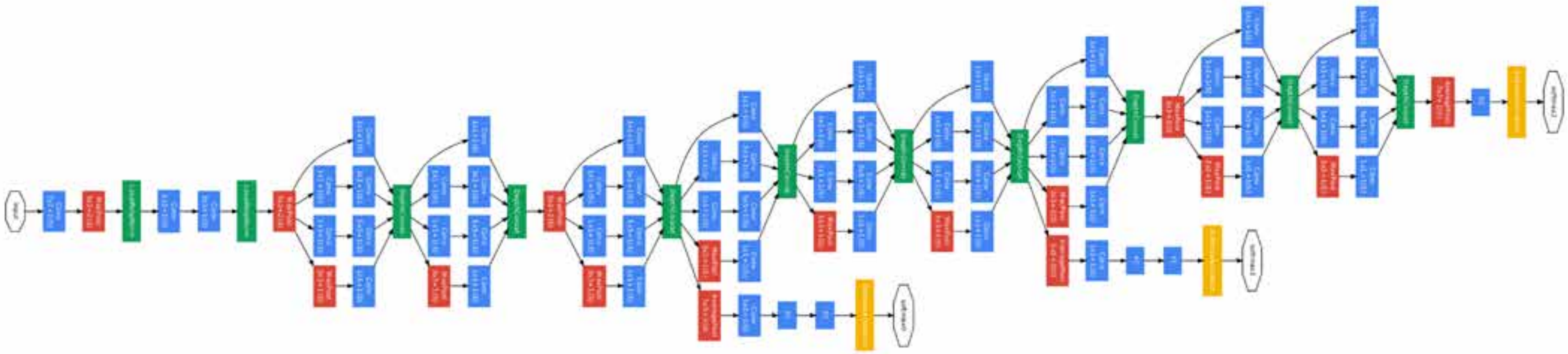


Small kernel size

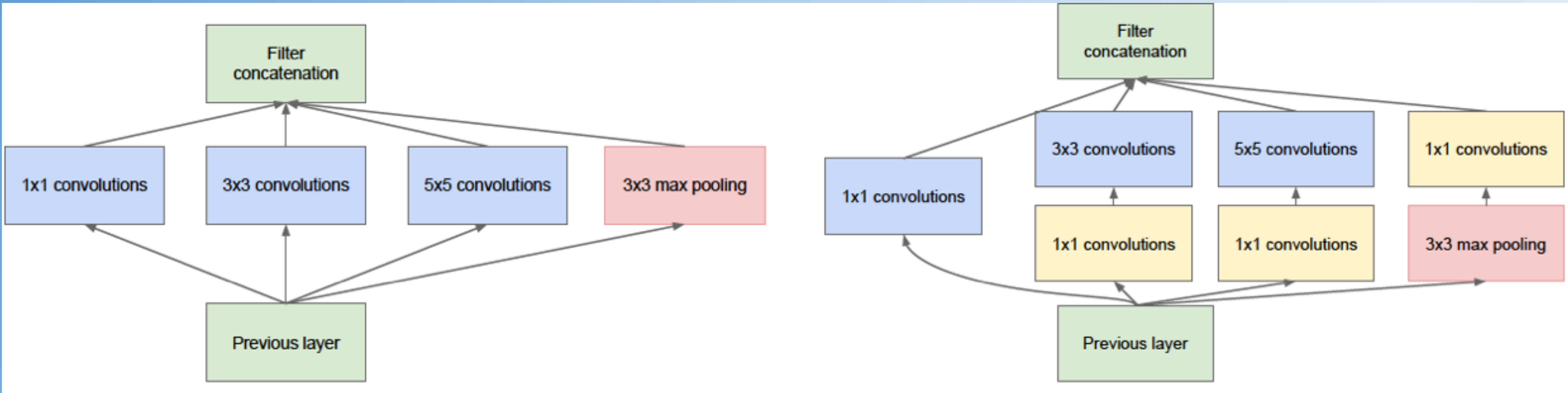
VGG-19

K. Simonyan, A. Zisserman. "Very deep convolutional networks for large-scale image recognition arXiv technical report, 2014

GoogleNet (Inception)



Objects with different size might need suitable kernel size.



(a) Inception module, naïve version (b) Inception module with dimension reductions

filters with **multiple sizes** operate on the **same level**

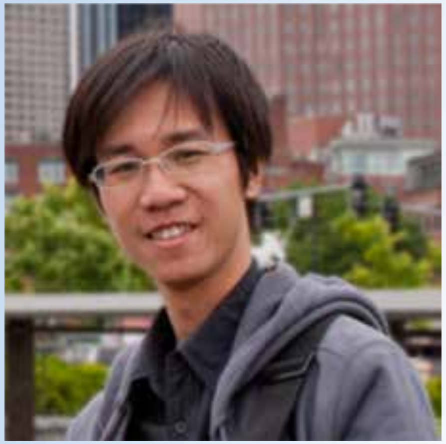
The total loss used by the inception net during training.
total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2

Problems of the previous CNNs

- Cannot go deeper, VGG at most 19 layers
- Why?
- Training is the bottleneck!
- Gradient vanishing makes the optimization extremely hard!
- Current architectures not suitable for very deep nets.

New milestone of CNN: ResNet

He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016. CVPR Best Paper



Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

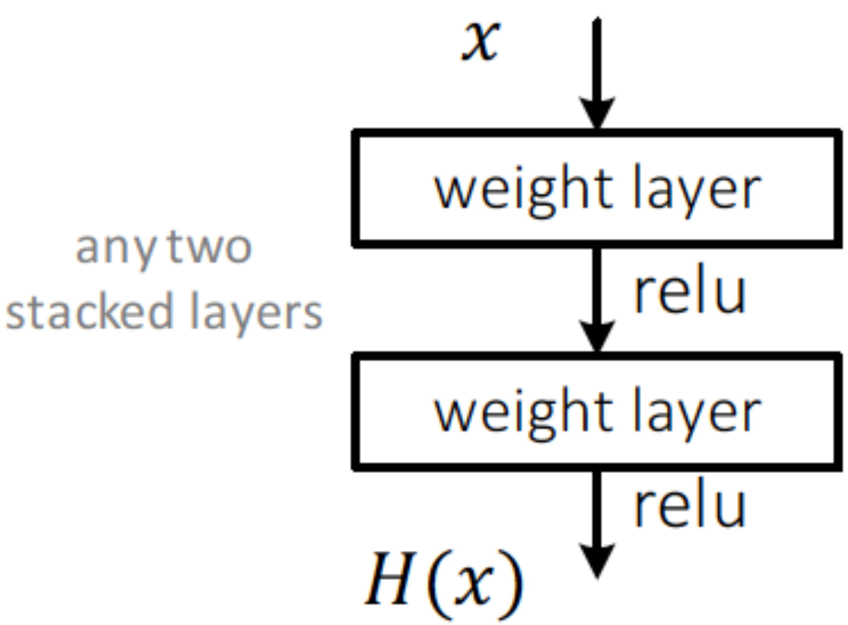
ResNet, 152 layers
(ILSVRC 2015)

Properties of ResNet

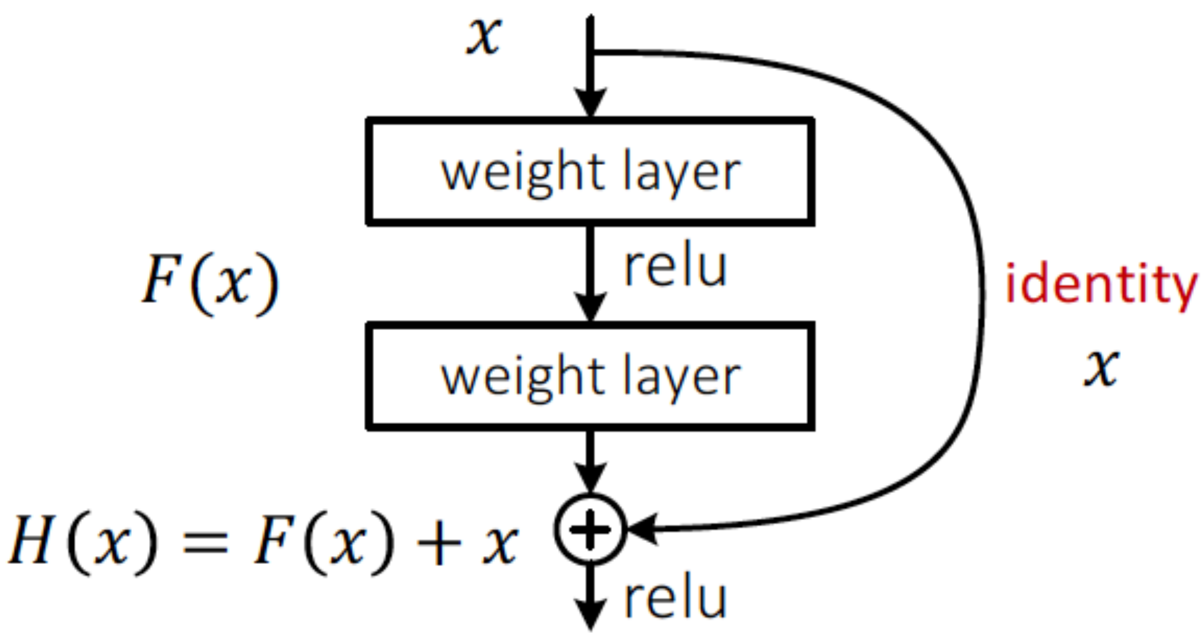
- Very deep
- Easy to train
- State-of-the-art performance on image recognition, object detection, semantic segmentation…
- A revolution on designing network structure

Shortcut connection makes difference.

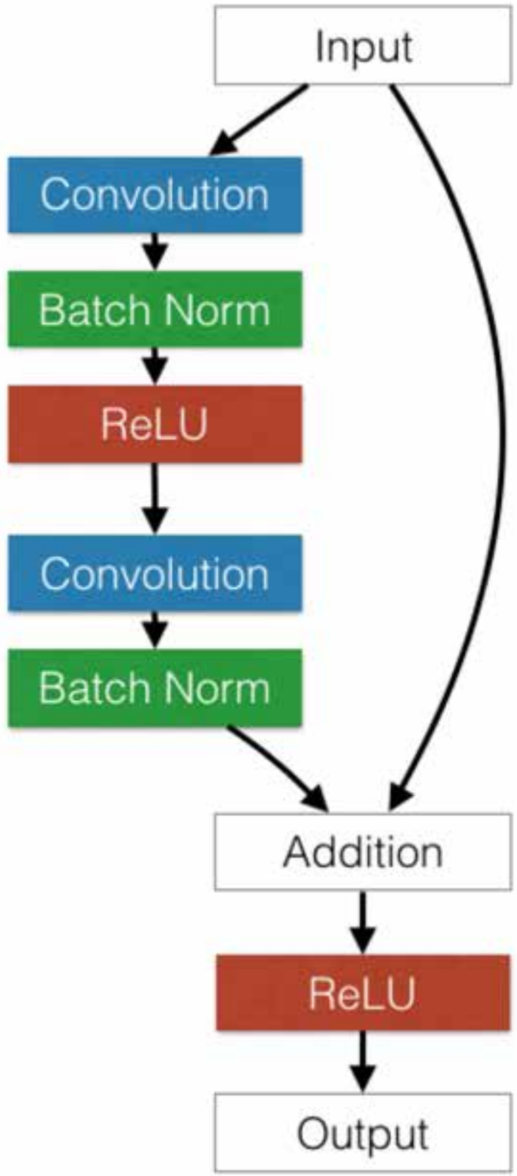
- Plaint net



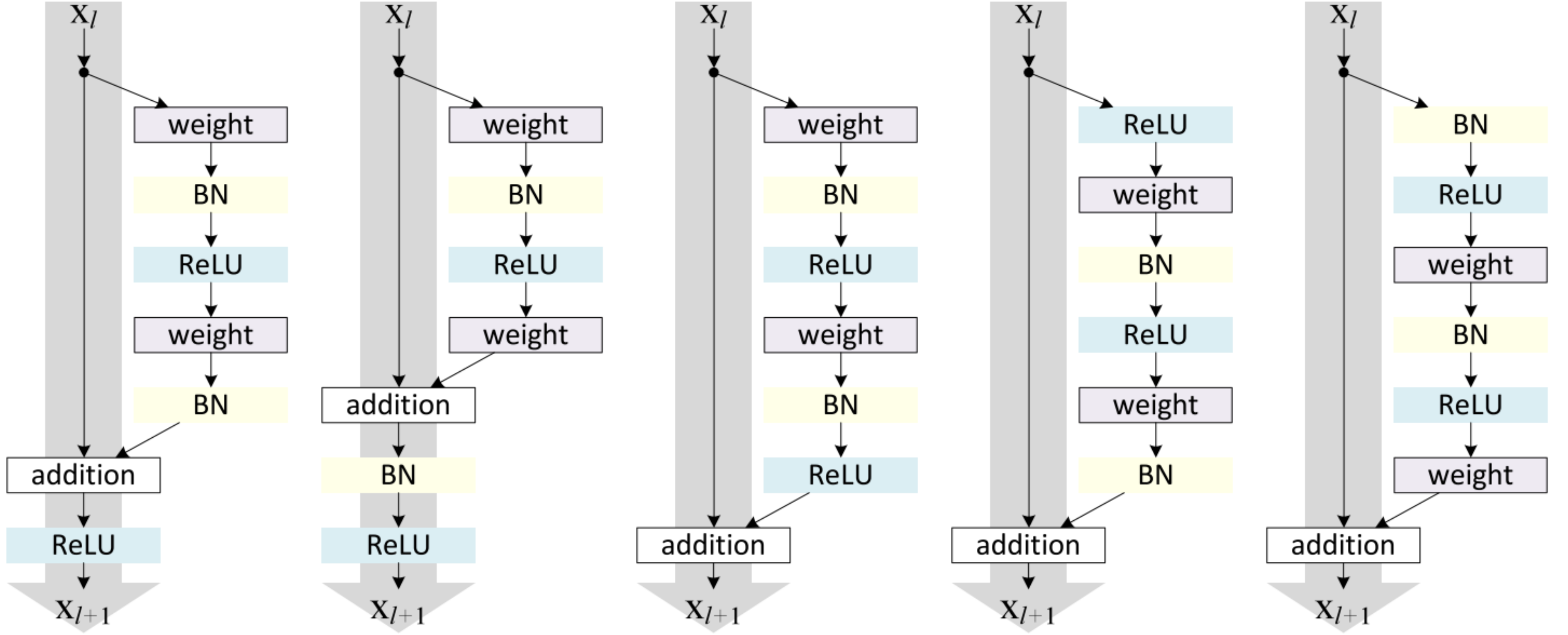
- Residual net



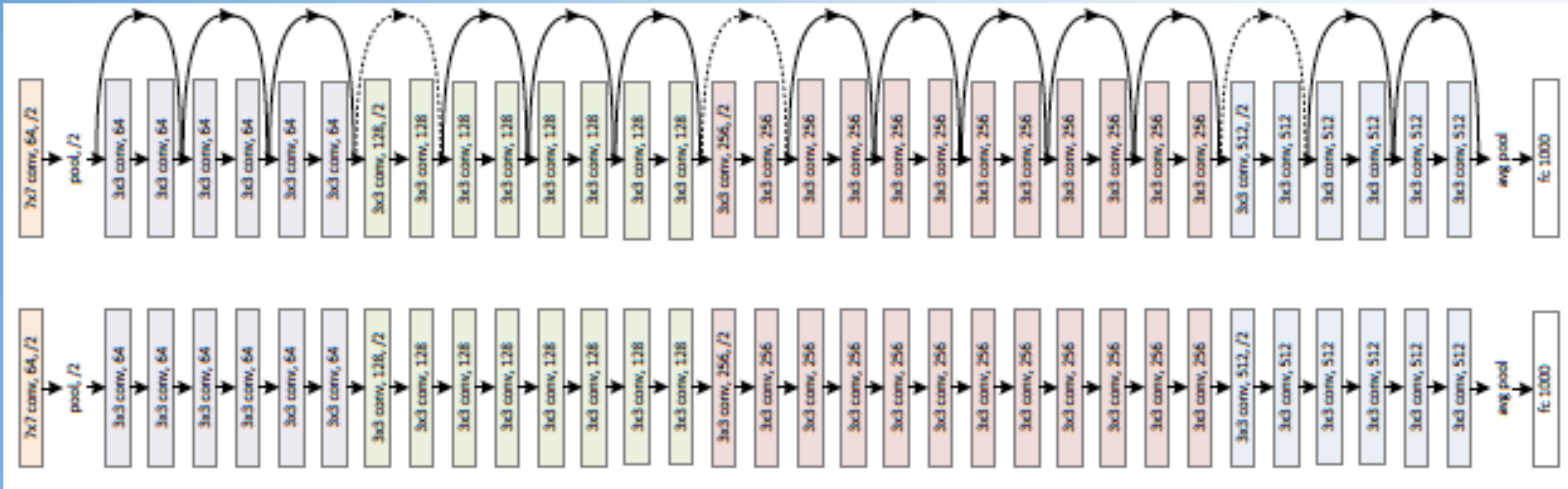
- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations

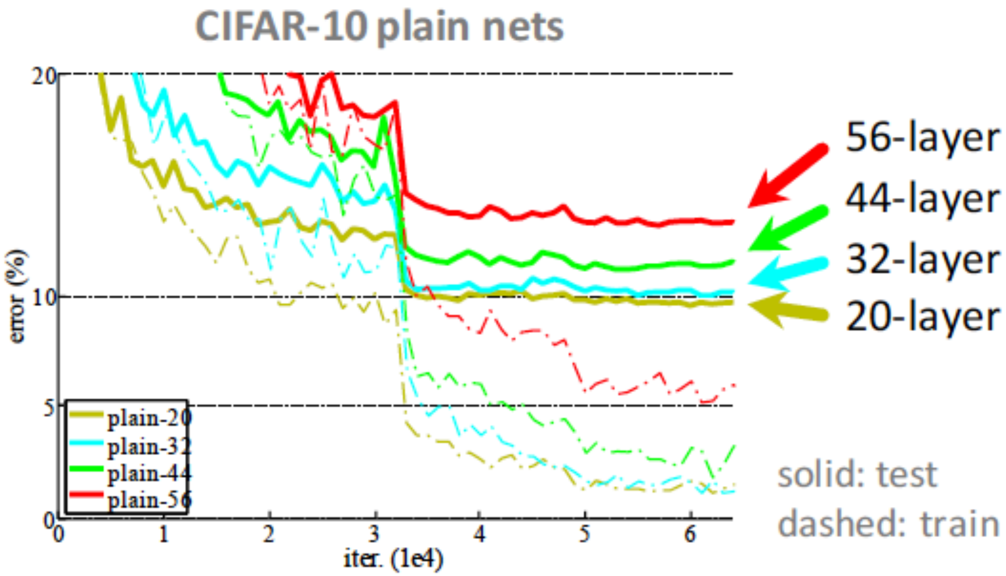

Input
Convolution
Batch Norm
ReLU
Convolution
Batch Norm
Addition
ReLU
Output

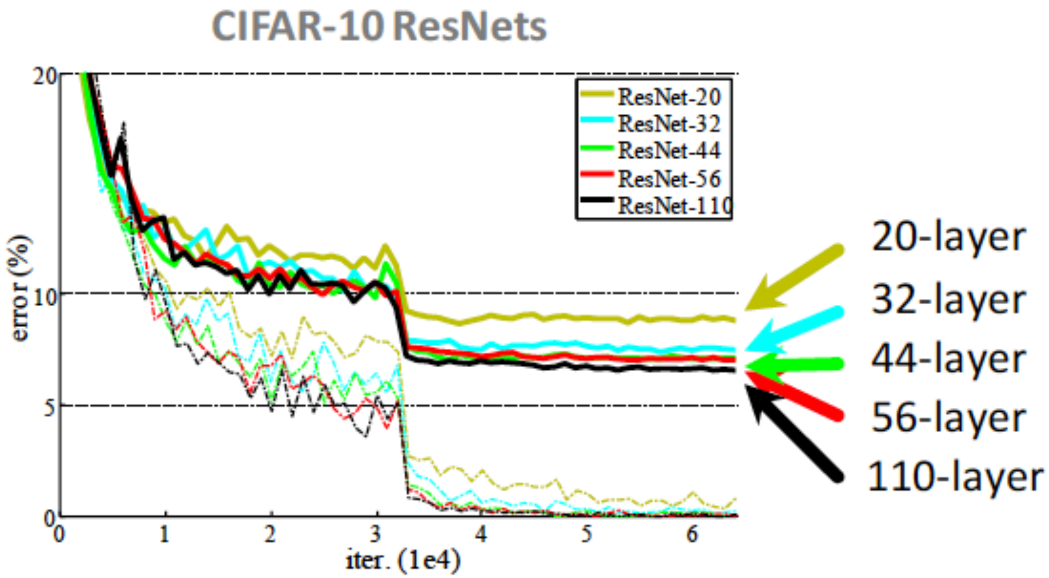
Variants of residual block



(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**


7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000


CIFAR-10 plain nets
20
10
5
0
0
1
2
3
4
5
6
error (%)
iter. (1e4)
plain-20
plain-32
plain-44
plain-56
56-layer
44-layer
32-layer
20-layer
solid: test
dashed: train


CIFAR-10 ResNets
20
10
5
0
0
1
2
3
4
5
6
error (%)
iter. (1e4)
ResNet-20
ResNet-32
ResNet-44
ResNet-56
ResNet-110
20-layer
32-layer
44-layer
56-layer
110-layer

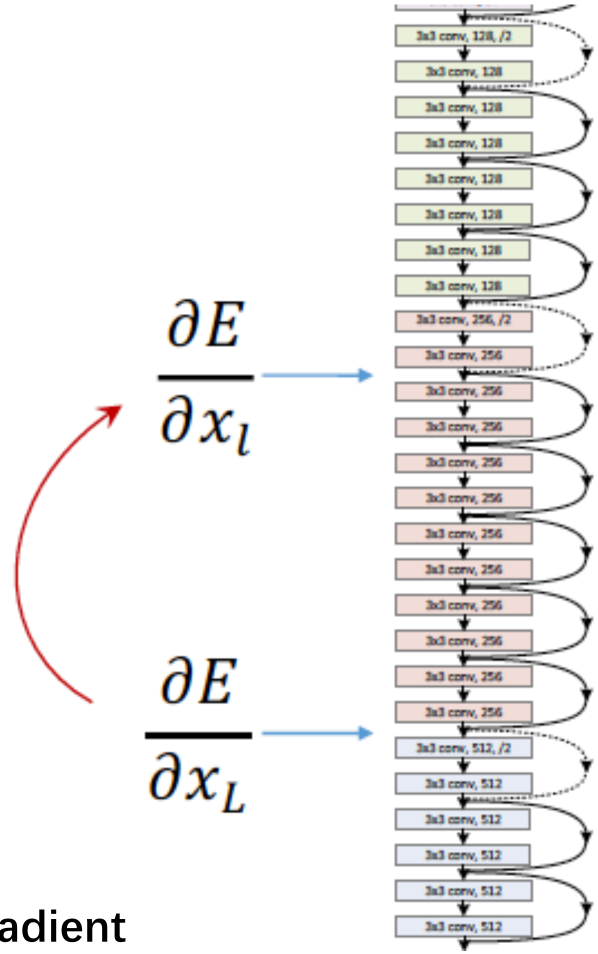
Very smooth backpropagation: no vanishing gradient

$$x_L = x_l + \sum_{i=l}^{L-1} F(x_i)$$

$$\frac{\partial E}{\partial x_l} = \frac{\partial E}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial E}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1} F(x_i)\right)$$

$$x_L = \prod_{i=l}^{L-1} W_i x_l$$

Multiplicative, easy to produce very small gradient



Visualize the loss landscape of neural networks

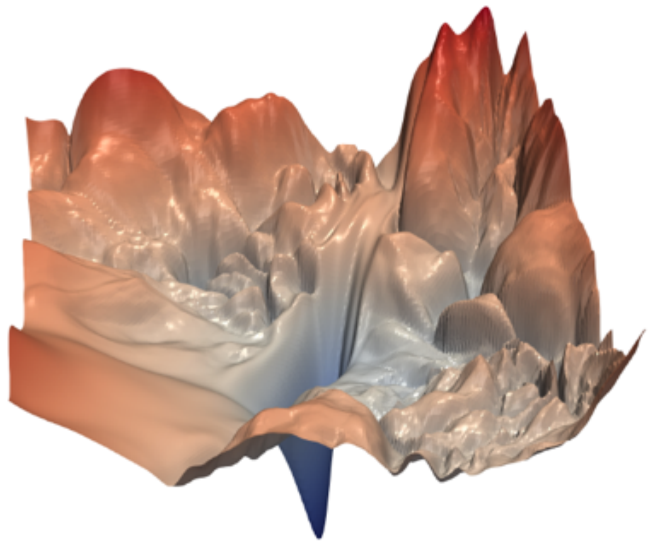
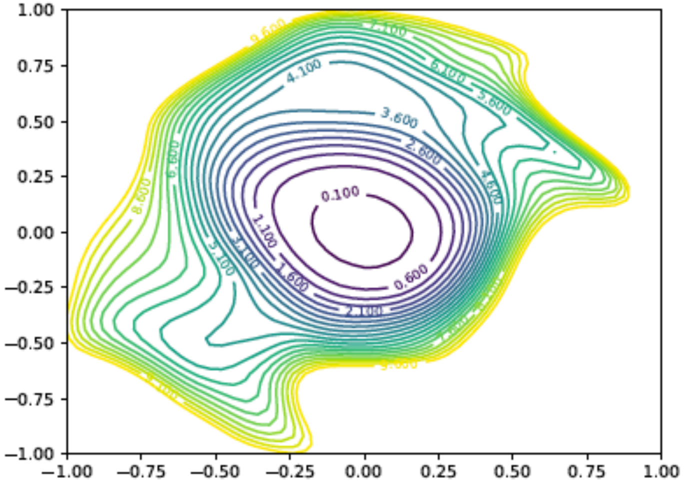


Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The vertical axis is logarithmic to show dynamic range. The x/y axes depict the same distance scale in both plots.

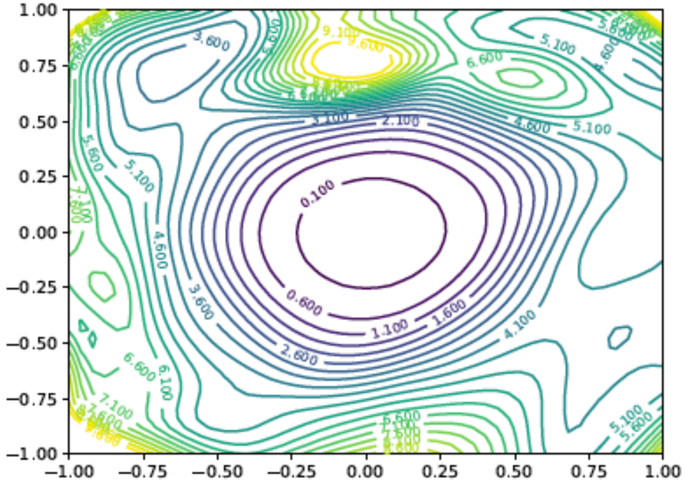
$$f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\eta)$$

Delta and eta are random directions
Alpha and beta are coefficients.

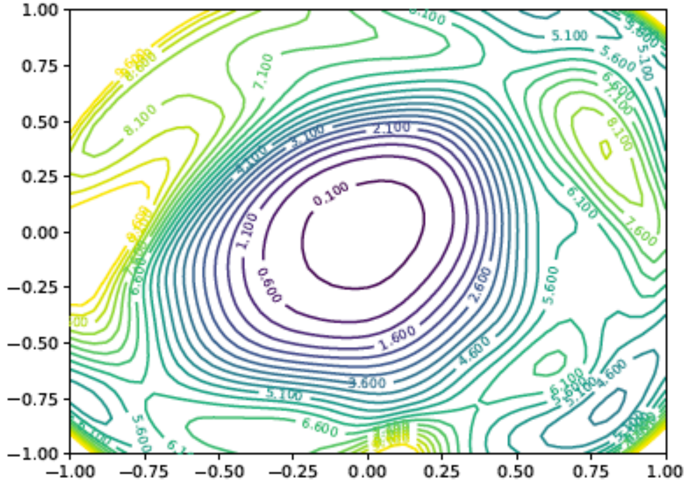
Li, Hao, et al. "Visualizing the loss landscape of neural nets." *Advances in Neural Information Processing Systems*. 2018.



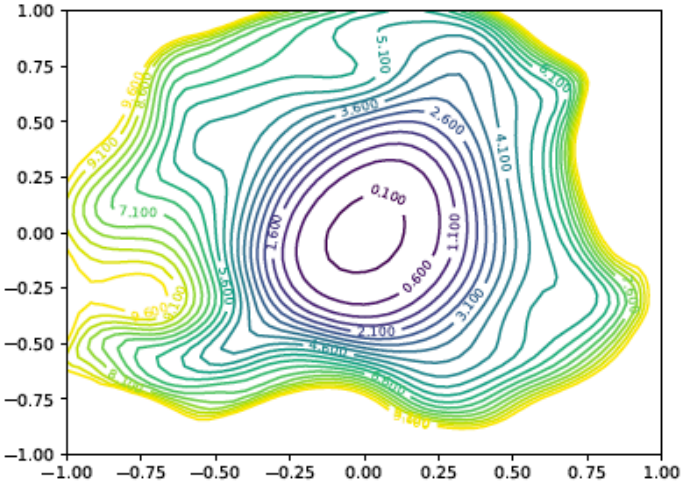
(a) ResNet-20



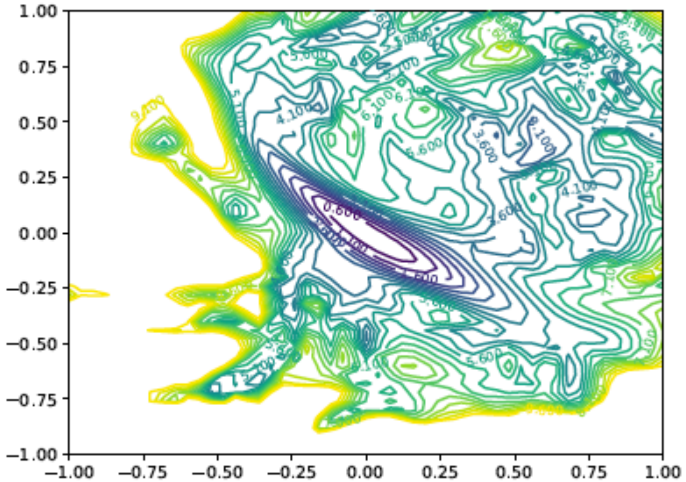
(b) ResNet-56



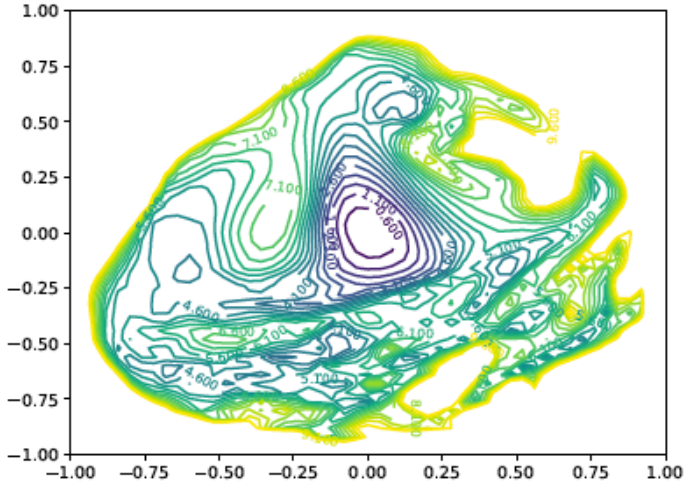
(c) ResNet-110



(d) ResNet-20-noshort



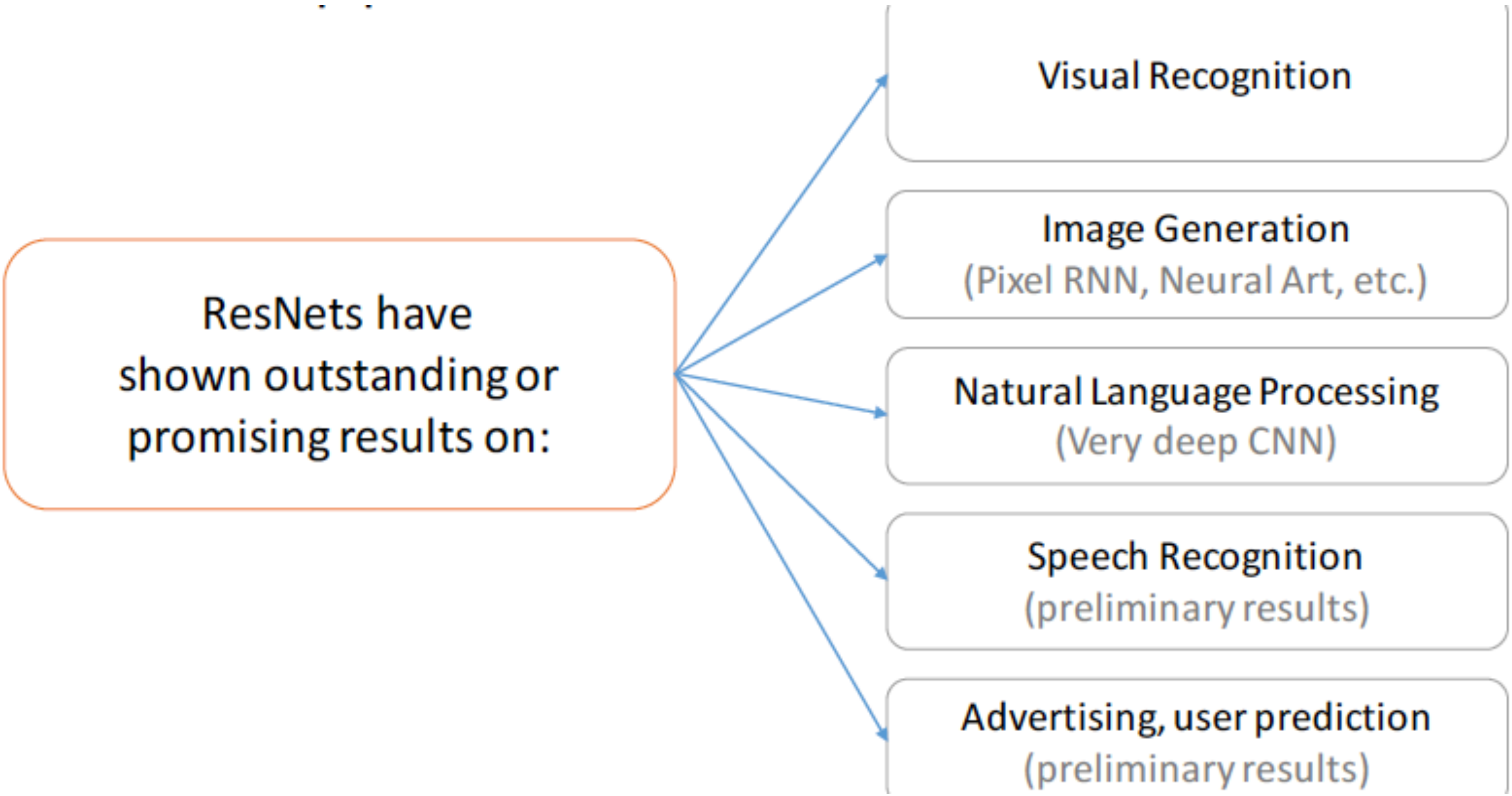
(e) ResNet-56-noshort



(f) ResNet-110-noshort

task	2nd-place winner	ResNets	margin (relative)
ImageNet Localization (top-5 error)	12.0	9.0	**27%**
ImageNet Detection (mAP@.5)	53.6	62.1	**16%**
COCO Detection (mAP@.5:.95)	33.5	37.3	**11%**
COCO Segmentation (mAP@.5:.95)	25.1	28.2	**12%**

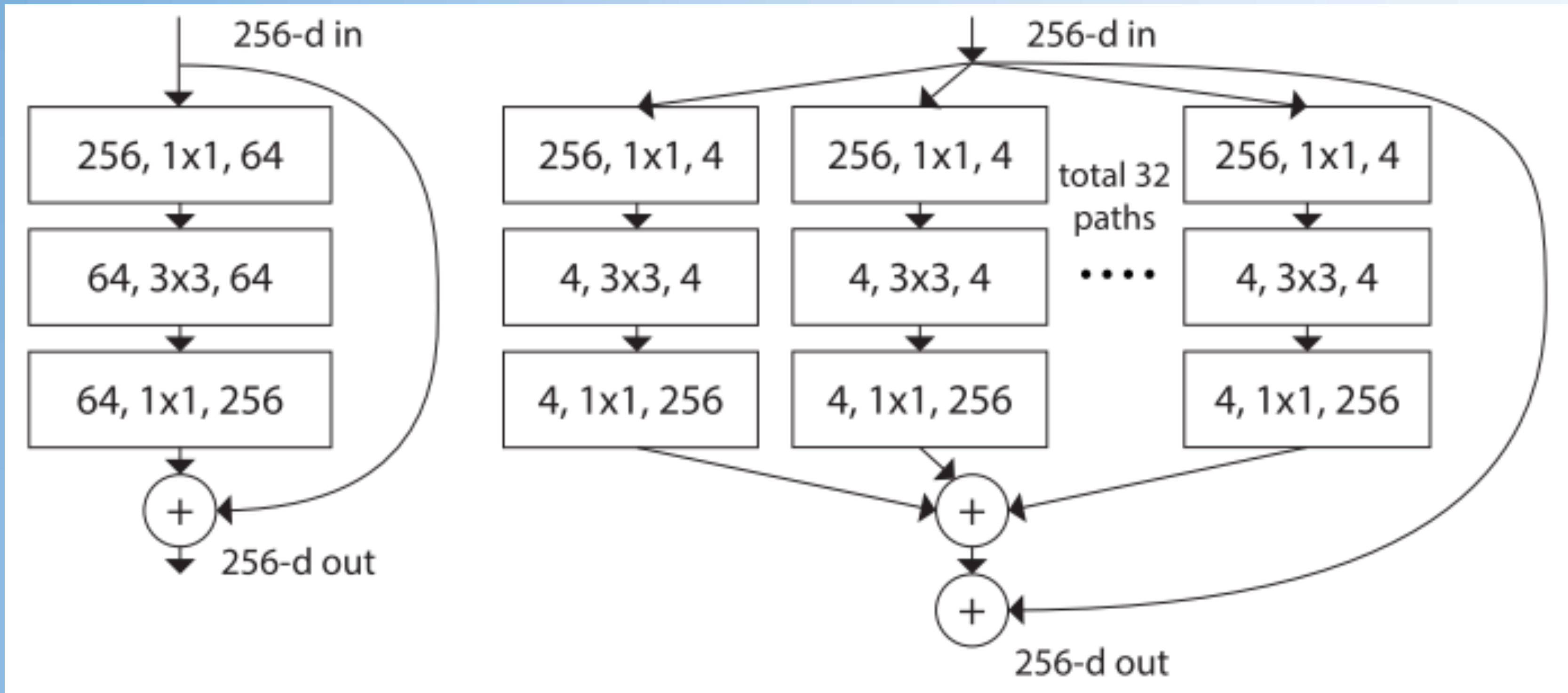
absolute 8.5% better!



ResNets have shown outstanding or promising results on:
Visual Recognition
Image Generation
(Pixel RNN, Neural Art, etc.)
Natural Language Processing
(Very deep CNN)
Speech Recognition
(preliminary results)
Advertising, user prediction
(preliminary results)

Some successful variants

- ResNeXt, FractalNet, DenseNet…



ResNeXt

Dynamical system view on ResNet and beyond

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + G(\mathbf{Y}_j, \boldsymbol{\theta}_j)$$

$$\mathbf{Y}_{j+1} = \mathbf{Y}_j + hF(\mathbf{Y}_j, \boldsymbol{\theta}_j)$$

$$\frac{\mathbf{Y}_{j+1} - \mathbf{Y}_j}{h} = F(\mathbf{Y}_j, \boldsymbol{\theta}_j)$$

$$\dot{\mathbf{Y}}(t) = F(\mathbf{Y}(t), \boldsymbol{\theta}(t)), \ \mathbf{Y}(0) = \mathbf{Y}_0, \text{ for } 0 \leq t \leq T$$

Ordinary differential equation (ODE)

Weinan, E. "A proposal on machine learning via dynamical systems." *Communications in Mathematics and Statistics* 5.1 (2017): 1-11.

Neural ODE

- Idea
  - Using ODE to represent the neural network

$$\frac{d\mathbf{h}(t)}{dt} = f(\mathbf{h}(t), t, \theta)$$

  - h(t=0) input； h(t=T) output
- Advantages
  - Memory efficiency
  - Adaptive computation
  - Parameter efficiency

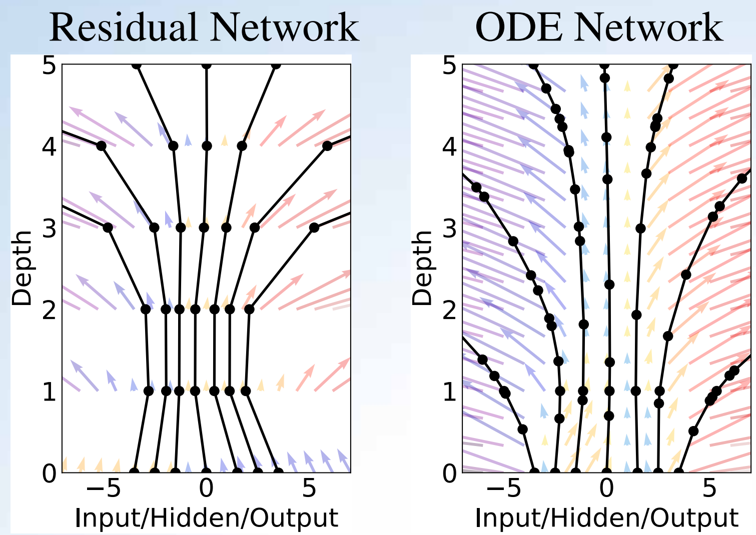


Figure 1: *Left:* A Residual network defines a discrete sequence of finite transformations. *Right:* A ODE network defines a vector field, which continuously transforms the state. *Both:* Circles represent evaluation locations.

Chen, Tian Qi, et al. "Neural ordinary differential equations." *Advances in neural information processing systems*. 2018.

Loss function:

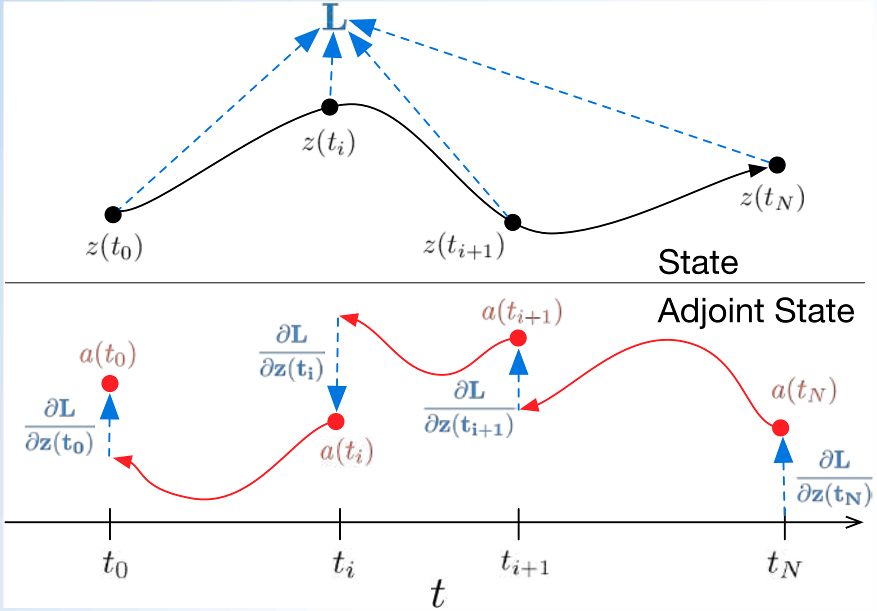
$$L(\mathbf{z}(t_1)) = L\left(\mathbf{z}(t_0) + \int_{t_0}^{t_1} f(\mathbf{z}(t), t, \theta)dt\right) = L\left(\text{ODESolve}(\mathbf{z}(t_0), f, t_0, t_1, \theta)\right)$$

Gradient:

$$adjoint\ \mathbf{a}(t) = {}^{\partial L}/_{\partial \mathbf{z}(t)}$$

$$\frac{d\mathbf{a}(t)}{dt} = -\mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \mathbf{z}}$$

$$\frac{dL}{d\theta} = \int_{t_1}^{t_0} \mathbf{a}(t)^{\mathsf{T}} \frac{\partial f(\mathbf{z}(t), t, \theta)}{\partial \theta} dt$$



Algorithm 1 Reverse-mode derivative of an ODE initial value problem

Input: dynamics parameters θ, start time t_0, stop time t_1, final state $\mathbf{z}(t_1)$, loss gradient ${}^{\partial L}/_{\partial \mathbf{z}(t_1)}$

$s_0 = [\mathbf{z}(t_1), \frac{\partial L}{\partial \mathbf{z}(t_1)}, \mathbf{0}_{|\theta|}]$ ▷ Define initial augmented state

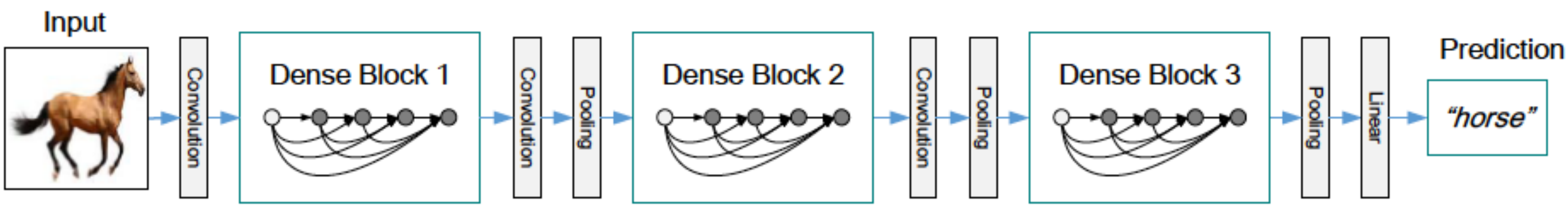
def aug_dynamics($[\mathbf{z}(t), \mathbf{a}(t), \cdot], t, \theta$): ▷ Define dynamics on augmented state

return $[f(\mathbf{z}(t), t, \theta), -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \mathbf{z}}, -\mathbf{a}(t)^\mathsf{T} \frac{\partial f}{\partial \theta}]$ ▷ Compute vector-Jacobian products

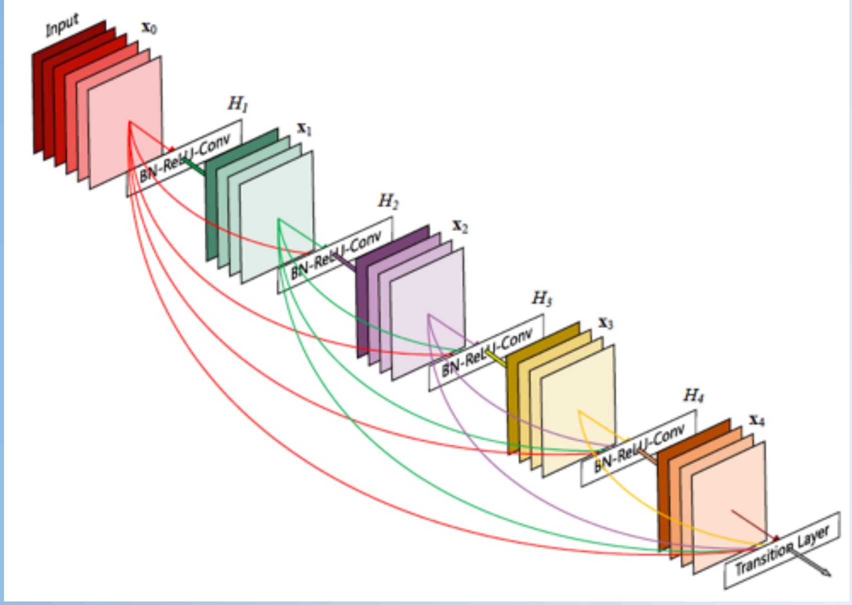
$[\mathbf{z}(t_0), \frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$ ▷ Solve reverse-time ODE

return $\frac{\partial L}{\partial \mathbf{z}(t_0)}, \frac{\partial L}{\partial \theta}$ ▷ Return gradients

DenseNet



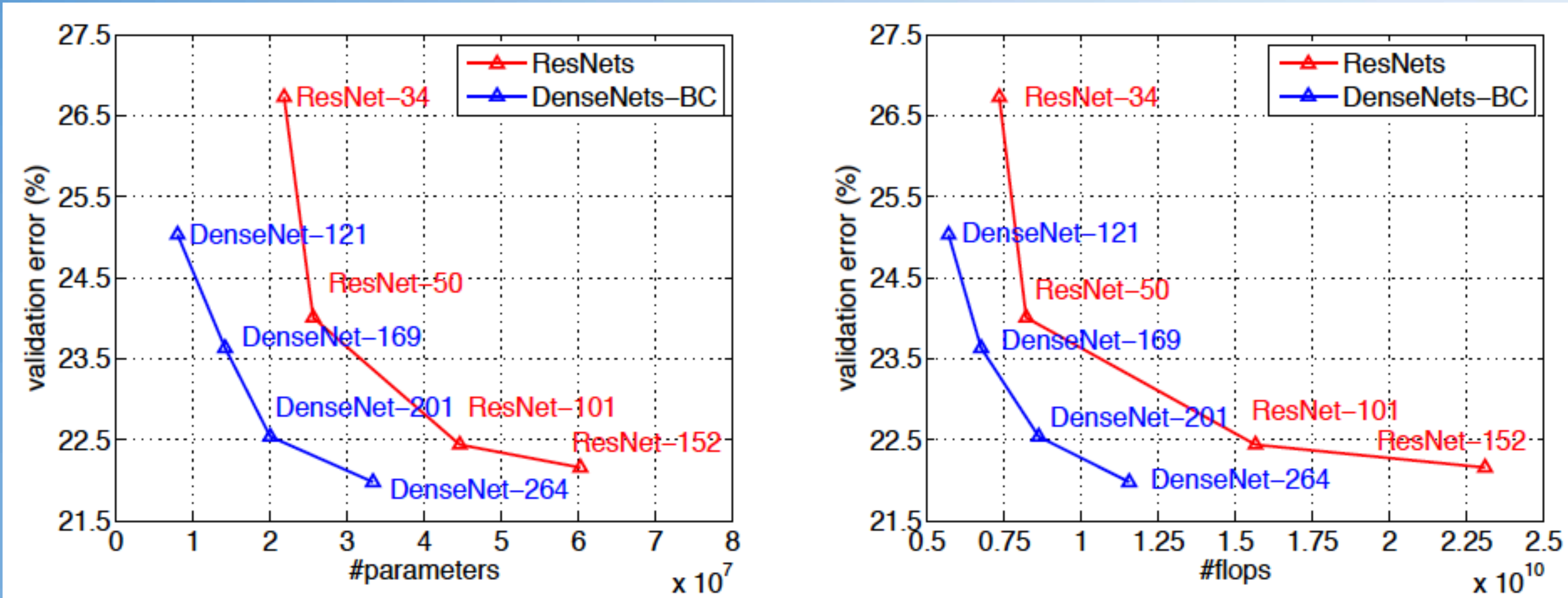
A dense block



$$\mathbf{x}_\ell = H_\ell([\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{\ell-1}])$$

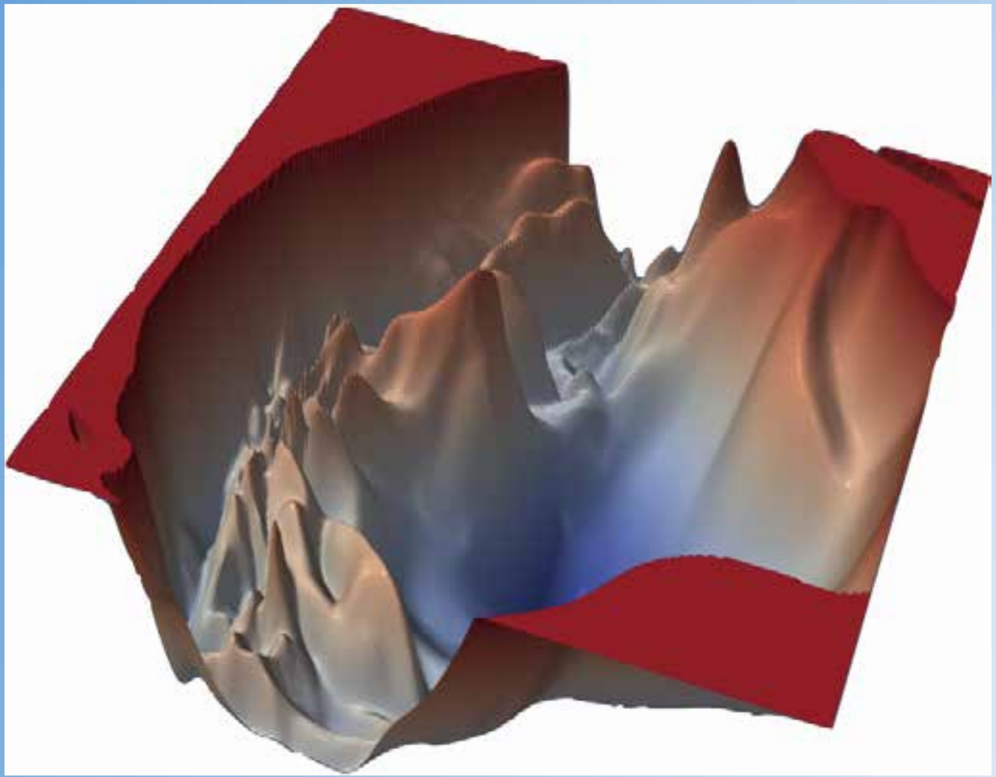
Multi-scale feature aggregation by concatenation

Huang et.al. Densely connected convolutional networks. CVPR 2017

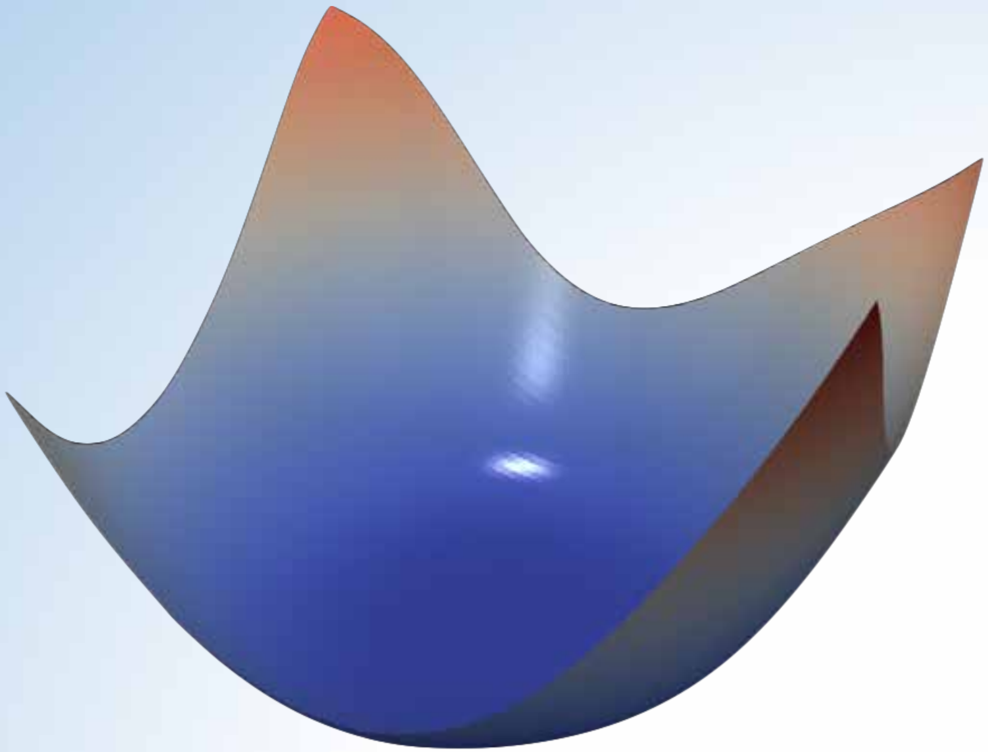


ResNets
DenseNets–BC
ResNet–34
ResNet–50
ResNet–101
ResNet–152
DenseNet–121
DenseNet–169
DenseNet–201
DenseNet–264
validation error (%)
27.5
26.5
25.5
24.5
23.5
22.5
21.5
0
1
2
3
4
5
6
7
8
#parameters
x 10^7
0.5
0.75
1
1.25
1.5
1.75
2
2.25
2.5
#flops
x 10^10

CIFAR-100



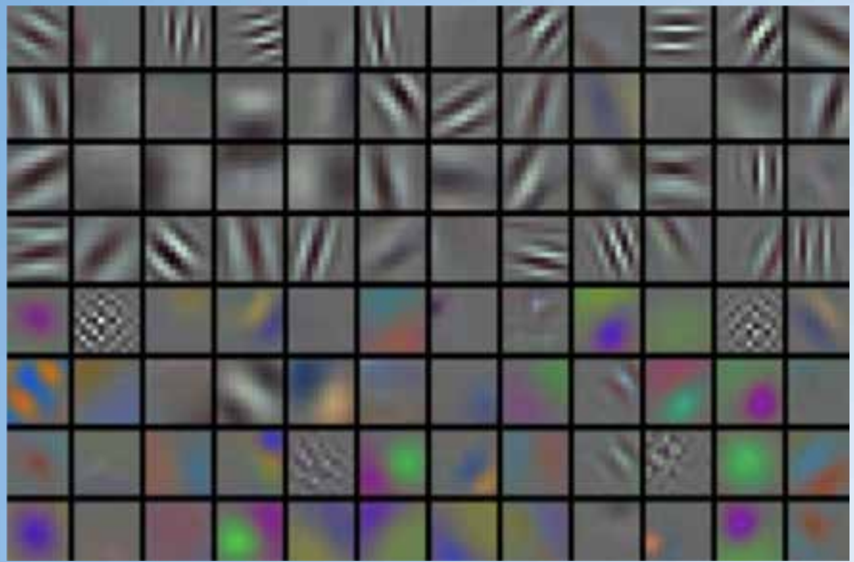
(a) ResNet-110, no skip connections



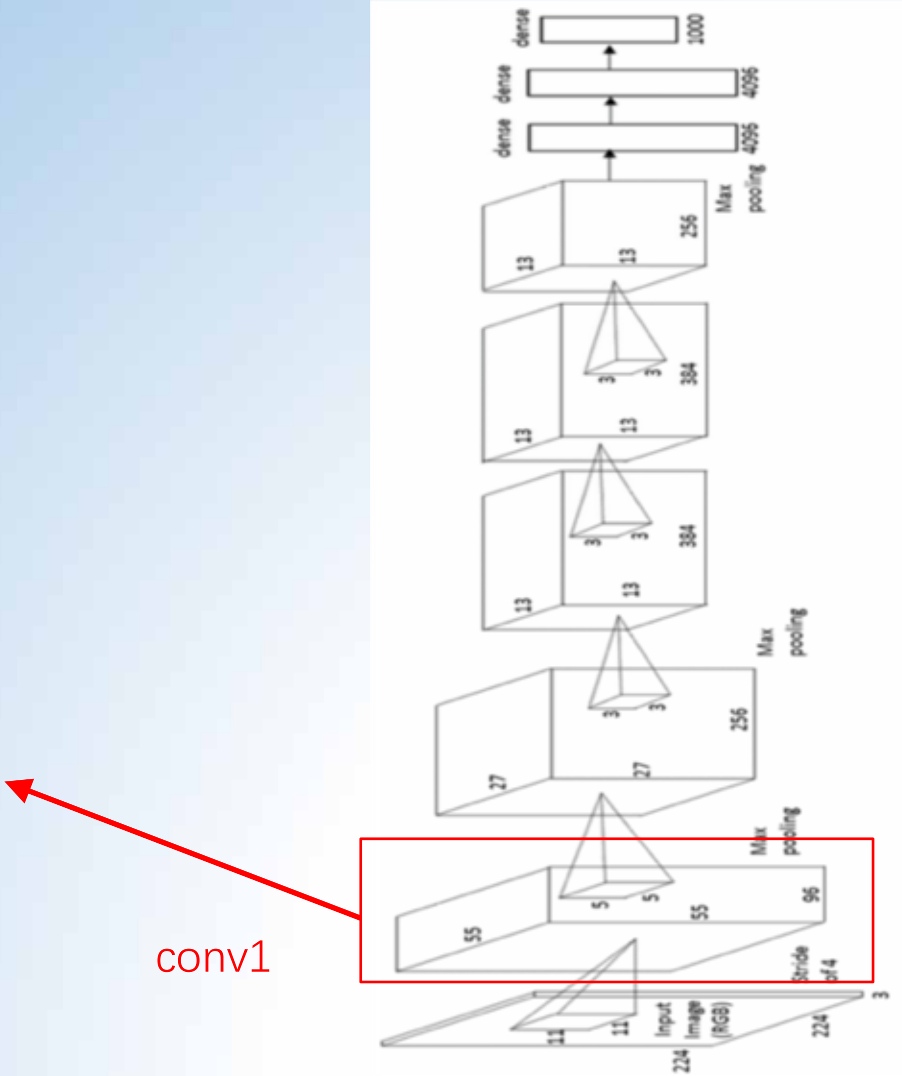
(b) DenseNet, 121 layers

Visualizing CNNs

Visualize filters (weights)

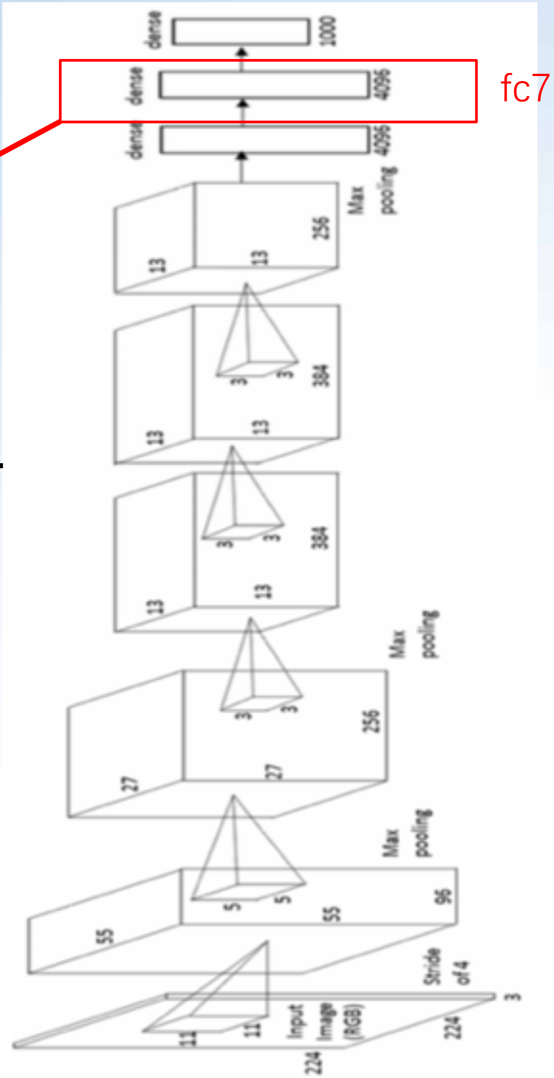


Edge, blob detectors.
Interpretable only for the first layer.

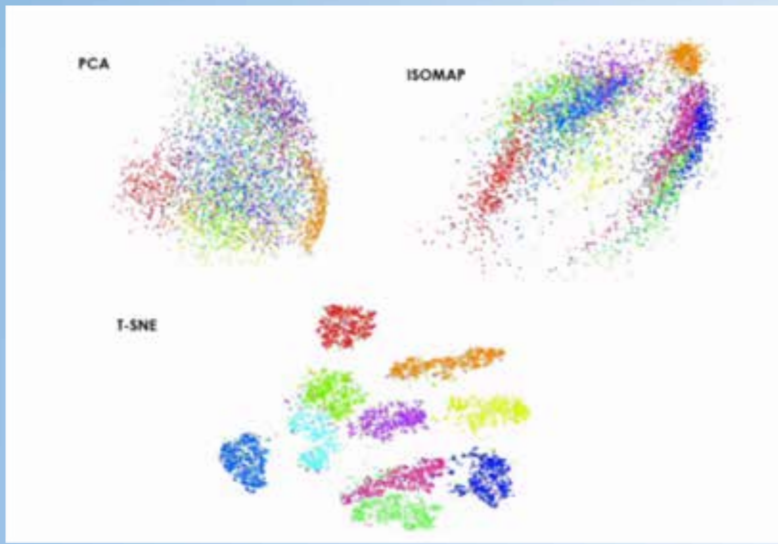


One stream AlexNet

Visualize the representation

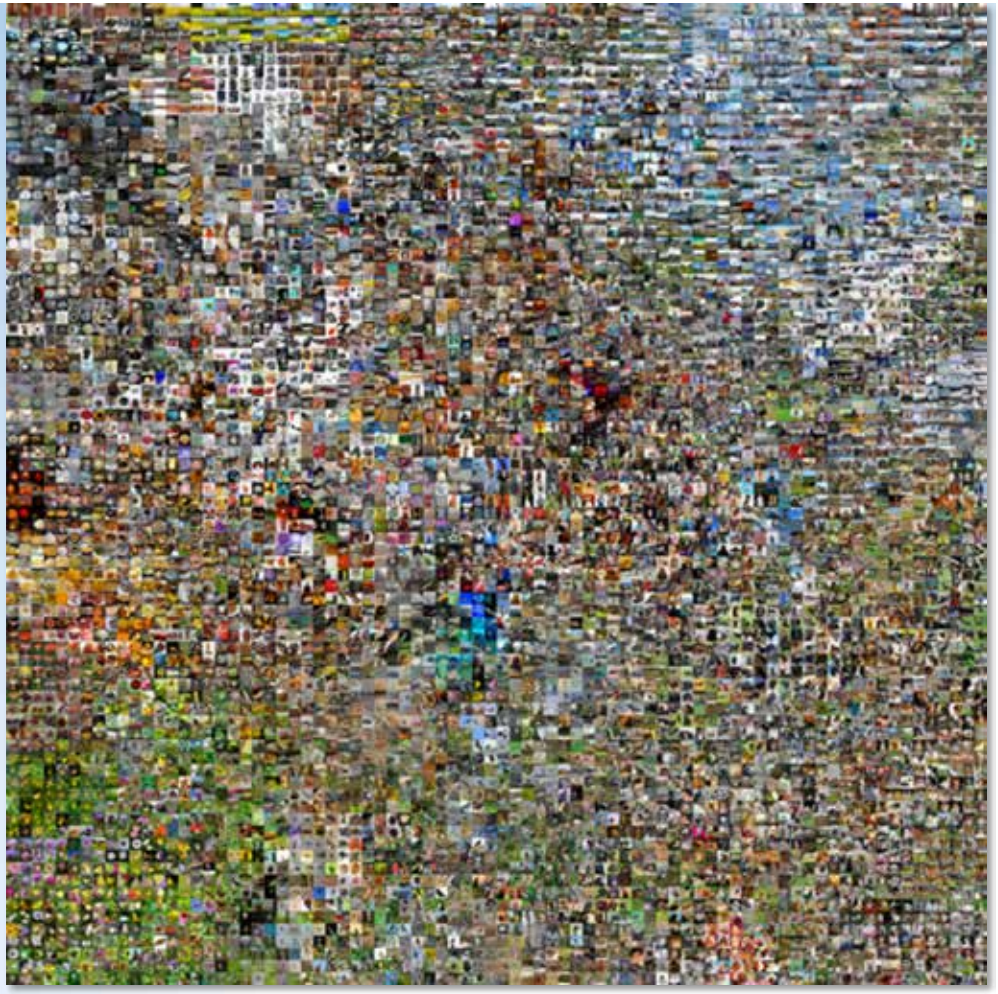


4096-dimensional representation of the input signal.



t-SNE visualization

Two images are placed nearby if their CNN codes are close.



Maaten, L.v.d. and Hinton, G., 2008. **Visualizing data using t-SNE** Journal of Machine Learning Research, Vol 9(Nov), pp. 2579—2605

tSNE in one slide

- Project high-dimensional data into 2D or 3D

$$\mathcal{X} = \{x_1, x_2, \ldots, x_n \in \mathbb{R}^h\} \rightarrow \mathcal{Y} = \{y_1, y_2, \ldots, y_n \in \mathbb{R}^l\}$$

$$\min_{\mathcal{Y}} C(\mathcal{X}, \mathcal{Y})$$

$$p_{j|i} = \frac{\exp(- \| x_i - x_j \|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(- \| x_i - x_k \|^2 / 2\sigma_i^2)}$$

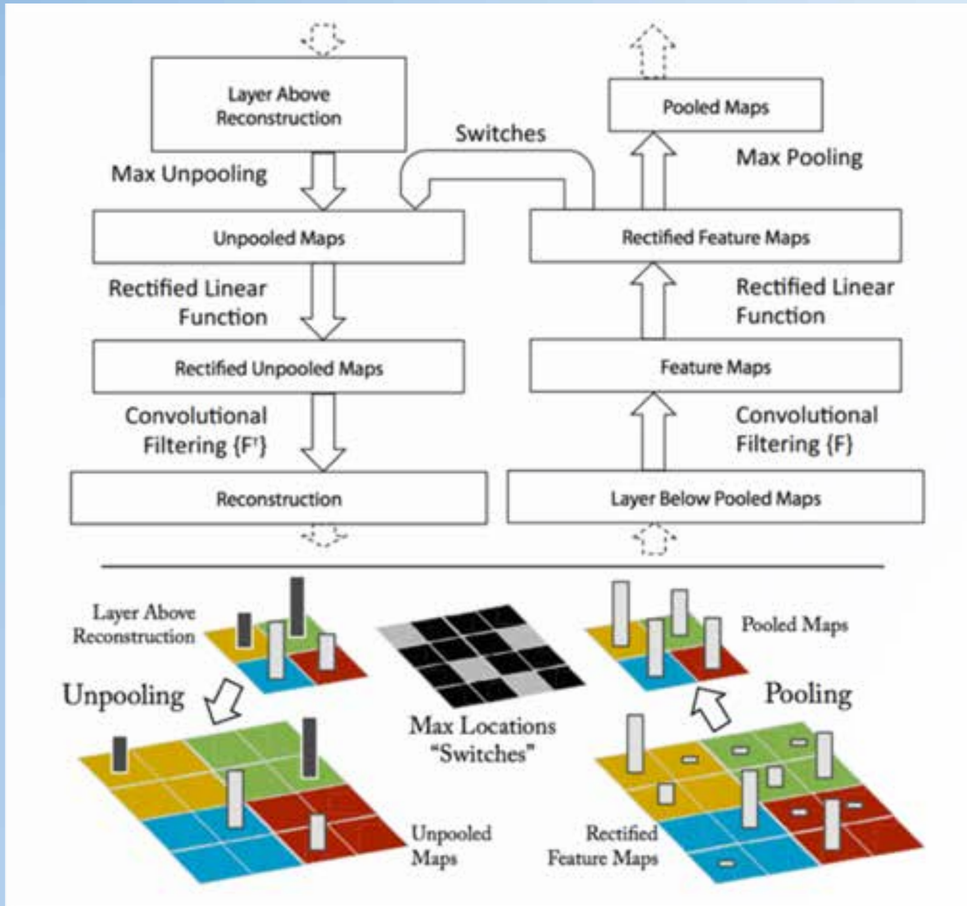
$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

Cost Function:

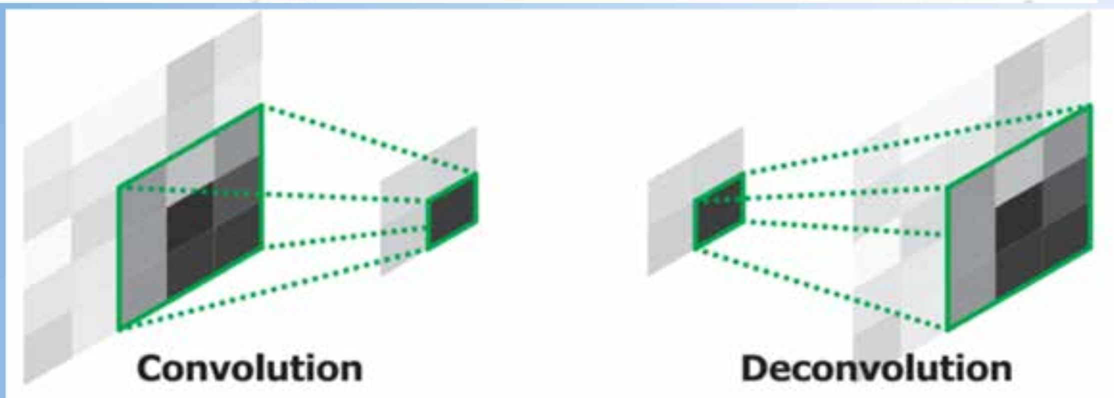
$$C = KL(P||Q)$$

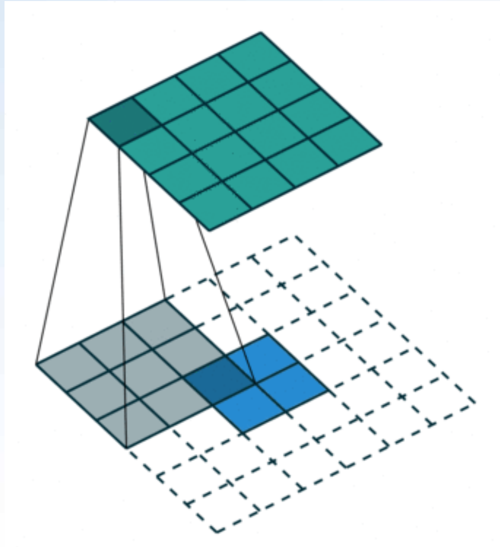
Deconvolution approach



Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." *European conference on computer vision*. Springer International Publishing, 2014.

Deconvolution approach

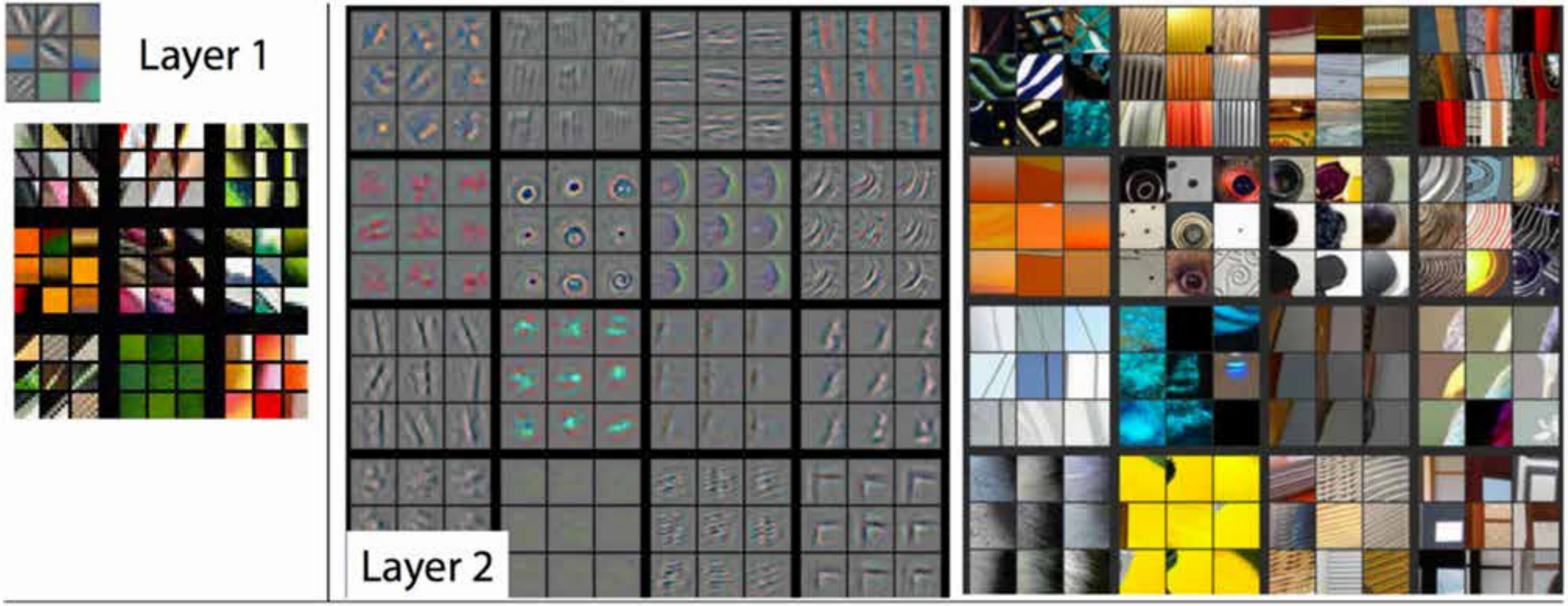




FCN reference

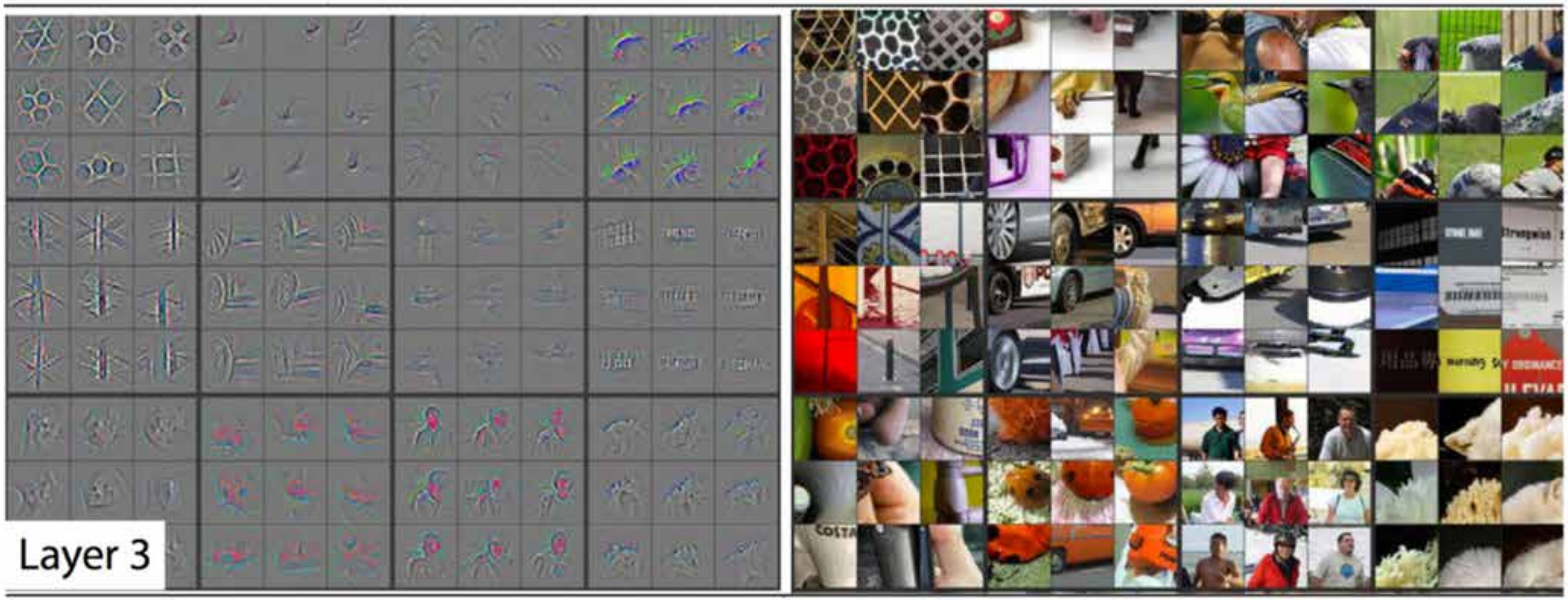
Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation.

Deconvolution approach



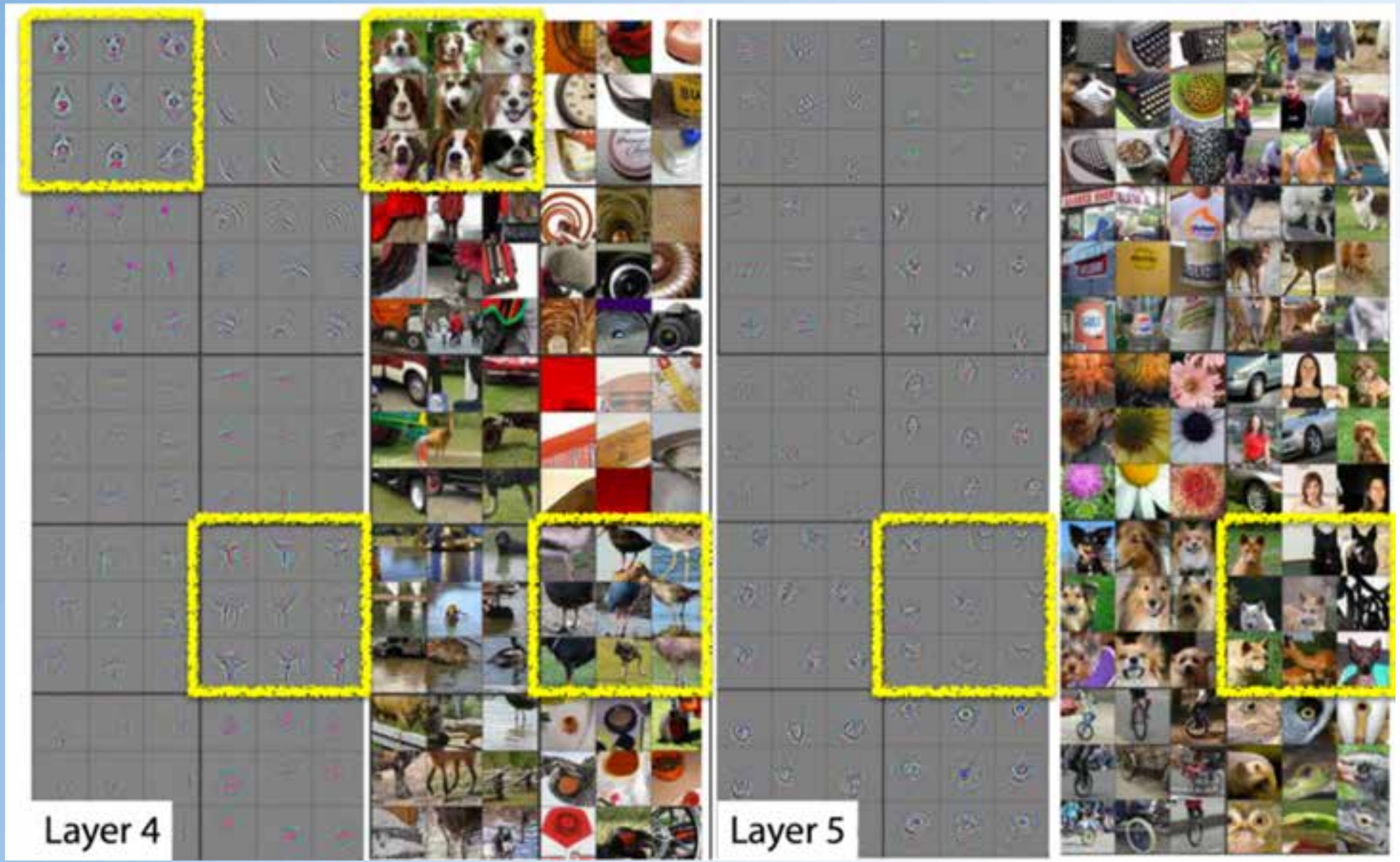
Corners & edge/color conjunctions

Deconvolution approach



Similar textures

Deconvolution approach

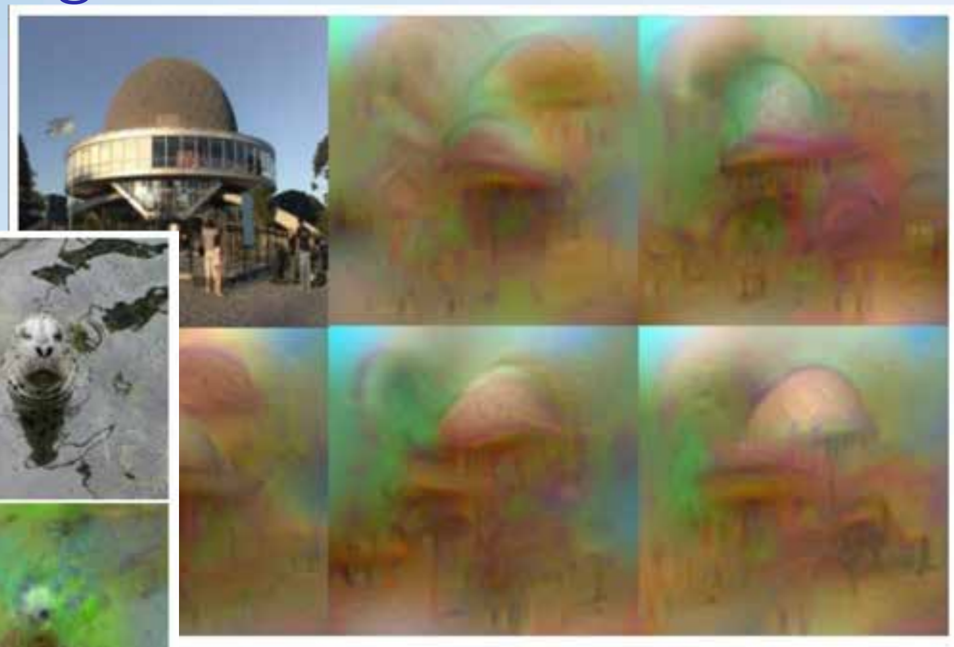


Object parts　　　　Entire object

Optimization over input image

Question: given a CNN code, is it possible to reconstruct the original image?

Optimization over input image



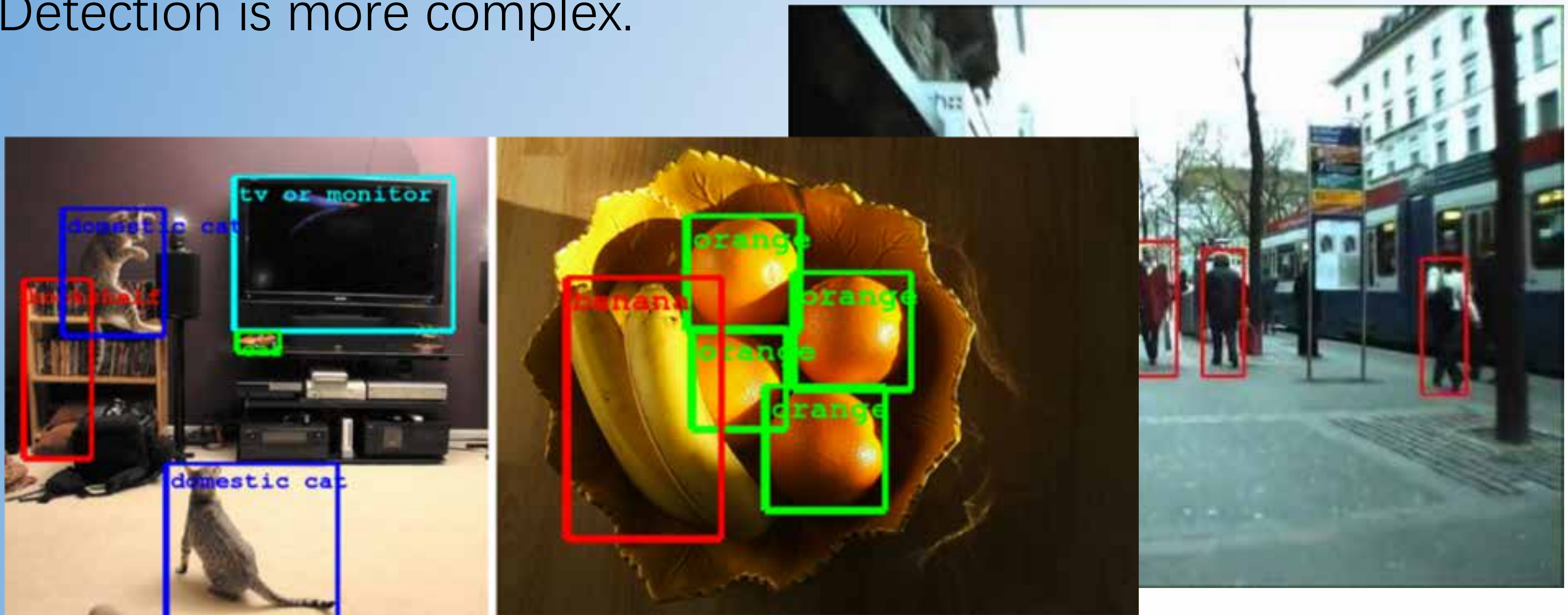
$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

Understanding Deep Image Representations by Inverting Them [Mahendran and Vedaldi, 2014]

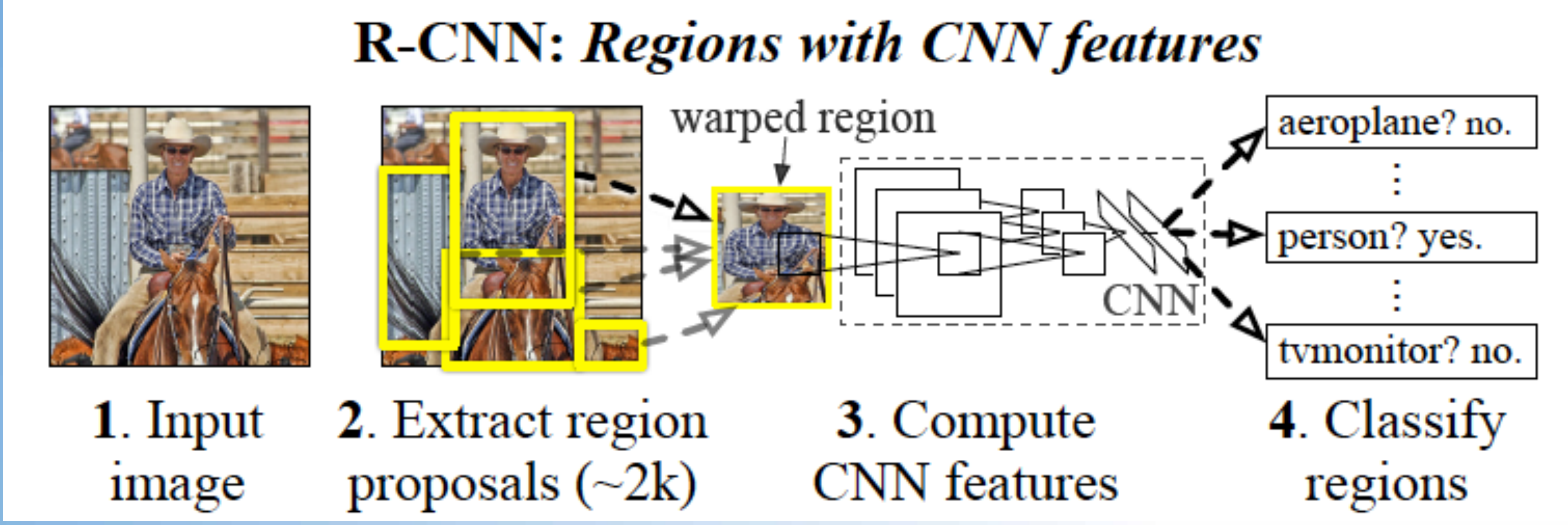
Applications of CNNs

Object recognition & detection

Recognition as a classification problem.
Detection is more complex.



R-CNN, Fast R-CNN, Faster R-CNN…



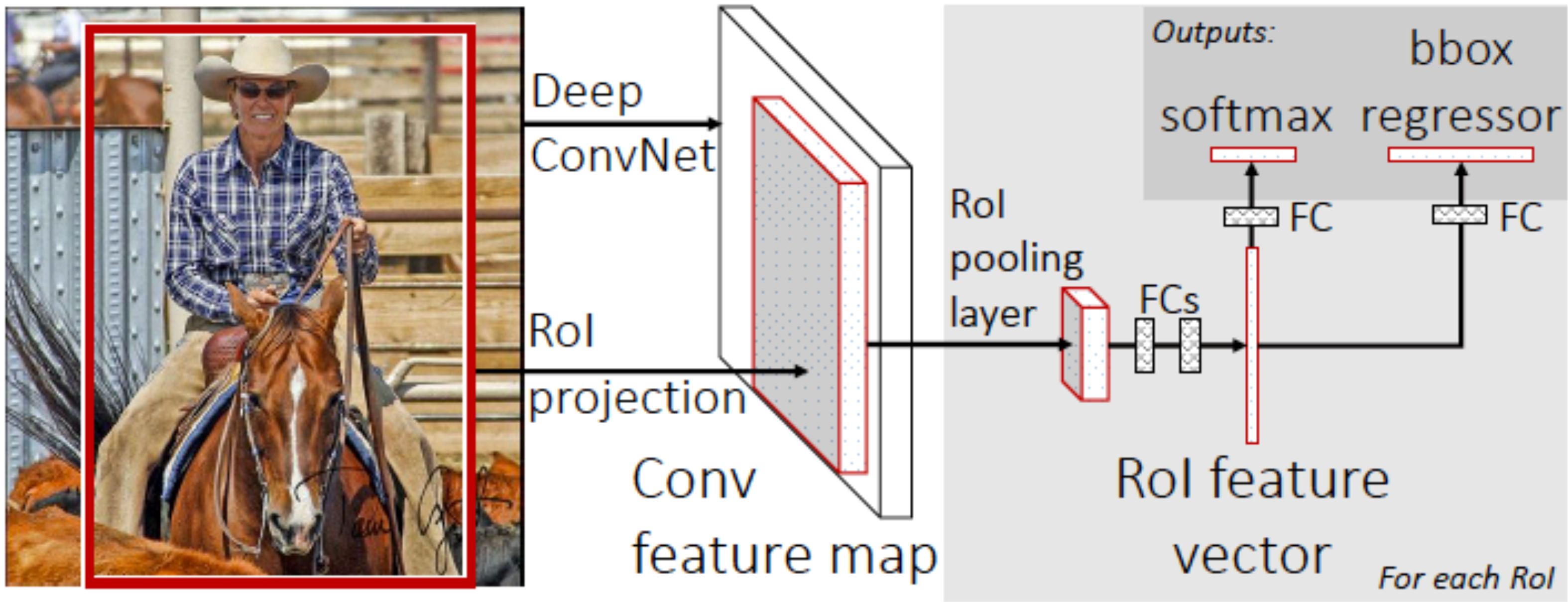
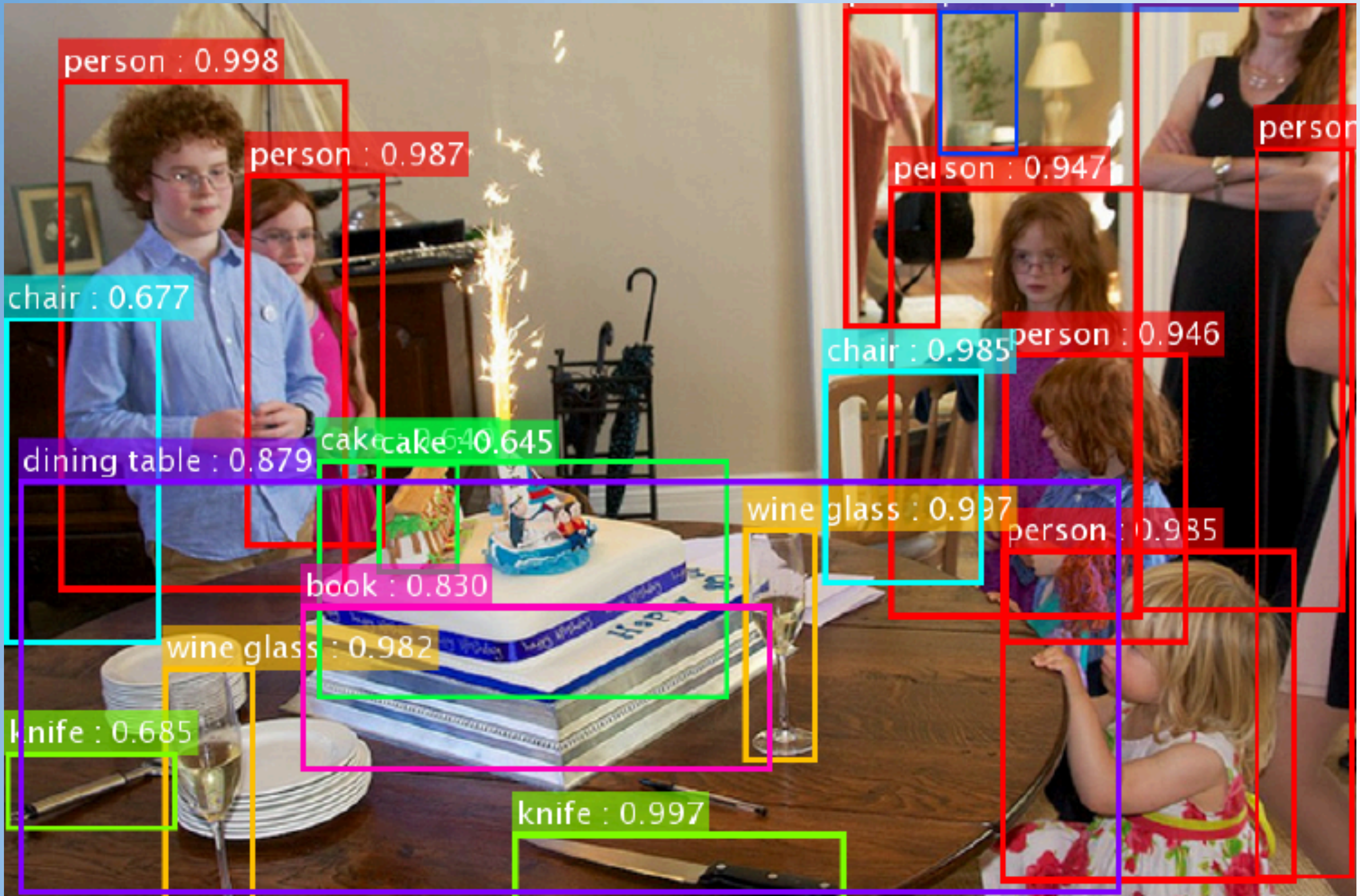


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

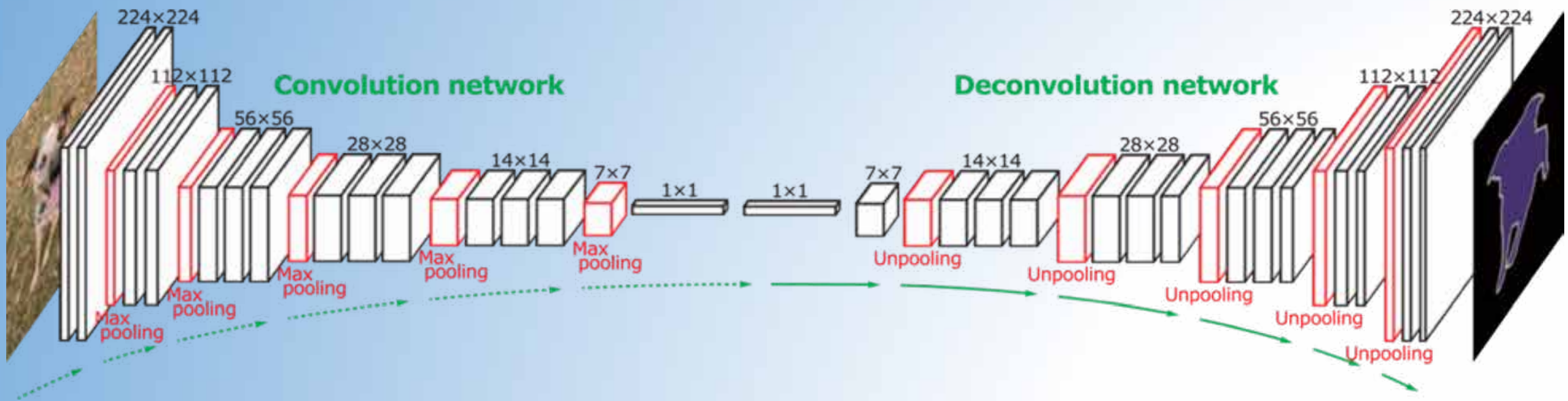


person : 0.998
person : 0.987
chair : 0.677
dining table : 0.879
cake : 0.645
book : 0.830
wine glass : 0.982
knife : 0.685
knife : 0.997
wine glass : 0.997
chair : 0.985
person : 0.947
person : 0.946
person : 0.985
person

Semantic segmentation

Semantic segmentation

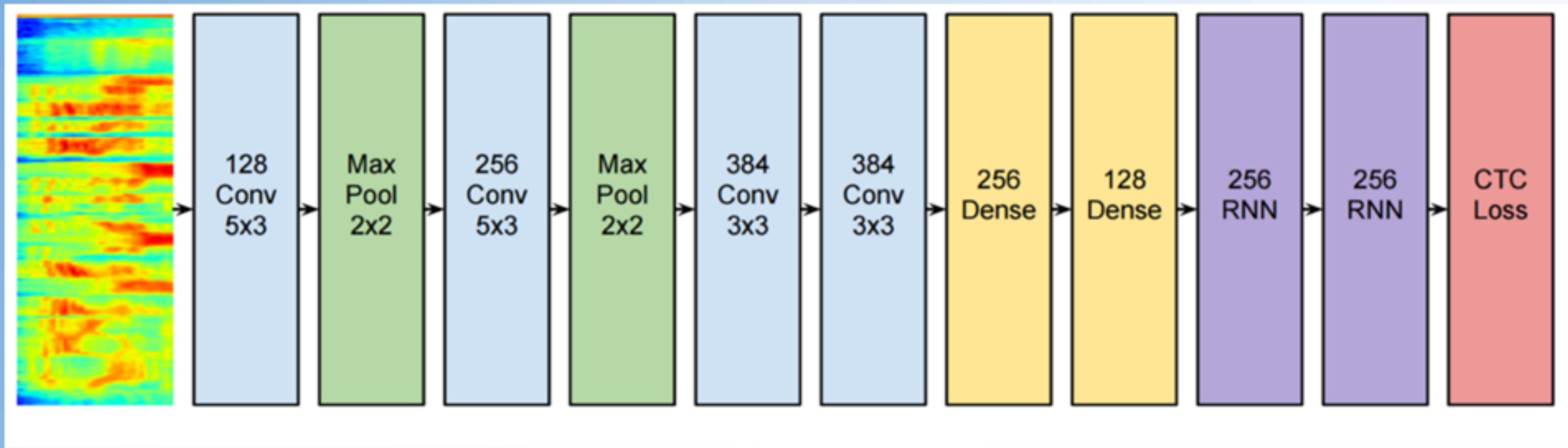
Remove FC layers. Based on VGG-16.



Hyeonwoo Noh, et.al. Learning deconvolution network for semantic segmentation. In ICCV, 2015.

Speech recognition

MFCC feature or STFT

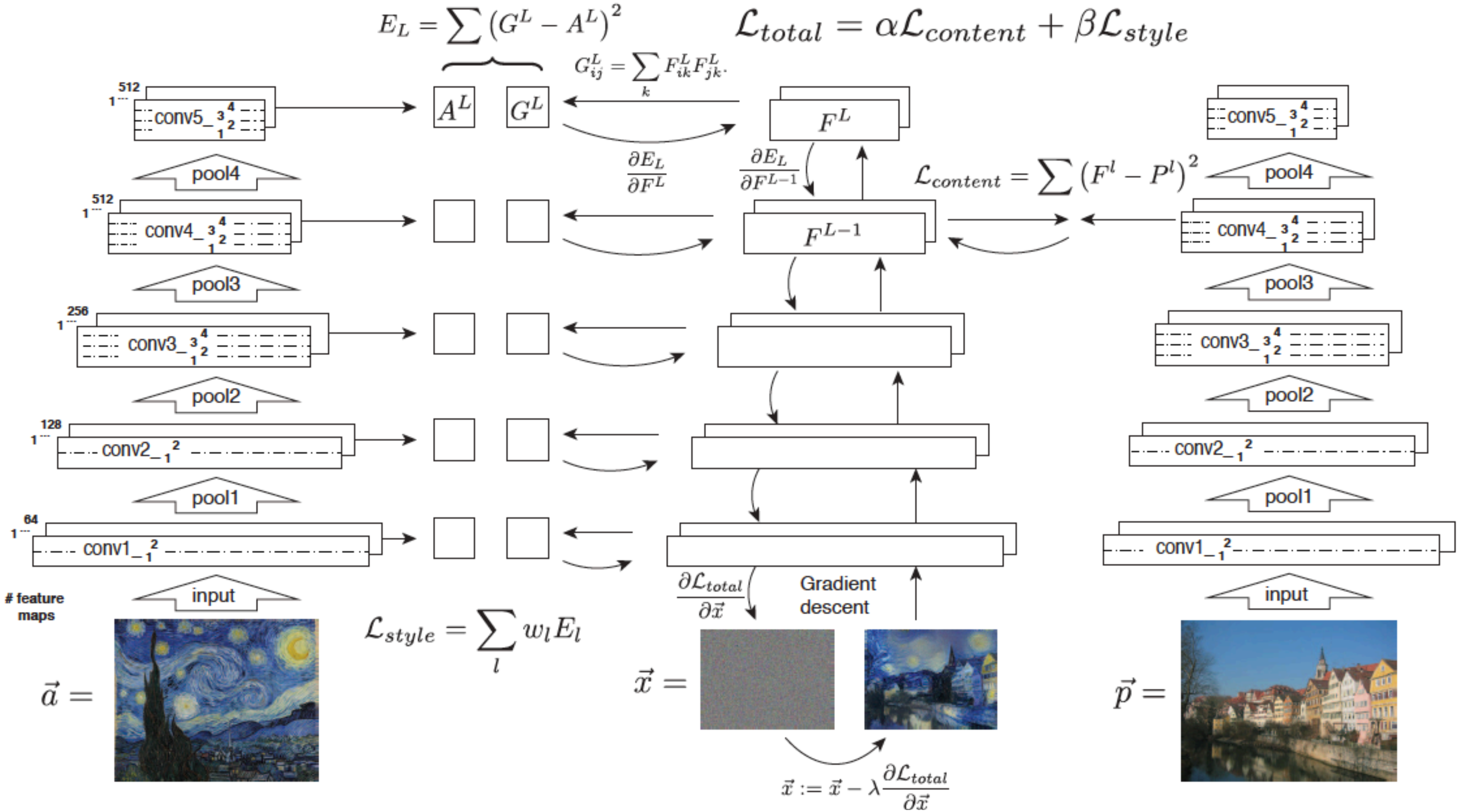


Neural Style

Picture credit to Harish Narayanan.

Gatys, Leon A. et.al. "A neural algorithm of artistic style." *arXiv preprint arXiv:1508.06576* (2015).

PRISMA


$E_L = \sum (G^L - A^L)^2$
$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$
$G^L_{ij} = \sum_k F^L_{ik} F^L_{jk}$
A^L
G^L
F^L
F^{L-1}
$\frac{\partial E_L}{\partial F^L}$
$\frac{\partial E_L}{\partial F^{L-1}}$
$\mathcal{L}_{content} = \sum (F^l - P^l)^2$
512
256
128
64
conv5_1 2 3 4
pool4
conv4_1 2 3 4
pool3
conv3_1 2 3 4
pool2
conv2_1 2
pool1
conv1_1 2
input
feature maps
$\vec{a} =$
$\mathcal{L}_{style} = \sum_l w_l E_l$
$\frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
Gradient descent
$\vec{x} =$
$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial \vec{x}}$
$\vec{p} =$

Neural Style

Basic net: VGG19, 16 convolutions+5 pooling

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

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

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

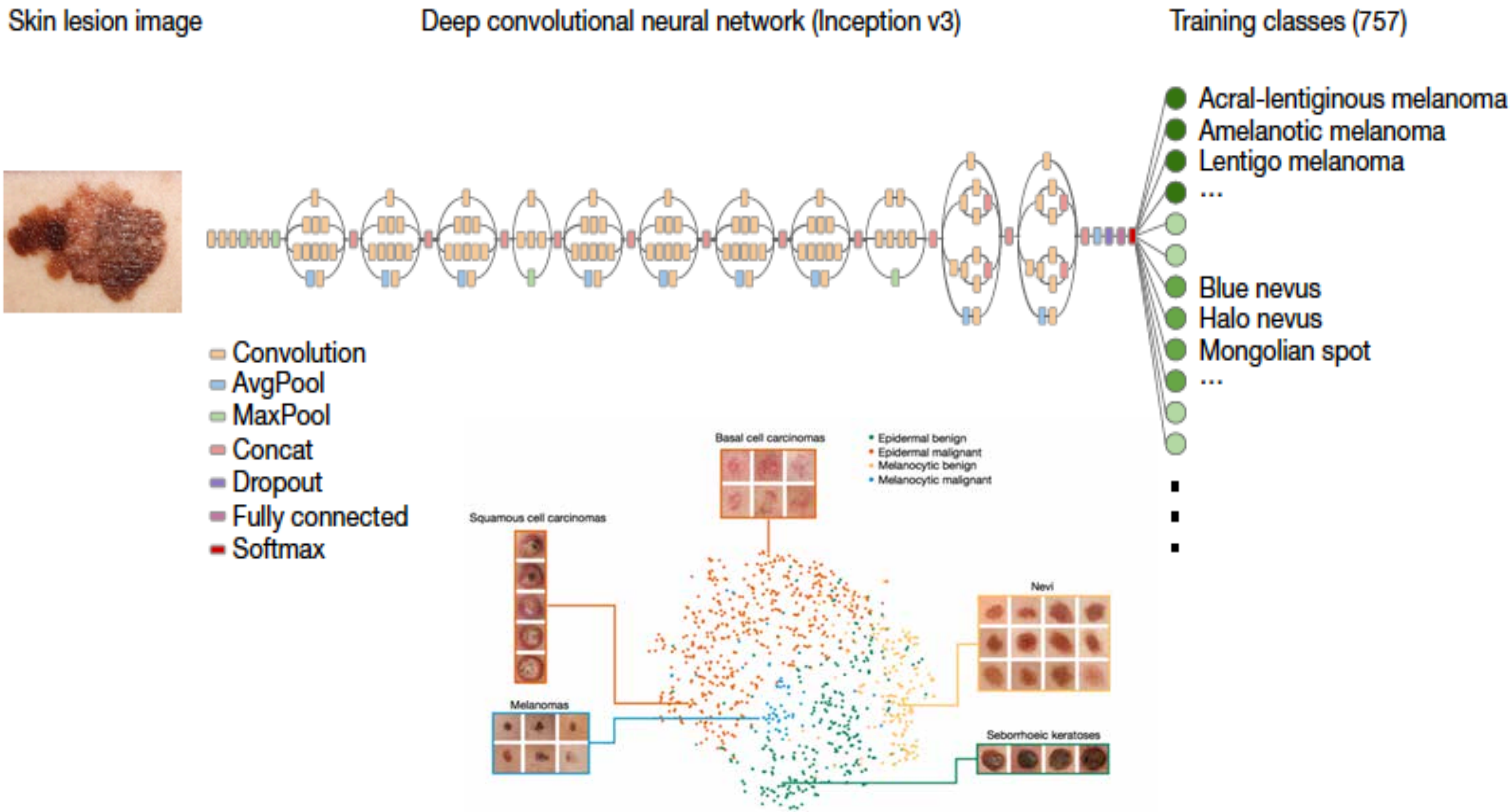
$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}.$$

$$x* = \arg\min_{x} \alpha \mathcal{L}_{\text{content}}(c, x) + \beta \mathcal{L}_{\text{style}}(s, x)$$

Healthcare: skin cancer classification

- **Skin images**
- **CNN with pretrained parameters from ImageNet**
- **Training: 129,450 clinical images, 2,032 diseases**
- **Overall Accuracy: 72.1%**

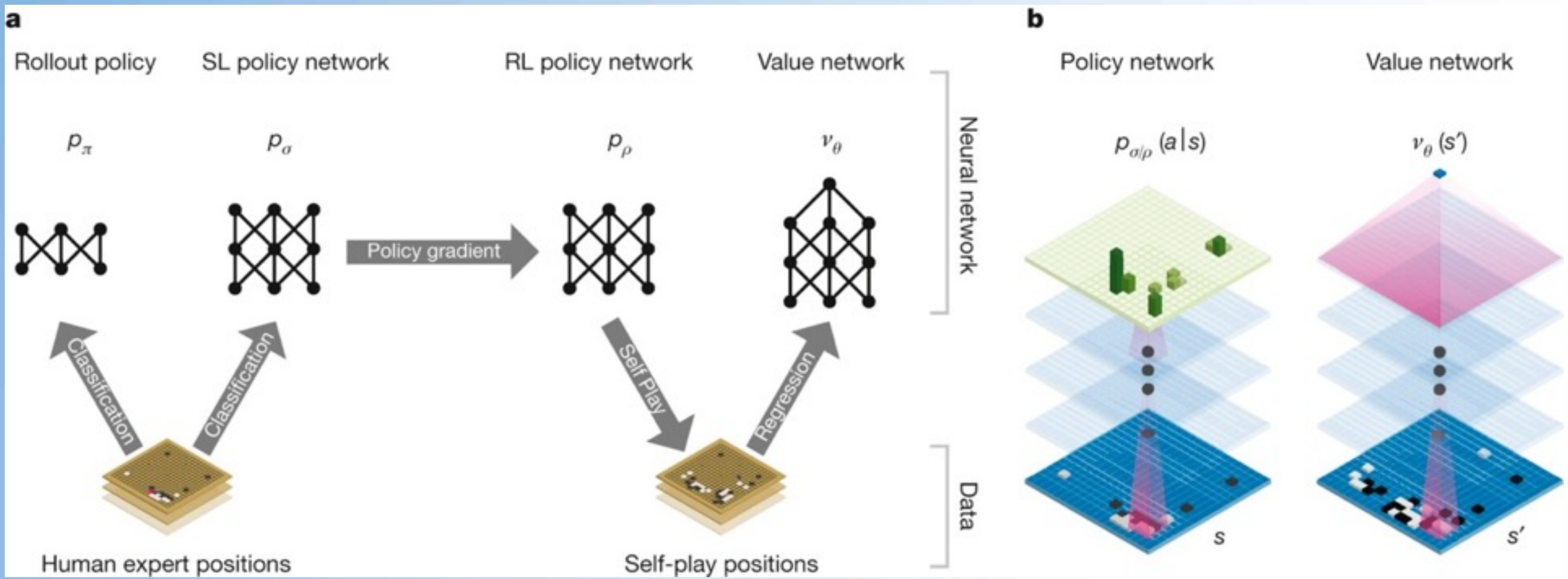
Andre Esteva, et al. "Dermatologist-level classification of skin cancer with deep neural networks." *Nature (2017)*

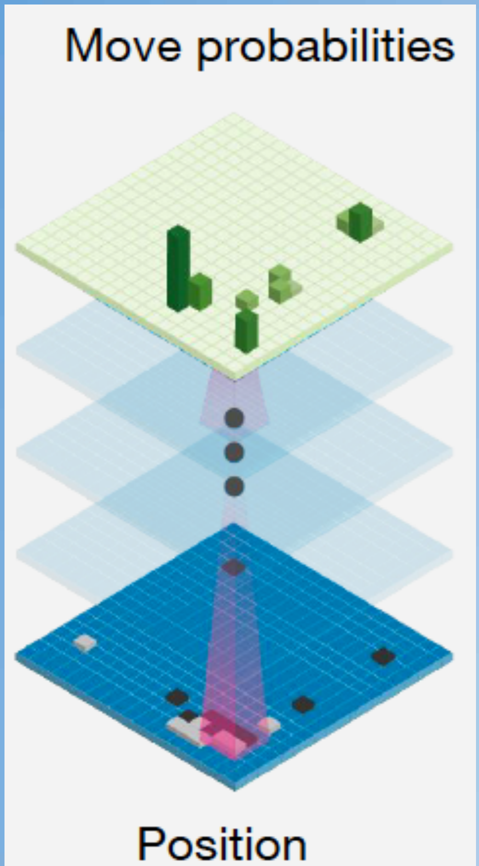

Skin lesion image
Deep convolutional neural network (Inception v3)
Training classes (757)
Acral-lentiginous melanoma
Amelanotic melanoma
Lentigo melanoma
...
Blue nevus
Halo nevus
Mongolian spot
...
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax
Basal cell carcinomas
Epidermal benign
Epidermal malignant
Melanocytic benign
Melanocytic malignant
Squamous cell carcinomas
Nevi
Melanomas
Seborrhoeic keratoses

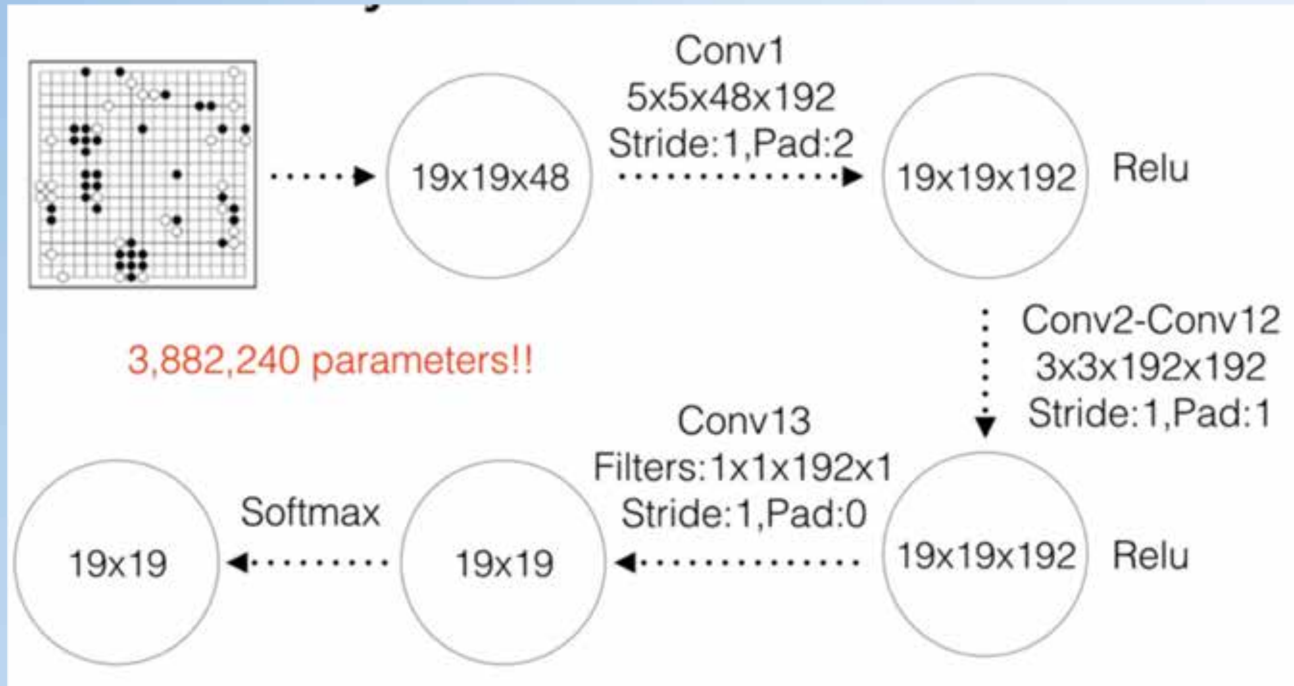
AlphaGo

Structure: Searching procedure + ConvNets

2 policy networks + 1 value network




Move probabilities
Position


19x19x48
Conv1
5x5x48x192
Stride:1,Pad:2
19x19x192
Relu
Conv2-Conv12
3x3x192x192
Stride:1,Pad:1
3,882,240 parameters!!
Conv13
Filters:1x1x192x1
Stride:1,Pad:0
19x19x192
Relu
Softmax
19x19
19x19

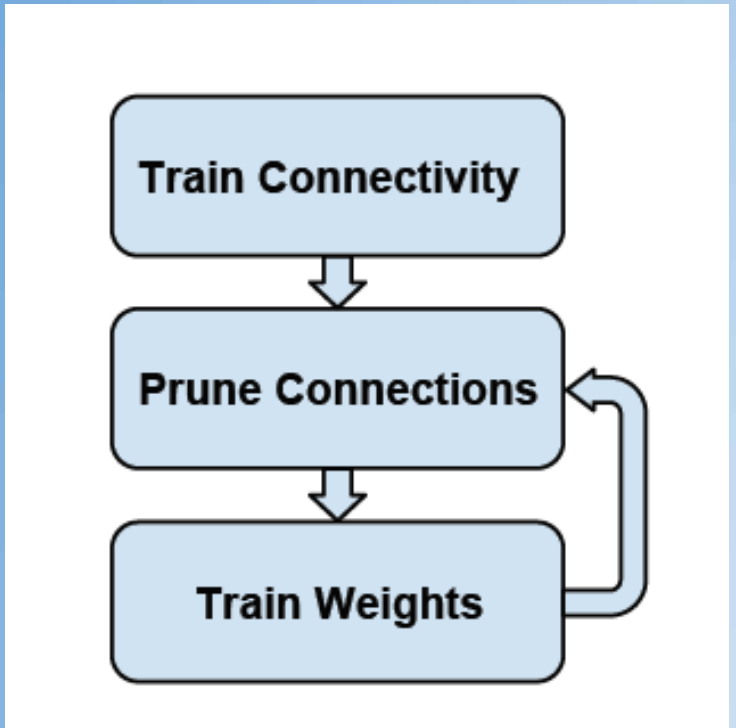
Model compression

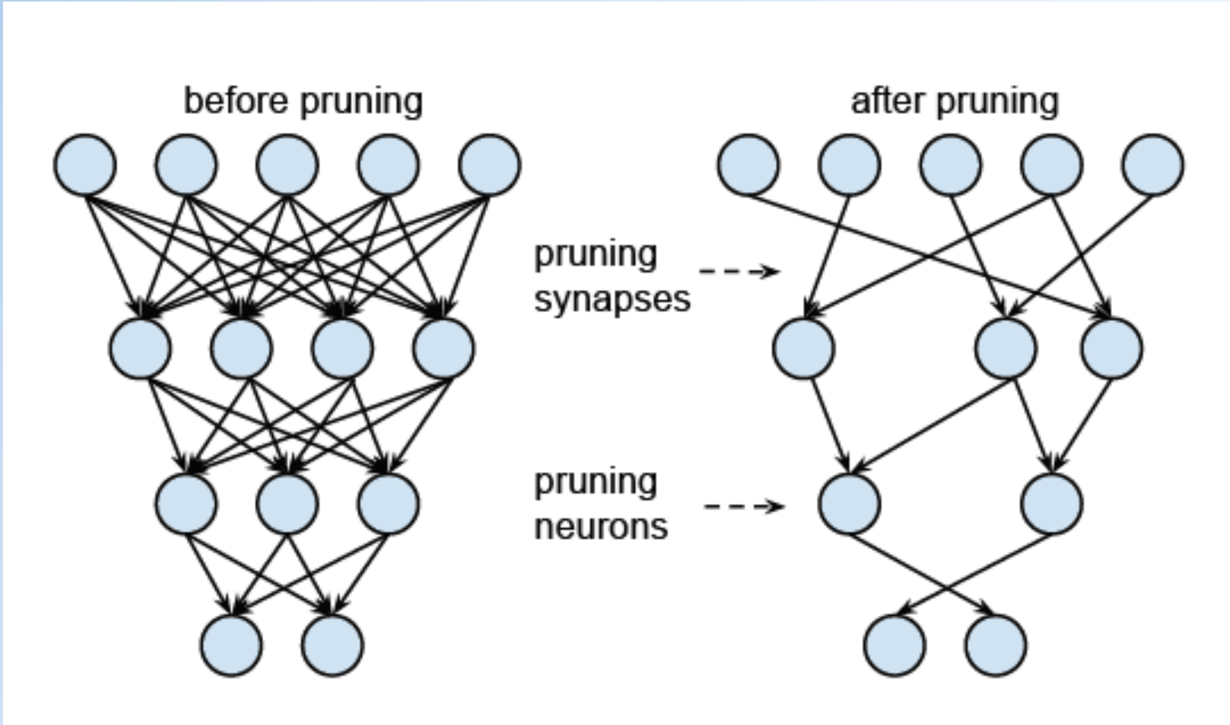
Quantization (low-bits networks)

1. Constrain the weights and activation to be +1 or -1, the evaluate the gradient.
2. Easy for storage
3. Easy for hardware implementation

Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. JMLR 2017
Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, Yoshua Bengio

Pruning Weights





[Learning both Weights and Connections for Efficient Neural Networks]
Song Han, Jeff Pool, John Tran, William J. Dally *Advances in Neural Information Processing Systems (NIPS), December 2015*

Table 1: Network pruning can save 9× to 13× parameters with no drop in predictive performance.

Network	Top-1 Error	Top-5 Error	Parameters	Compression Rate
LeNet-300-100 Ref	1.64%	-	267K	
LeNet-300-100 Pruned	1.59%	-	**22K**	**12**×
LeNet-5 Ref	0.80%	-	431K	
LeNet-5 Pruned	0.77%	-	**36K**	**12**×
AlexNet Ref	42.78%	19.73%	61M	
AlexNet Pruned	42.77%	19.67%	**6.7M**	**9**×
VGG-16 Ref	31.50%	11.32%	138M	
VGG-16 Pruned	31.34%	10.88%	**10.3M**	**13**×

Pruning Filters

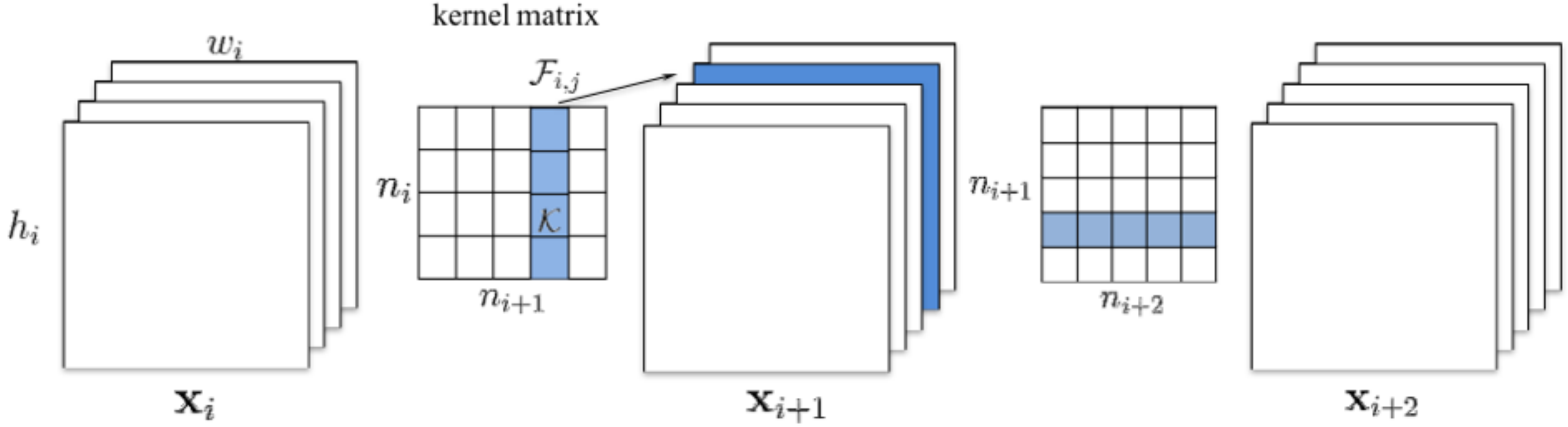


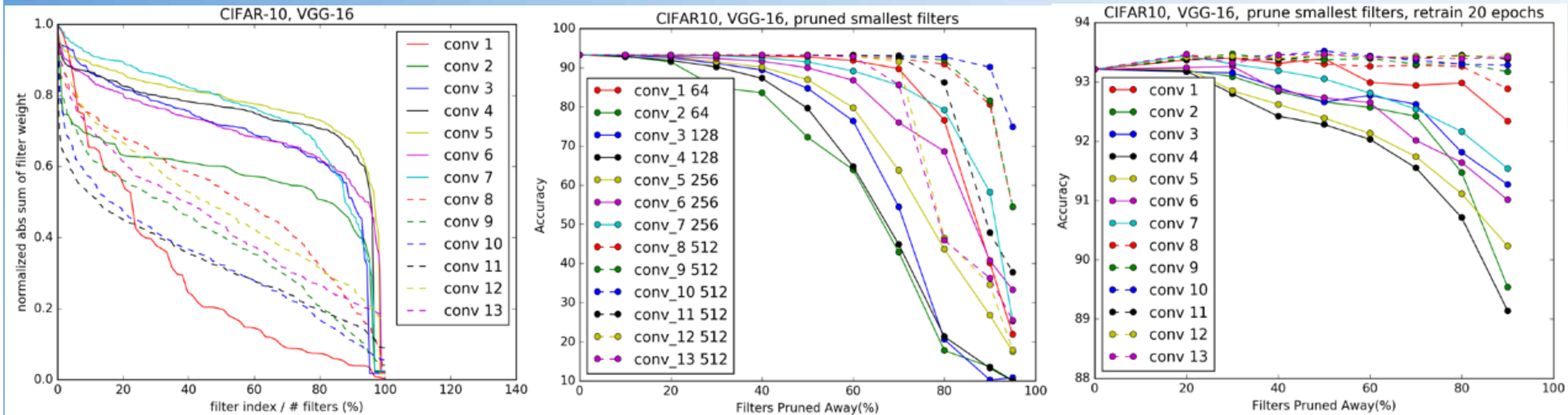
Figure 1: Pruning a filter results in removal of its corresponding feature map and related kernels in the next layer.

Li, Hao, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. "Pruning filters for efficient convnets." *arXiv preprint arXiv:1608.08710* (2017).

Pruning procedure:

1. For each filter $\mathcal{F}_{i,j}$, calculate the sum of its absolute kernel weights $s_j = \sum_{l=1}^{n_i} \sum |\mathcal{K}_l|$.
2. Sort the filters by s_j.
3. Prune m filters with the smallest sum values and their corresponding feature maps. The kernels in the next convolutional layer corresponding to the pruned feature maps are also removed.
4. A new kernel matrix is created for both the ith and $i + 1$th layers, and the remaining kernel weights are copied to the new model.

Approximately performs like Group Lasso



(a) Filters are ranked by s_j (b) Prune the smallest filters (c) Prune and retrain

Distillation (Teacher-student model)

- The idea:

  use a smaller model to mimic the predictions of the original large model

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

Distilled model

$$\frac{\partial C}{\partial z_i} = \frac{1}{T}(q_i - p_i) = \frac{1}{T}\left(\frac{e^{z_i/T}}{\sum_j e^{z_j/T}} - \frac{e^{v_i/T}}{\sum_j e^{v_j/T}}\right)$$

Distilling the Knowledge in a Neural Network

Geoffrey Hinton, Oriol Vinyals, Jeff Dean

Instability of CNNs

Does deep learning really work?

Failure case: the emergence of adversarial examples

- Deep neural networks are easily fooled by adversarial examples!

f(x;w*) → P("panda") = 57.7%

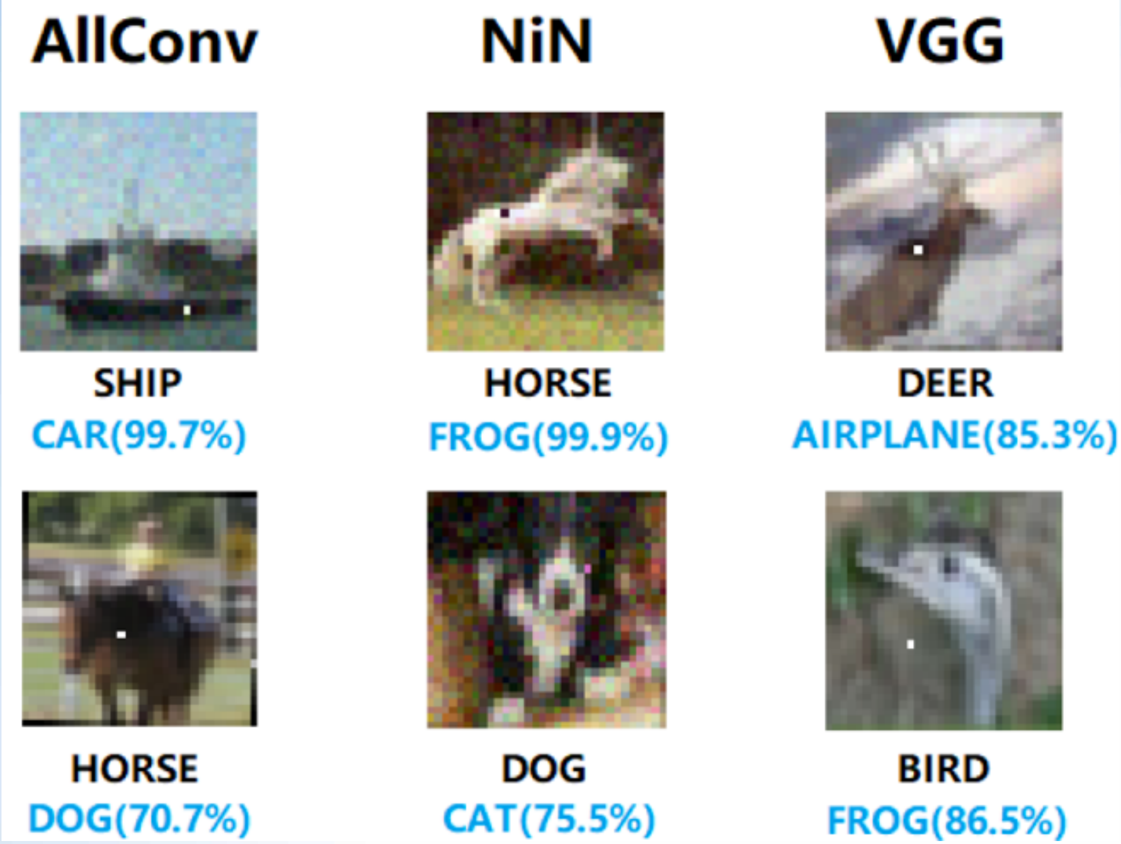
f(x+eta;w*) → P("gorilla") = 99.3% ?!

Uncontrollable Lipschitz constant

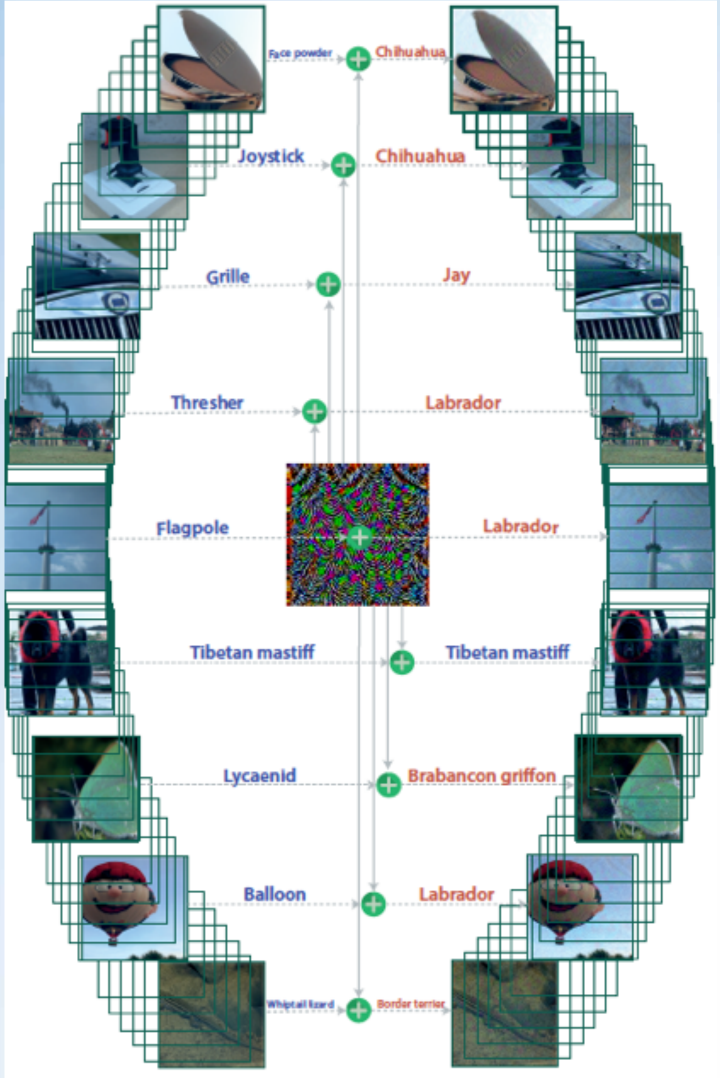
$$\|f(x') - f(x)\| \leq L\|x' - x\|$$

Various Types of Adversarial Attacks

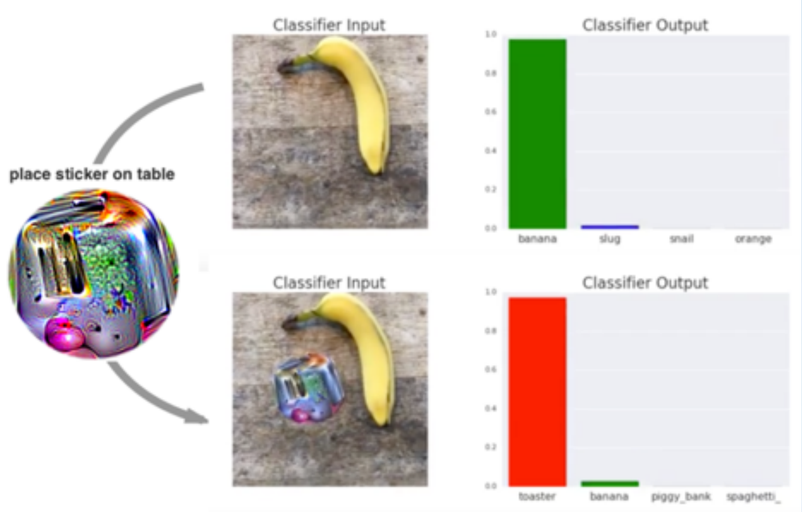
- One-pixel attack (Su et.al 2017)



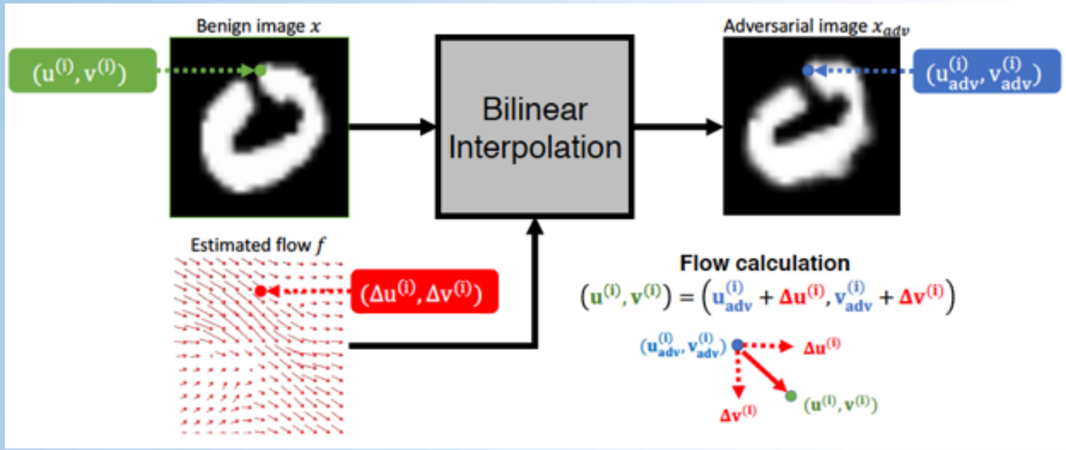
- Universal adversarial perturbation
 (Moosavi-Dezfooli et.al 2017)



- Adversarial Patch (Brown et.al 2017, Thys et.al 2019)

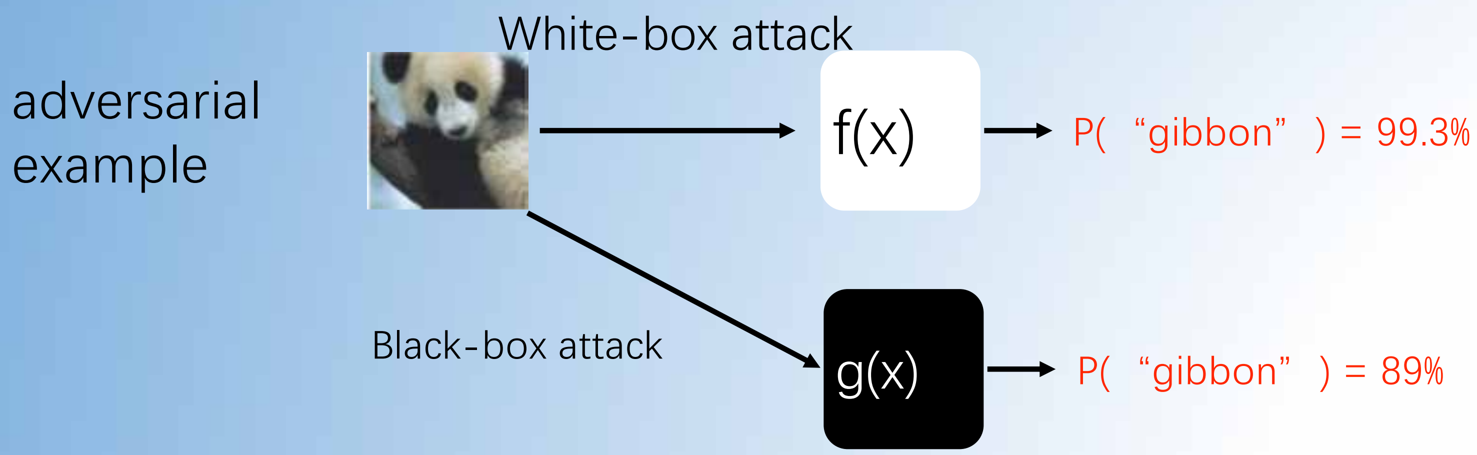


- Spatially transformed attacks (Brown et.al 2017)



More unfortunately, adversarial examples can transfer

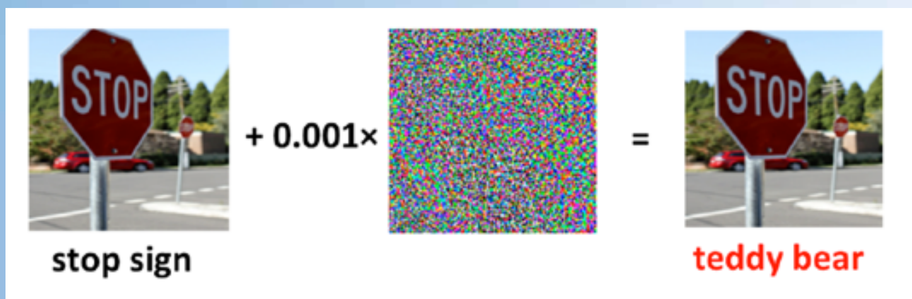
- Adversarial examples constructed based on **f(x)** can also easily fool another network **g(x), even without any queries**



Lei Wu, Zhanxing Zhu, and Cheng Tai. Understanding and Enhancing the Transferability of Adversarial Examples, arXiv-preprint.

Weak Robustness of Current Deep Learning Systems

- Neural networks are fragile, vulnerable, **not robust as expected**
- A large gap between deep networks and human visual systems
- Serious security issues arise when deploying AI systems based on neural networks
  - Autonomous vehicles / medical and health domains



Construct adversarial examples

- An optimization problem

$$\begin{aligned} \text{maximize} \quad & J(f(\boldsymbol{x}+\boldsymbol{\eta}), y^{\text{true}}) \\ \text{s.t.} \quad & \|\boldsymbol{\eta}\| \leq \varepsilon, \end{aligned}$$

  - Fast Gradient Sign Method (FGSM, Goodfellow et.al 2015)

l_∞ norm

$$\boldsymbol{x}^{adv} \leftarrow \boldsymbol{x} + \varepsilon\, g(\boldsymbol{x})$$

$$g^{\infty}(\boldsymbol{x}) = \text{sign}\left(\nabla_{\boldsymbol{x}} J(f(\boldsymbol{x}); y^{\text{true}})\right)$$

→ **white-box attacks**

  - Iterative Gradient Method

$$\boldsymbol{x}^{t+1} \leftarrow \text{clip}^{\boldsymbol{x}^0,\varepsilon}\left(\boldsymbol{x}^t + \alpha\, g(\boldsymbol{x}^t)\right)$$

$$\boldsymbol{x}^t \in [0, 255]^d \cap \{\boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^0\| \leq \varepsilon\}$$

Defense adversarial examples: adversarial training

- Adversarial training (Goodfellow et.al 2014, Madry et.al 2017)

$$\min_{\theta} \max_{\|\eta\| \leq \epsilon} E_{P_{emp}(x)}[J(f(x+\eta;\theta),y)]$$

Generate adv. examples

- Normal training

$$\min_{\theta} E_{P_{emp}(x)}[J(f(x;\theta),y)]$$

Regularization of CNNs

Regularization

- "Modification of initialization, learning and others—aiming to reduce test error."
  - Parameter norm penalty：L1、L2、Elastic net(L1+L2)
  - Early Stopping
  - Data Augmentation
  - Bagging and other ensemble methods
  - Dropout
  - Batch Normalization
  - Initialization

Norm penalty

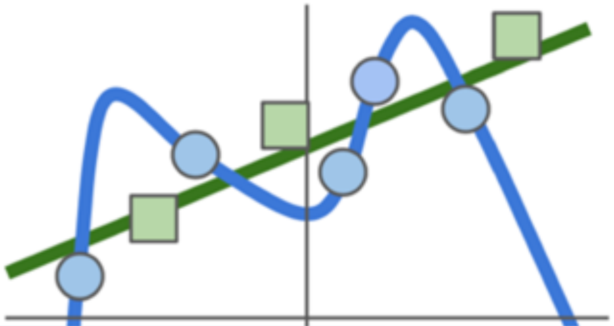
- 模型复杂度与训练误差的均衡

L1 or L2

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

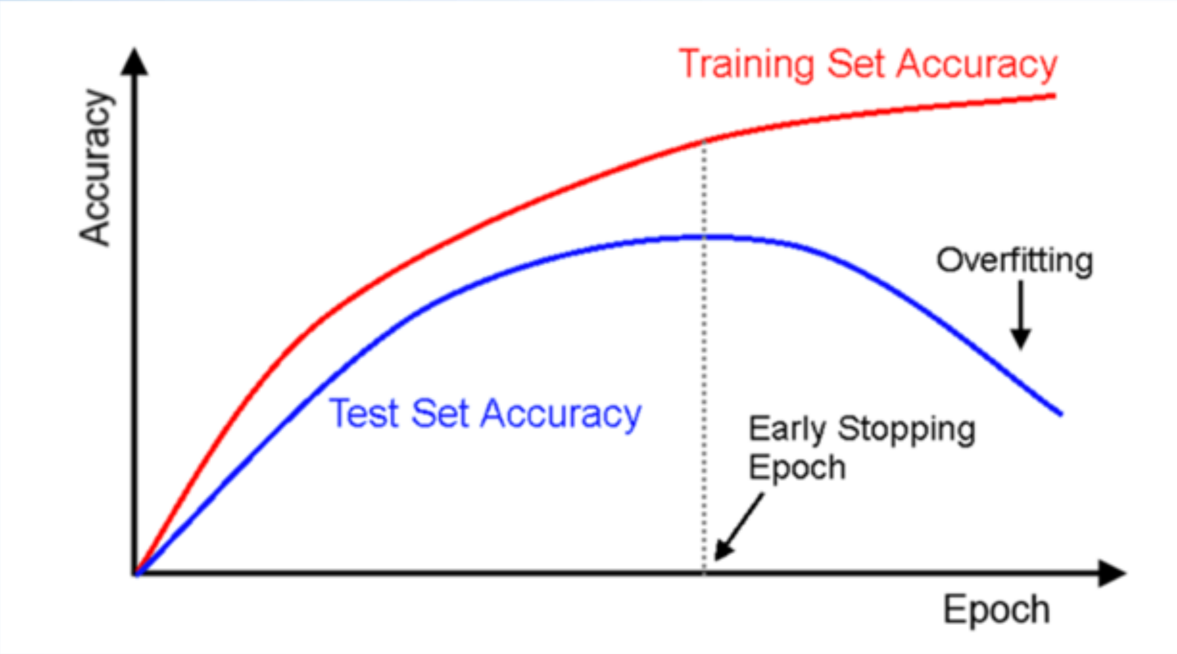
Data loss: Model predictions should match training data

Regularization: Model should be "simple", so it works on test data



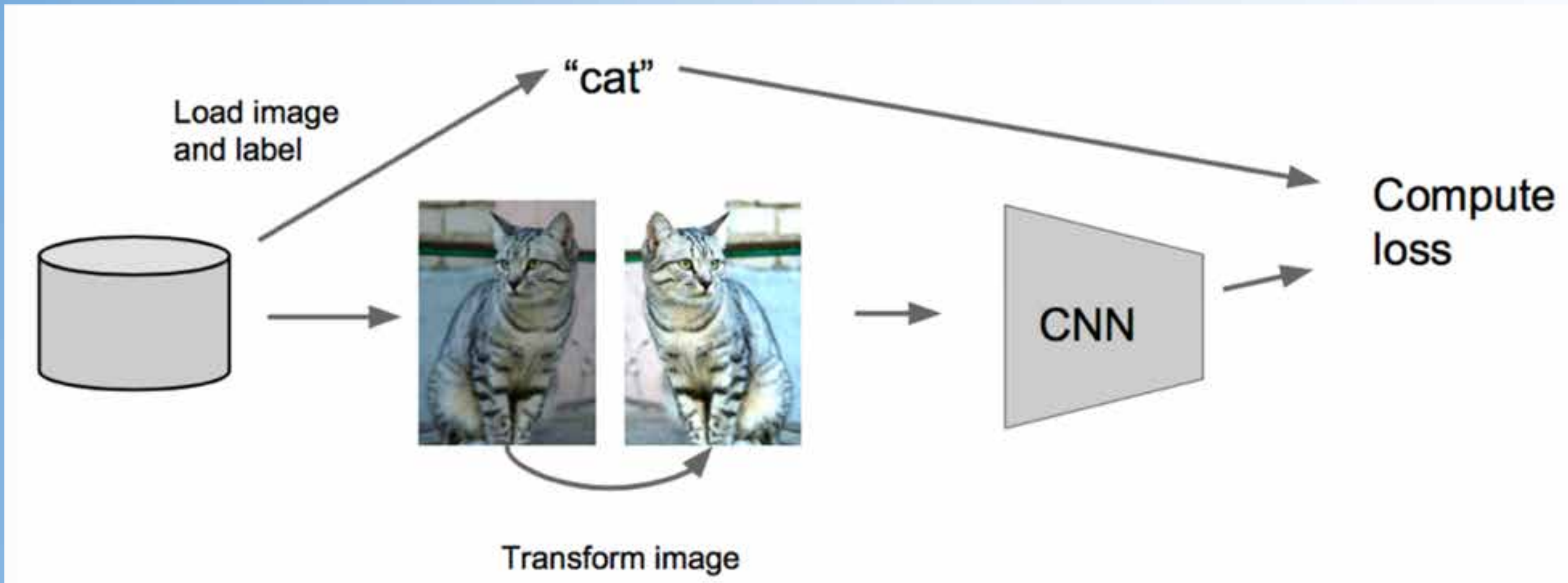
Early stopping

- Over-training makes training error decrease but validation error increase.
- Tune the training iteration/epochs to achieve optimal validation error.



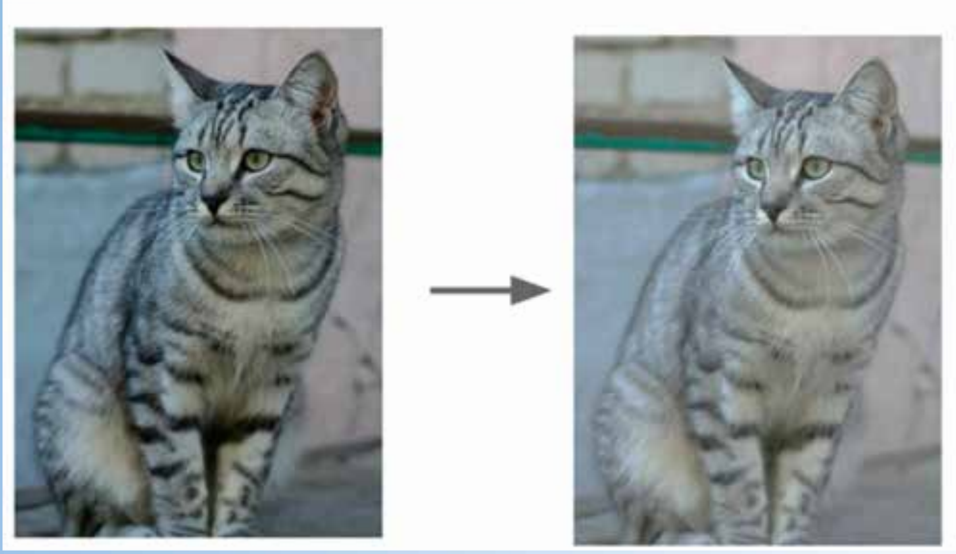
Data augmentation

- Flip, rotation



Data augmentation

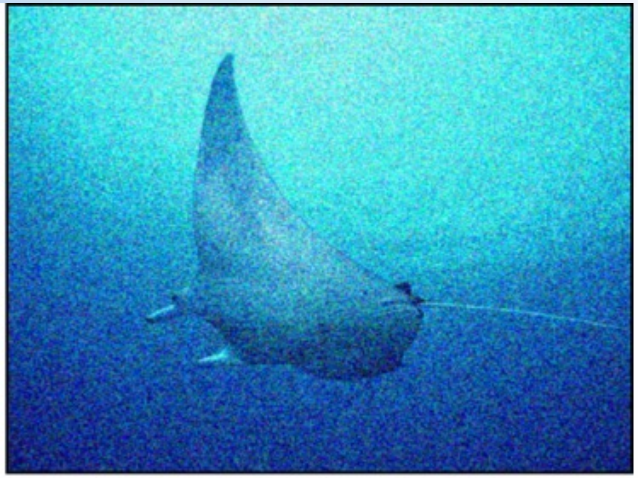
- Random cropping, zooming
- Color jitter (saturation and brightness)



Data augmentation

- Adding random noise

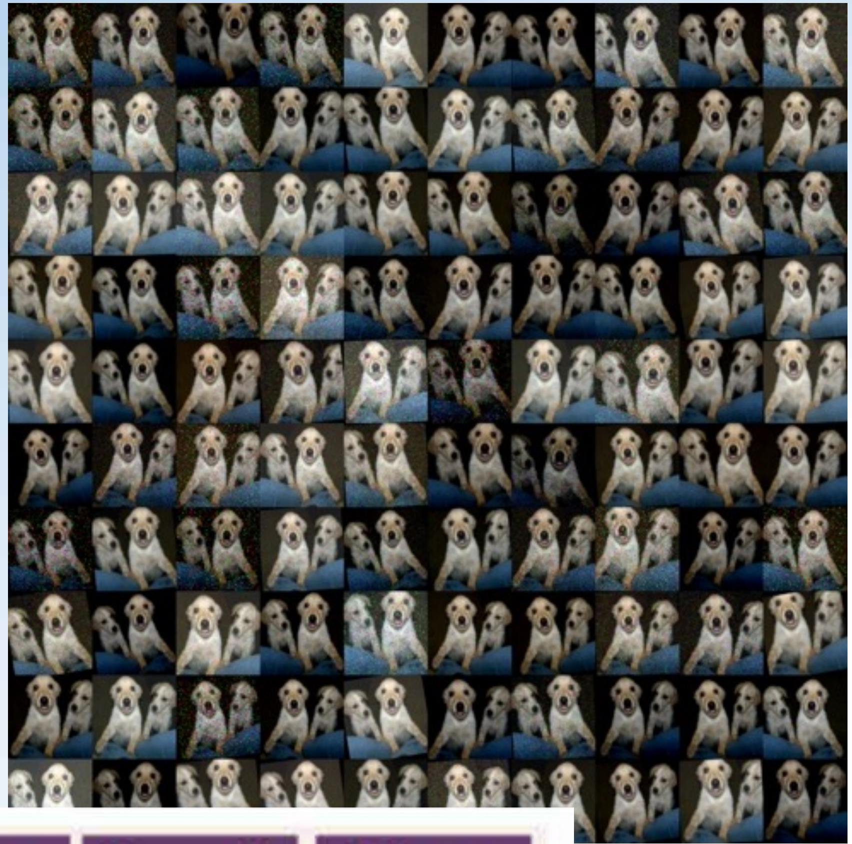
Original image



Add Noise filter applied

Data augmentation

- Combination of various operations
- Or be creative!



Data augmentation

- Improving test performance

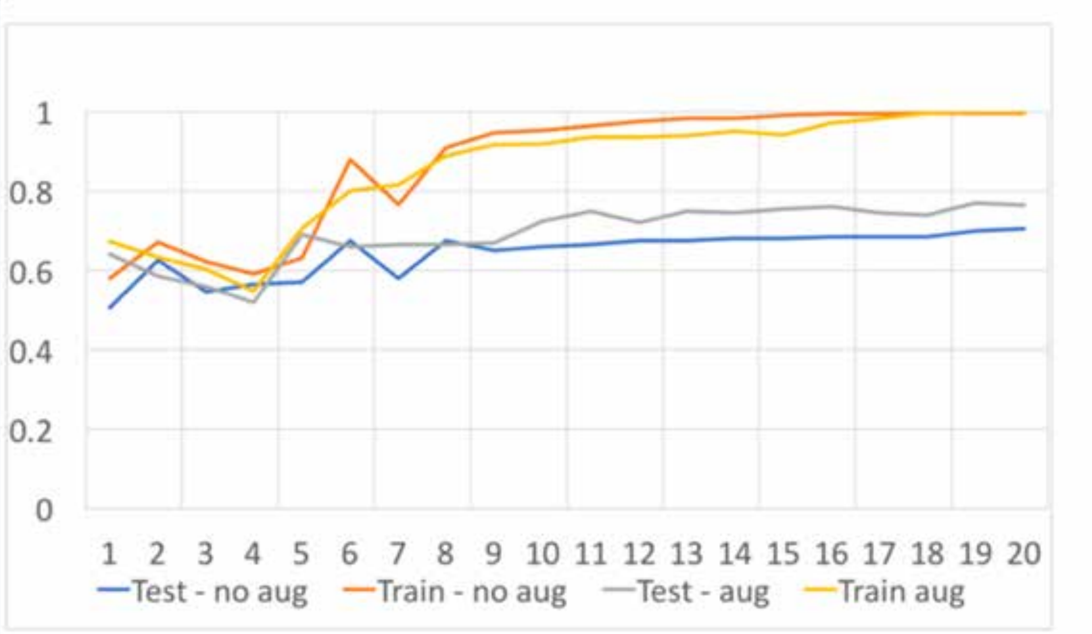


Figure XI: Accuracy plots

Dogs vs Goldfish	
Augmentation	Val. Acc.
None	0.855
Traditional	0.890
GANs	0.865
Neural + No Loss	**0.915**
Neural + Content Loss	0.900
Neural + Style	0.890
Control	0.840

Table I: Quantitative Results on Dogs vs Goldfish

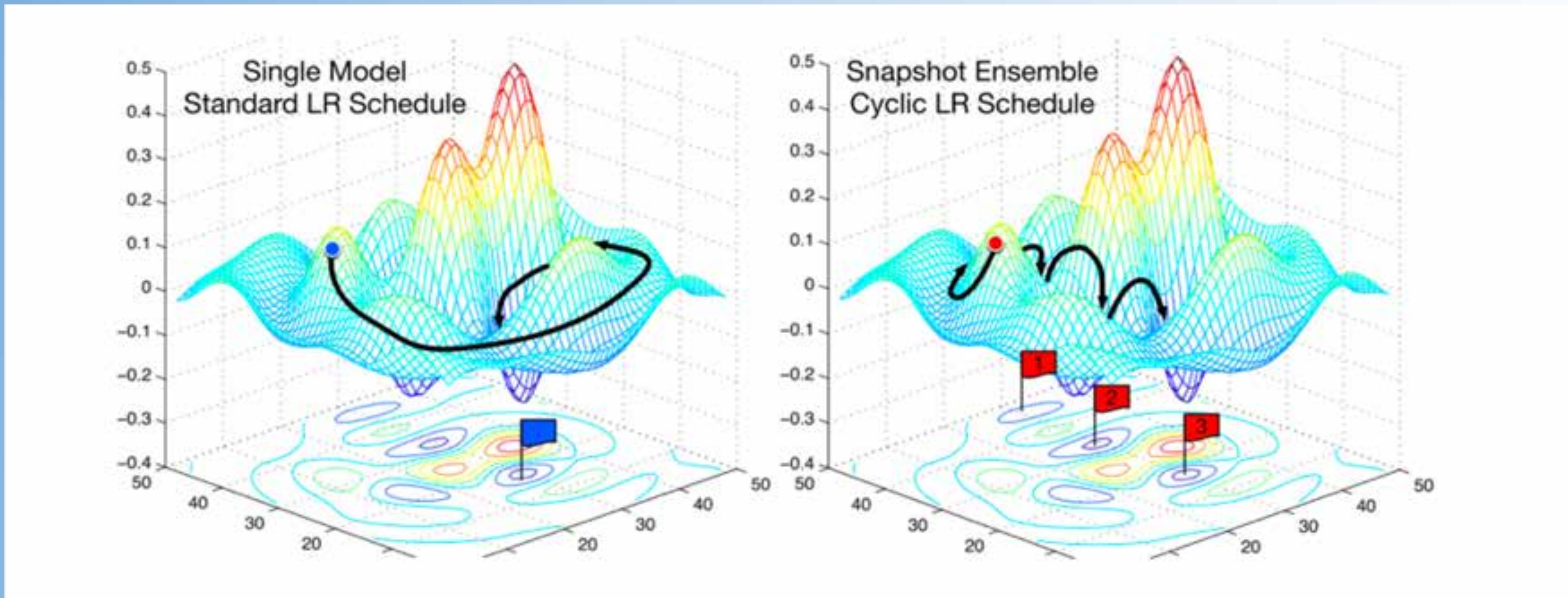
Dogs vs Cat	
Augmentation	Val. Acc.
None	0.705
Traditional	**0.775**
GANs	0.720
Neural + No Loss	0.765
Neural + Content Loss	0.770
Neural + Style	0.740
Control	0.710

Table II: Quantitative Results on Dogs vs Cats

Wang J, Perez L. The Effectiveness of Data Augmentation in Image Classification using Deep Learning[J].

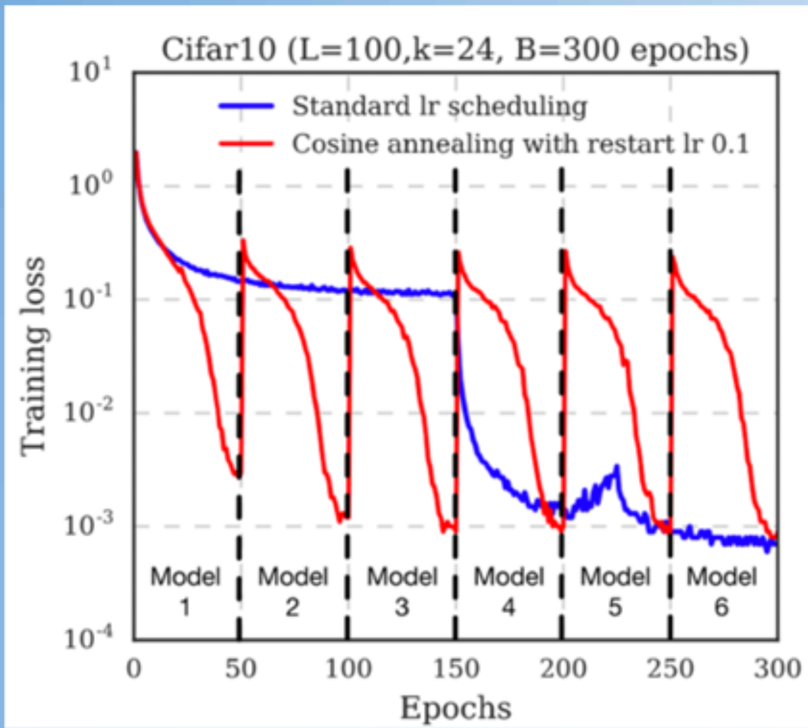
Bagging and other ensemble methods

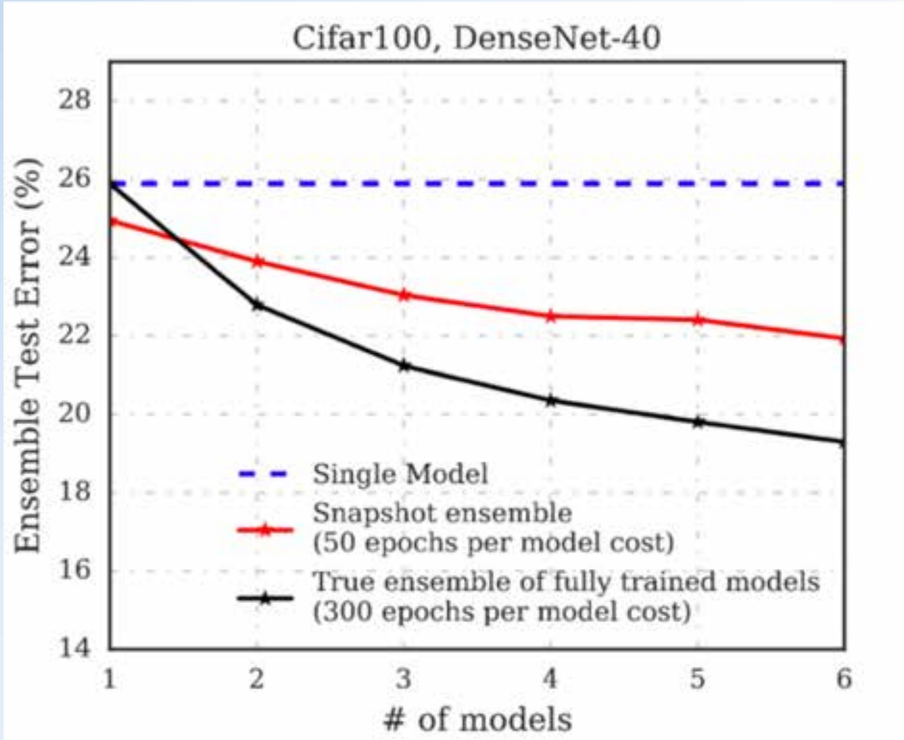
- Train multiple models and average them.



Huang, Gao, et al. "Snapshot ensembles: Train 1, get m for free." *arXiv preprint arXiv:1704.00109* (2017).

- Tricks : Cyclic learning rate schedule

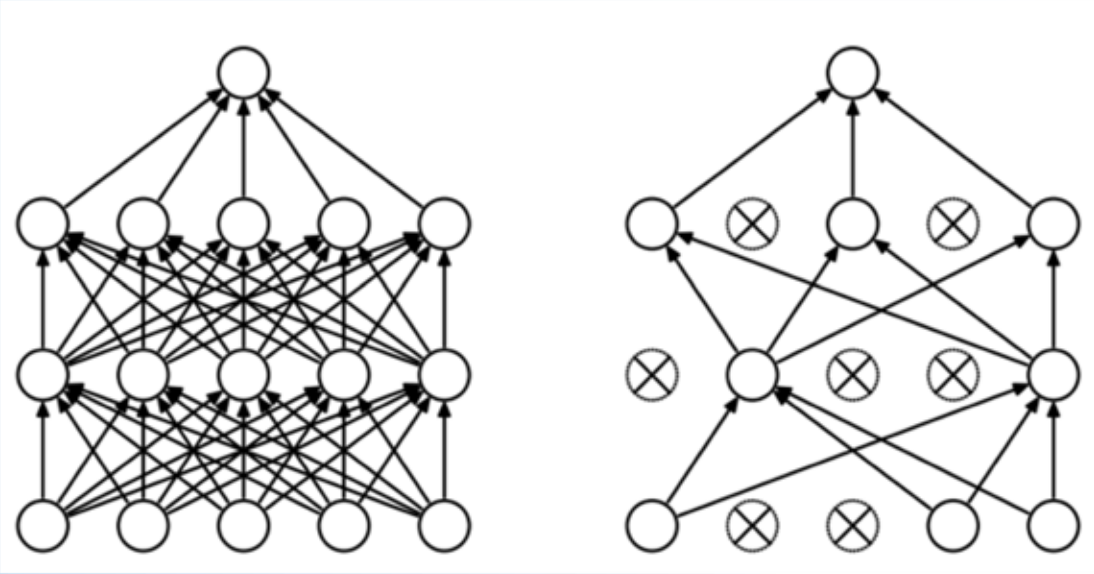




Huang, Gao, et al. "Snapshot ensembles: Train 1, get m for free." *arXiv preprint arXiv:1704.00109* (2017).

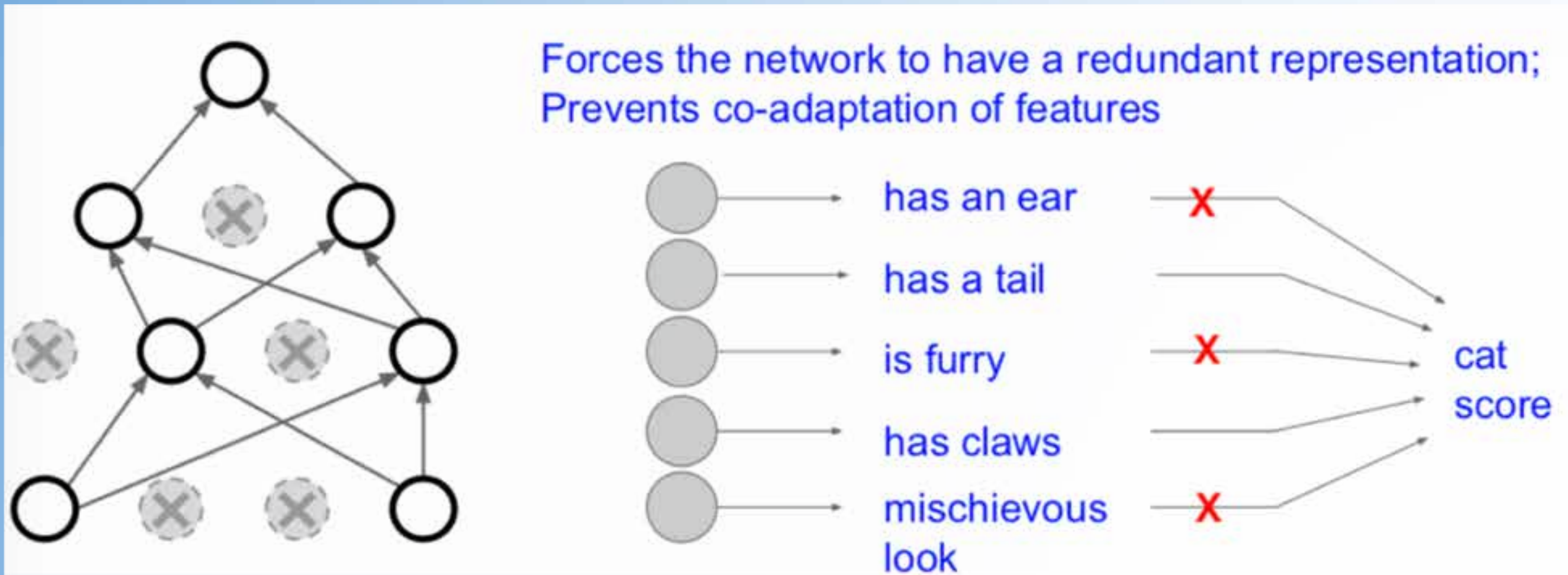
Dropout [Srivastava et al., 2014]

- Randomly drop neurons from original net, and train the subnet
- Drop rate p is hyperparameter , typically p=0.2 for input , p=0.5 for hidden ones



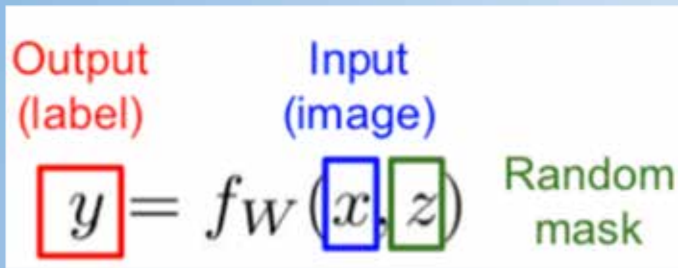
Dropout

- Why does Dropout work?



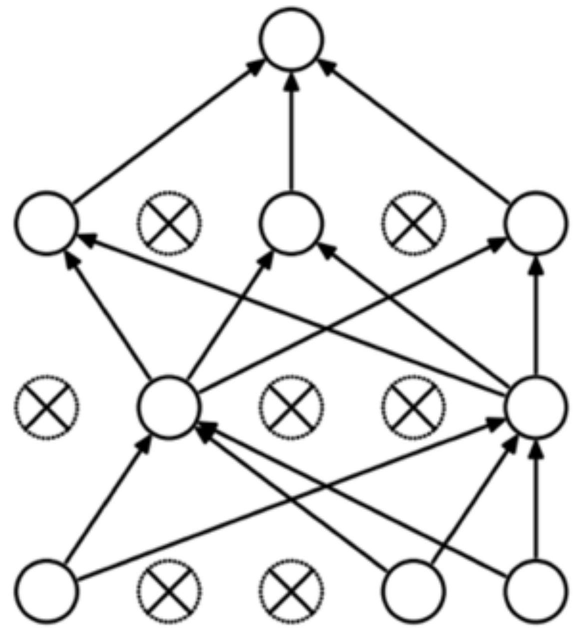
Dropout

- Another interpretation
- Ensemble many subnets: bagging.



Test

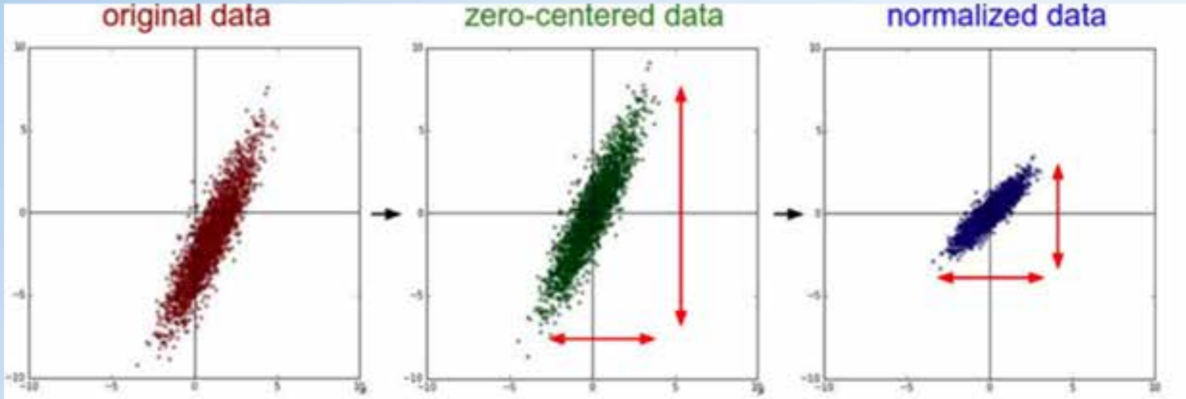
$$y = f(x) = E_z[f(x,z)] = \int p(z) f(x,z) dz$$

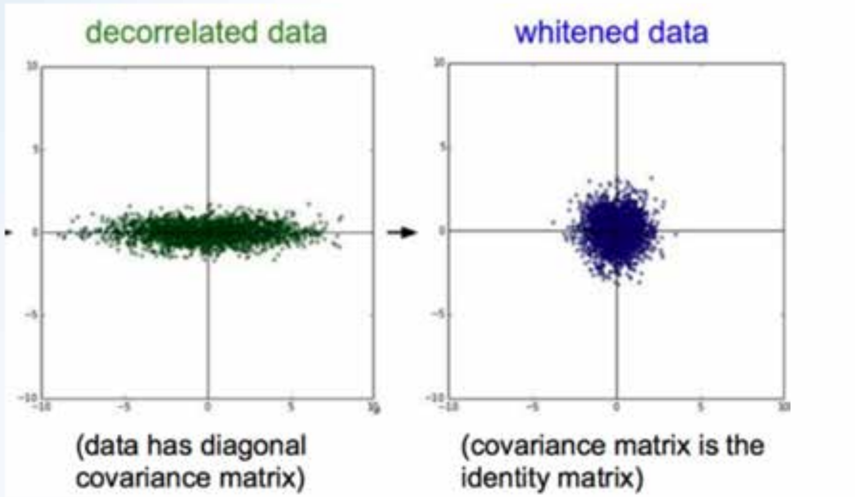


Batch Normalization (BN, Iffoe & Szegedy, 2015)

Adaptive reparameterization: quite effective!

- Covariate shift: input a system changes
- Internal covariate shift in deep networks
  - change of the input distribution in one layer affects next layers dramatically.
- SGD requires careful tuning of hyperparameters
- **Simple version of whitening.**





BN

First simplification over whitening

- normalize each feature independently, not jointly.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

- insert the transformation into the network to guarantee correct representation, not to affect network’ s capacity. (tradeoff)

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

BN

- Normalization inside each mini-batch
- Using mini-batch to estimate mean and variance.
- **BN as a new layer:** typically after convolution or fully connected layer

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

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

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

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

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

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

Input: Network N with trainable parameters Θ; subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$

```
N_BN^tr ← N   // Training BN network
for k = 1 ... K do
    Add transformation y^(k) = BN_{γ^(k),β^(k)}(x^(k)) to N_BN^tr (Alg. 1)
    Modify each layer in N_BN^tr with input x^(k) to take y^(k) instead
end for
Train N_BN^tr to optimize the parameters Θ ∪ {γ^(k), β^(k)}_{k=1}^K
N_BN^inf ← N_BN^tr   // Inference BN network with frozen
                     // parameters
for k = 1 ... K do
    // For clarity, x ≡ x^(k), γ ≡ γ^(k), μ_B ≡ μ_B^(k), etc.
    Process multiple training mini-batches B, each of size m, and average over them:
```

$$\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$

$$\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

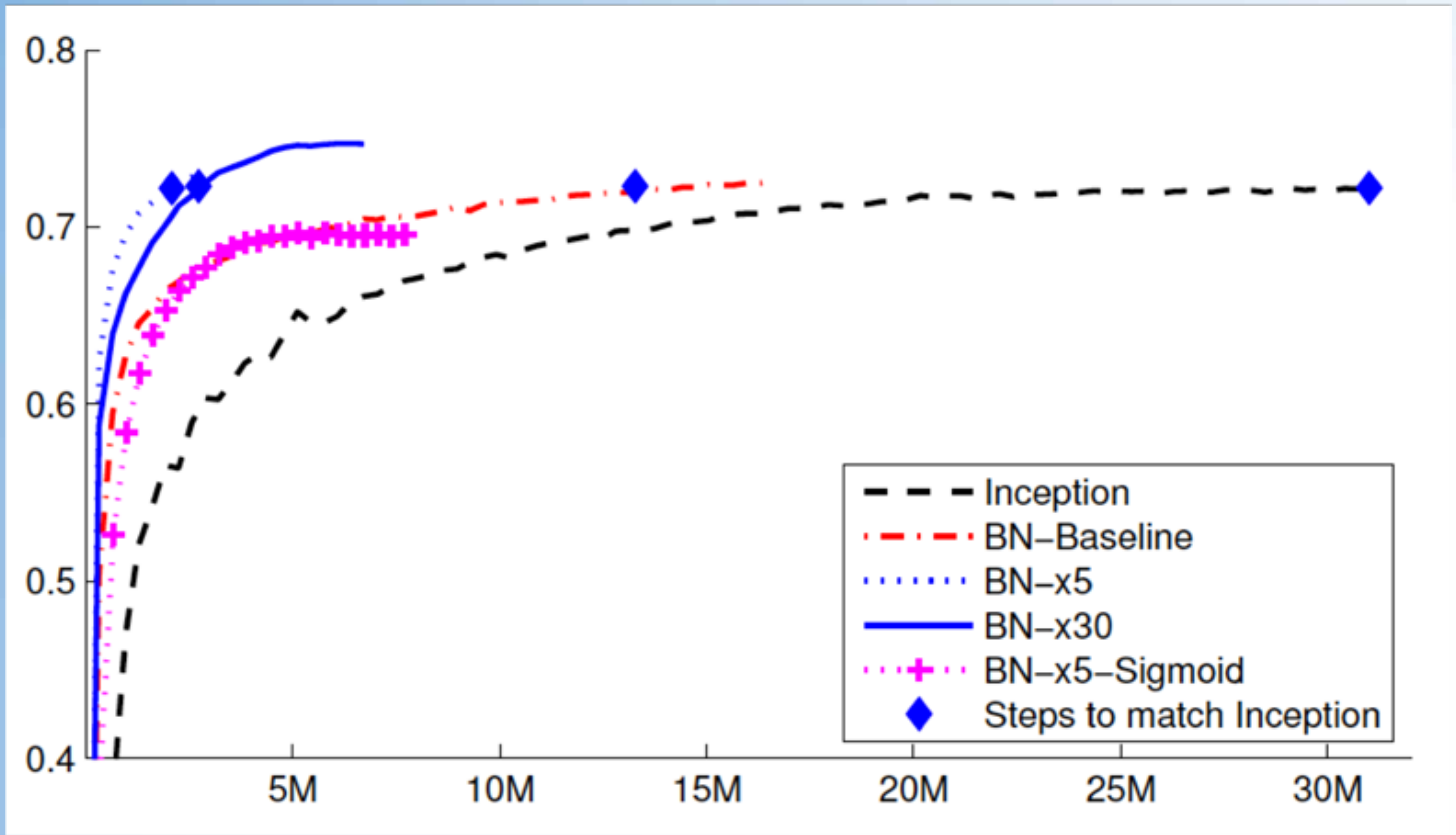
Population statistics

```
    In N_BN^inf, replace the transform y = BN_{γ,β}(x) with
```

$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$$

```
end for
```

BN allows larger learning rates



Summary

- CNN architecture
- Applications
- Visualization of CNNs
- Model compression
- Instability of CNN
- Regularization techniques for neural networks

Thanks!