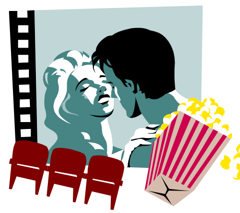
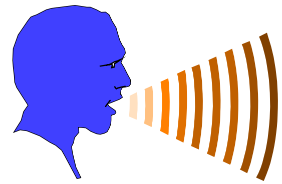
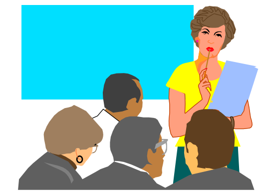





Image and Video Processing

Convolutional Networks for Image Processing (Part II)

Yao Wang
Tandon School of Engineering, New York University
Many contents from Sundeep Rangan:
https://github.com/sdrangan/introml/blob/master/sequence.md

Outline (Part I)

- Supervised learning: General concepts
- Neural network architecture
  - From single perceptron to multi-layer perceptrons
- Convolutional network architecture
  - Why using convolution and many layers
  - Multichannel convolution
  - Pooling
- Deep networks
- Model training
  - Loss functions
  - Stochastic gradient descent: general concept
  - Data Preprocessing and Regularization
- Training, validation and testing and cross validation
- Demo: training a ConvNet classifier

Outline (Part II)

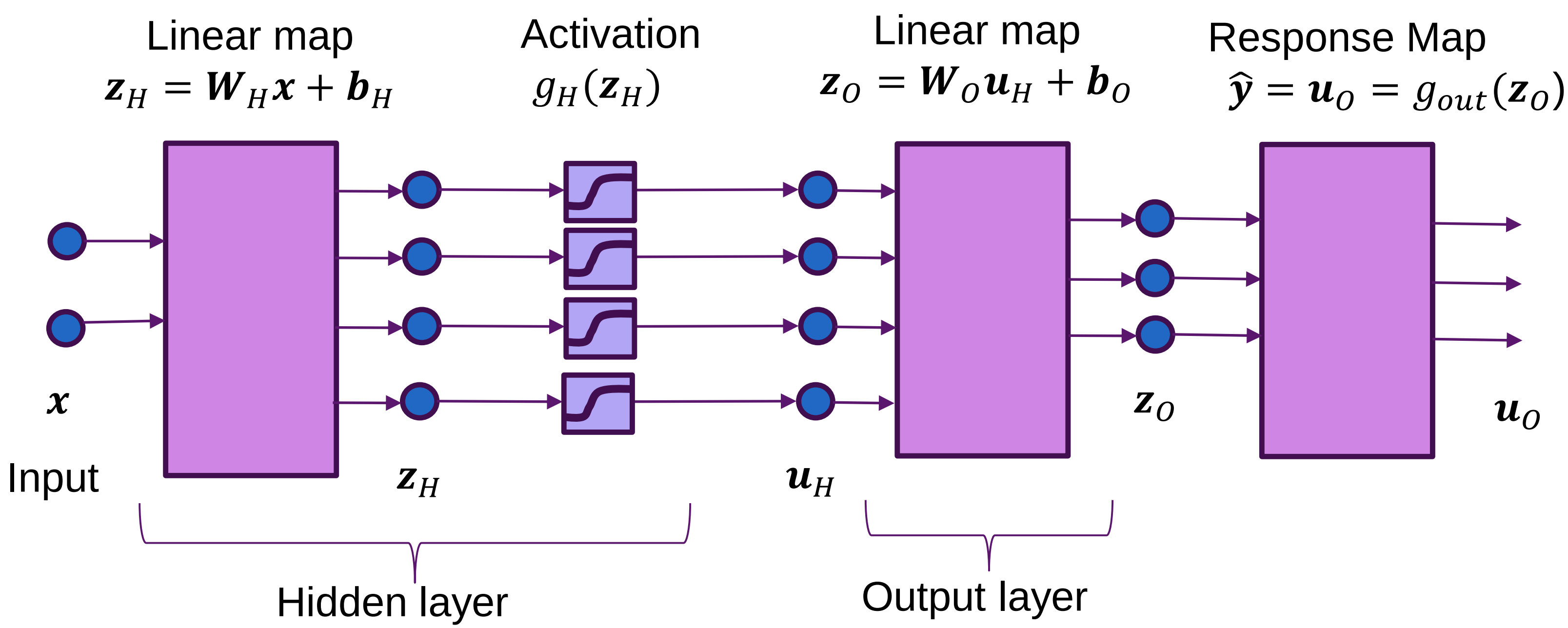
- Neural Nets and Conv Nets and Model Training (Review)
- Gradient calculation
- Some important extensions of conv. layers
- Popular classification models and transfer learning

Outline (Part III)

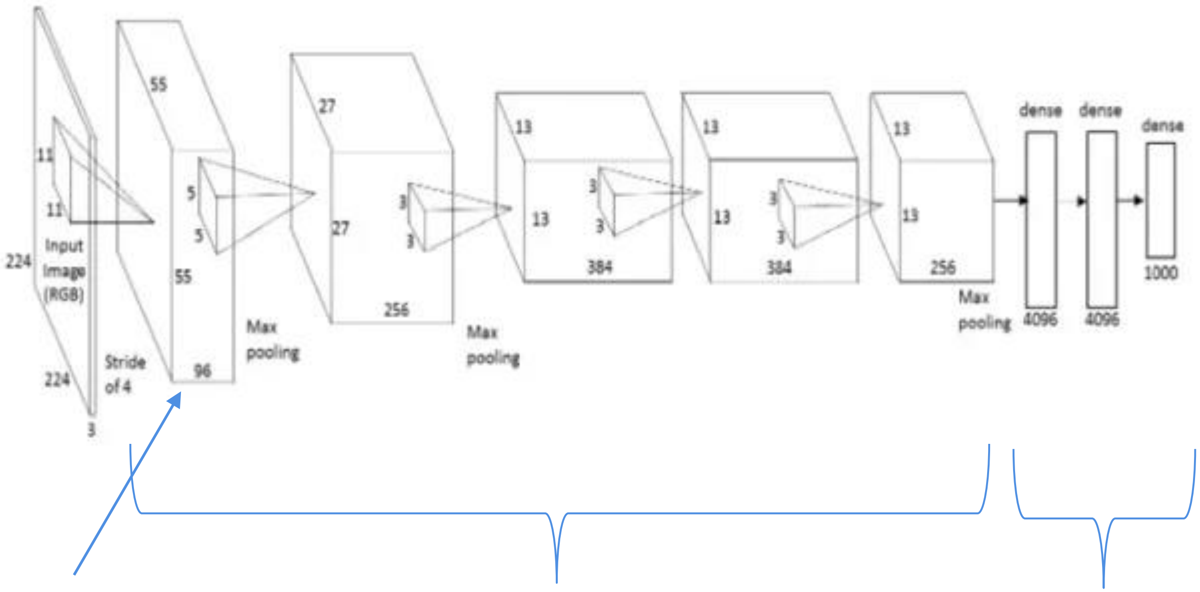
- Image to image autoencoder
- Semantic Segmentation using Multiresolution Autoencoder
- Object detection and classification
- Instance segmentation

Two-Layer Neural Net for Multiple Outputs

- Hidden layer: $\boldsymbol{z}_H = \boldsymbol{W}_H \boldsymbol{x} + \boldsymbol{b}_H, \quad \boldsymbol{u}_H = g_{act}(\boldsymbol{z}_H)$
- Output layer: $\boldsymbol{z}_O = \boldsymbol{W}_O \boldsymbol{u}_H + \boldsymbol{b}_O$
- Response map: $\hat{y} = u_O = g_{out}(\boldsymbol{z}_O)$



Example Conv. Network



- Alex Net
- Each convolutional layer has:
  - 2D convolution
  - Activation (eg. ReLU)
  - Pooling or sub-sampling

96 feature maps of size 55x55 each

Convolutional layers
For feature extraction

2D convolution with Activation and pooling / sub-sampling

Fully connected layers
For Classification task

Matrix multiplication & activation

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.

Training with Gradient Descent

- Given training data: $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$
- Learn parameters: $\theta = (W_H, b_H, W_o, b_o)$
  - Weights and biases for hidden and output layers
  - W_H are filter kernels in conv. layer
- Neural network training (like all training): Minimize loss function

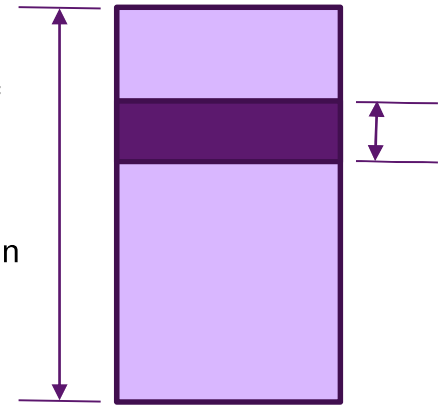
$$\hat{\theta} = \arg\min_{\theta} L(\theta), \qquad L(\theta) = \sum_{i=1}^{N} L_i(\theta, \boldsymbol{x}_i, y_i)$$

  - $L_i(\theta, \boldsymbol{x}_i, y_i)$ = loss on sample i for parameter θ
- Standard gradient descent:

$$\theta^{k+1} = \theta^k - \alpha \nabla L\left(\theta^k\right) = \theta^k - \alpha \sum_{i=1}^{N} \nabla L_i(\theta^k, \boldsymbol{x}_i, y_i)$$

  - Each iteration requires computing N loss functions and gradients
  - But, gradient computation is expensive when data size N large

Stochastic Gradient Descent



- In each step:
  - Select random small "mini-batch"
  - Evaluate gradient on mini-batch

- For $t = 1$ to N_{steps}
  - Select random mini-batch $I \subset \{1, \dots, N\}$
  - Compute gradient approximation: $$g^t = \frac{1}{|I|} \sum_{i \in I} \nabla L(x_i, y_i, \theta)$$
  - Update parameters: $\theta^{t+1} = \theta^t - \alpha^t g^t$

Loss Function: Regression

- Regression case:
  - y_i = target variable for sample i
  - Typically continuous valued
- Output layer:
  - $\hat{y}_i = z_{Oi}$ = estimate of y_i
- Loss function: Use L2 loss

$$L(\theta) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

- For vector $\boldsymbol{y}_i = (y_{i1}, \dots, y_{iK})$, use vector L2 loss

$$L(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{K} \left(y_{ik} - \hat{y}_{i,k}\right)^2$$

Loss Function: Binary Classification

- Binary classification:
  - Sample: x_i with label $y_i = \{0,1\}$ = class label,
  - Predicted output: $\hat{y}_i = P(y_i = 1|x_i, \theta)$; $1 - \hat{y}_i = P(y_i = 0|x_i, \theta)$
  - Output given by sigmoid on $z_{O,i}$: $\hat{y}_i = \frac{1}{1+e^{-z_{O,i}}}$

- Objective: maximize the likelihood (probability of y_i given x_i for all samples, assuming independence among samples)
  - $P(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = \prod_{i=1}^{N} P(y_i|x_i, \theta)$

- Maximizing the likelihood = minimizing negative log likelihood:

$$
\begin{aligned}
L(\theta) &= -\sum_{i=1}^{N} \ln P(y_i|x_i, \theta) \\
&= -\sum_{i=1}^{N} \underbrace{y_i \ln \hat{y}_i}_{\text{activate when } y_i=1} + \underbrace{(1 - y_i) \ln (1 - \hat{y}_i)}_{\text{activate when } y_i=0}
\end{aligned}
$$

  - Called the binary cross-entropy

Loss Function: Multi-Class Classification

- Use one-hot-encoding to describe the label y_i

$$y_i = (y_{i1}, \dots, y_{iK}), \quad y_{ik} = \begin{cases} 1 & y_i = k \\ 0 & y_i \neq k \end{cases} \quad k = 1, \dots, K$$

- Output: $\hat{y}_i = (\hat{y}_{i,1}, \dots, \hat{y}_{i,K})$; $\hat{y}_{i,k} = P(y_i = k | x_i, \theta)$
  - Output given by softmax on $z_{O,i}$: $\hat{y}_{i,k} = \frac{e^{z_{O,ik}}}{\sum_\ell e^{z_{O,il}}}$

- Negative log-likelihood given by:

$$L(\theta) = -\sum_i \ln P(y_i = k | x_i, \theta) = -\sum_i \sum_{k=1}^{K} y_{ik} \ln \hat{y}_{i,k}$$

  - Called the categorical cross-entropy

How to compute gradients?

- For two-layer neural net: $\theta = (W_H, b_H, W_o, b_o)$
- Gradient is computed with respect to each parameter in each batch of M samples:

$$L(\theta) = \sum_{i=1}^{M} L_i(\theta, \boldsymbol{x}_i, y_i) \qquad \nabla L(\theta) = \sum_{i=1}^{M} \nabla L_i(\theta, \boldsymbol{x}_i, y_i)$$

$$\nabla L_i(\theta) = [\nabla_{W_H} L_i(\theta), \nabla_{b_H} L_i(\theta), \nabla_{W_O} L_i(\theta), \nabla_{b_O} L_i(\theta)]$$

- Gradient descent is performed on each parameter:

$$W_H \leftarrow W_H - \alpha \nabla_{W_H} L(\theta),$$
$$b_H \leftarrow b_H - \alpha \nabla_{b_H} L(\theta),$$
$$\ldots.$$

- How to compute $\nabla_{W_H} L_i(\theta), \nabla_{b_H} L_i(\theta)$, etc.?
- W_H, b_H, etc. are vectors and more generally tensors!
- Variables $\boldsymbol{x}_i$, $\boldsymbol{z}_i$, $\boldsymbol{u}_i$, $\hat{\boldsymbol{y}}_i$ are also tensors!

Outline (Part II)

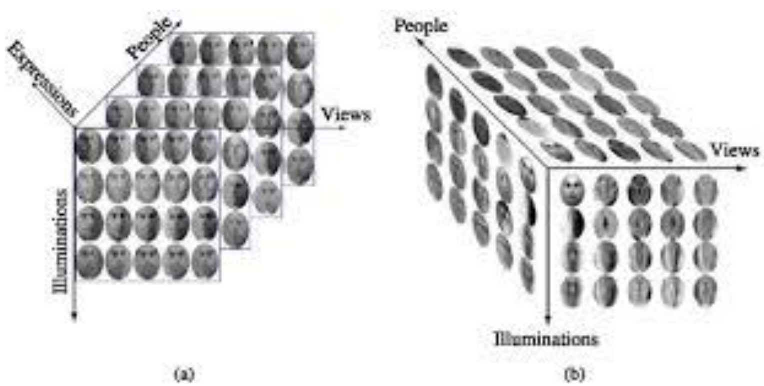
- Neural Nets and Conv Nets and Model Training (Review)
- Gradient calculation
- Some important extensions of conv. layers
- Popular classification models and transfer learning

Gradient Calculation

- Tensor definition
- Tensor gradients
- Tensor gradient chain rule
- Backpropagation
- Forward and backward pass

What is a Tensor?

- A multi-dimensional array
- Examples:
  - 2D: A grayscale image [height x width]
  - 3D: A color image [height x width x rgb]
  - 4D: A collection of images [height x width x rgb x image number]
- Like numpy ndarray
- Basic unit in tensorflow
- Rank or order = Number of dimensions
  - Note: Rank has different meaning in linear algebra



Indexing Tensors

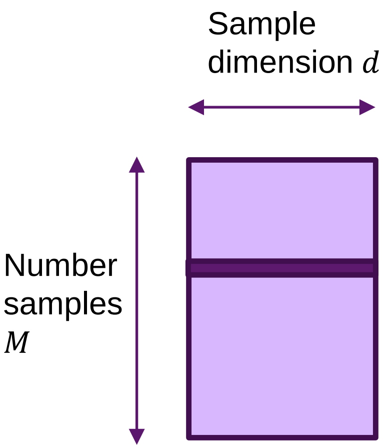
- Suppose $\boldsymbol{X}$ is a tensor of order N
- Index with a multi-index $X[i_1, \dots, i_N]$
  - May also use subscript: $\mathrm{X}_{i_1,\dots,i_N}$

- Example: Suppose $\boldsymbol{X}$ = collection of images [height x width x rgb x image number]
  - $X[100,150,1,30] =$ pixel (100,150) for color channel 1 (green) on image 30

- If $i_1 \in \{0, \dots, d_1 - 1\}, i_2 \in \{0, \dots, d_2 - 1\}, \dots$ then total number of elements = $d_1 d_2 \dots d_N$

Tensors and Neural Networks

- Need to be consistent with indexing
- For a single input x:
  - Input x: vector of dimension d
  - Hidden layer: z_H, u_H: vectors of dimension N_H
  - Outputs: z_O: dimension K
- A batch of inputs with M samples:
  - Input x: Matrix of dimension $M \times d$
  - Hidden layer: z_H, u_H: vectors of dimension $M \times N_H$
  - Outputs: z_O: dimension $M \times K$
- Can generalize to other shapes of input



Gradient for Tensor Inputs & Outputs

- How do we consider general tensor inputs and outputs?
- General setting: $\boldsymbol{y} = f(\boldsymbol{x})$
  - $\boldsymbol{x}$ is a tensor of order N, $\boldsymbol{y}$ is a tensor of order M
- Gradient tensor: A tensor of order $N + M$

$$\left[\frac{\partial f(x)}{\partial x}\right]_{i_1,\ldots i_M, j_1,\ldots,j_N} = \frac{\partial f_{i_1,\ldots,i_M}(x)}{\partial x_{j_1,\ldots,j_N}}$$

  - Tensor has the derivative of every output with respect to every input.
- Ex: $\boldsymbol{x}$ has shape (50,30), $\boldsymbol{y}$ has shape (10,20,40)
  - $\frac{\partial f(x)}{\partial x}$ has shape (10,20,40,50,30)
  - 10(20)(40)(50)(30) = 1.2(10)^7 elements

Gradient Examples 1 and 2

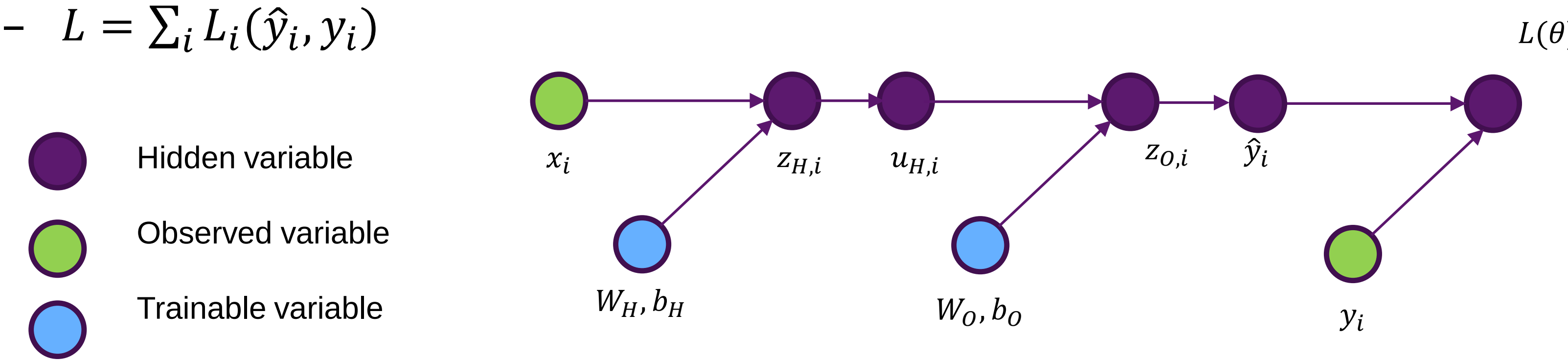
- Example 1: $f(w) = (w_1 w_2, w_1^2 + w_3^3)$
  - 2 outputs, 3 inputs.
  - Gradient tensor is 2×3

$$\frac{\partial f(w)}{\partial w} = \begin{bmatrix} w_2 & w_1 & 0 \\ 2w_1 & 0 & 3w_3^2 \end{bmatrix}$$

- Example 2: $z = f(w) = Aw$, A is $M \times N$
  - M outputs, N inputs: $z_i = \sum_{j=1}^{N} A_{ij} w_j$
  - Gradient components: $\frac{\partial z_i}{\partial w_j} = A_{ij}$

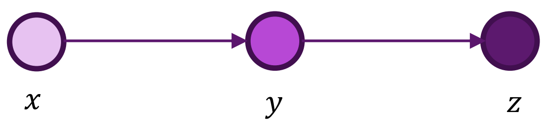
Computation Graph & Forward Pass

- Neural network loss function can be computed via a computation graph
- Sequence of operations starting from measured data and parameters
- Loss function computed via a forward pass in the computation graph
  - $z_{H,i} = W_H x_i + b_H$
  - $u_{H,i} = g_{act}(z_{H,i})$
  - $z_{O,i} = W_O u_{H,i} + b_O$
  - $\hat{y}_i = g_{out}(z_{O,i})$
  - $L = \sum_i L_i(\hat{y}_i, y_i)$



Chain Rule

- How do we compute gradient?
- Consider a three node computation graph:
  - $y = h(x),\ z = g(y)$
  - So $z = f(x) = g(h(x))$
  - What is $\frac{\partial z}{\partial x}$?



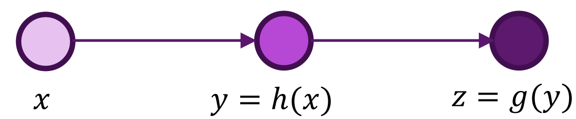
- If variables were scalars, we could compute gradients via chain rule:

$$\frac{\partial z}{\partial x} = \frac{\partial f(x)}{\partial x} = \frac{\partial g(y)}{\partial y}\frac{\partial h(x)}{\partial x}$$

- What happens for tensors?

Tensor Chain Rule

- Consider Tensor case:
  - x has shape $(n_1, \ldots, n_N)$,
  - y has shape $(m_1, \ldots, m_M)$
  - z has shape $(r_1, \ldots, r_R)$



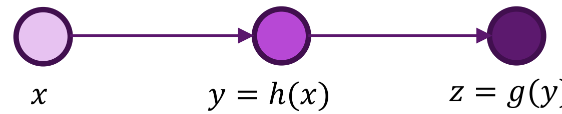
- First, compute gradient tensors between input and output of each node:
  - $\frac{\partial g(y)}{\partial y}$ has shape $(r_1, \ldots, r_R, m_1, \ldots, m_M)$
  - $\frac{\partial h(x)}{\partial x}$ has shape $(m_1, \ldots, m_M, n_1, \ldots, n_N)$
- Next, apply tensor chain rule:

$\frac{\partial z}{\partial x}$ has shape $(r_1, \ldots, r_R, n_1, \ldots, n_N)$: How to compute this?

$$\frac{\partial z_{i_1,\ldots,i_R}}{\partial x_{j_1,\ldots,j_N}} = \frac{\partial f_{i_1,\ldots,i_R}(x)}{\partial x_{j_1,\ldots j_N}} = \underbrace{\sum_{k_1=1}^{m_1} \cdots \sum_{k_M=1}^{m_M}}_{\text{Sum over indices of } \boldsymbol{y}} \frac{\partial g_{i_1,\ldots,i_R}(y)}{\partial y_{k_1,\ldots k_M}} \frac{\partial h_{k_1,\ldots k_M}(x)}{\partial x_{j_1,\ldots j_N}}$$

$$\frac{\partial z}{\partial x} = \frac{\partial f(x)}{\partial x} = \left\langle \frac{\partial g(y)}{\partial y}, \frac{\partial h(x)}{\partial x} \right\rangle$$

Tensor Chain Rule Summary



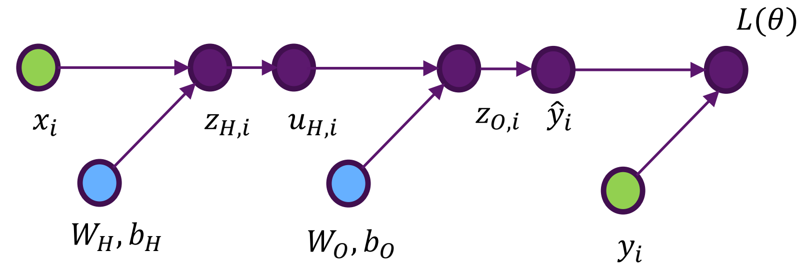
- It is all about keeping track of indices!
- Step 1. Decide on some indexing
  - $x_{j_1,\ldots j_N}$, $y_{k_1,\ldots k_M}$, $z_{i_1,\ldots,i_R}$
- Step 2. Compute all partial derivatives $\frac{\partial g_{i_1,\ldots,i_R}(y)}{\partial y_{k_1,\ldots k_M}}$ and $\frac{\partial h_{k_1,\ldots k_M}(x)}{\partial x_{j_1,\ldots j_N}}$
- Step 3. Use tensor chain rule

$$\frac{\partial z_{i_1,\ldots,i_R}}{\partial x_{j_1,\ldots,j_N}} = \frac{\partial f_{i_1,\ldots,i_R}(x)}{\partial x_{j_1,\ldots j_N}} = \sum_{k_1=1}^{m_1} \cdots \sum_{k_M=1}^{m_M} \frac{\partial g_{i_1,\ldots,i_R}(y)}{\partial y_{k_1,\ldots k_M}} \frac{\partial h_{k_1,\ldots k_M}(x)}{\partial x_{j_1,\ldots j_N}}$$

- Sometimes write with tensor inner product

$$\frac{\partial z}{\partial x} = \frac{\partial f(x)}{\partial x} = \left\langle \frac{\partial g(y)}{\partial y}, \frac{\partial h(x)}{\partial x} \right\rangle$$

Gradients on a Computation Graph



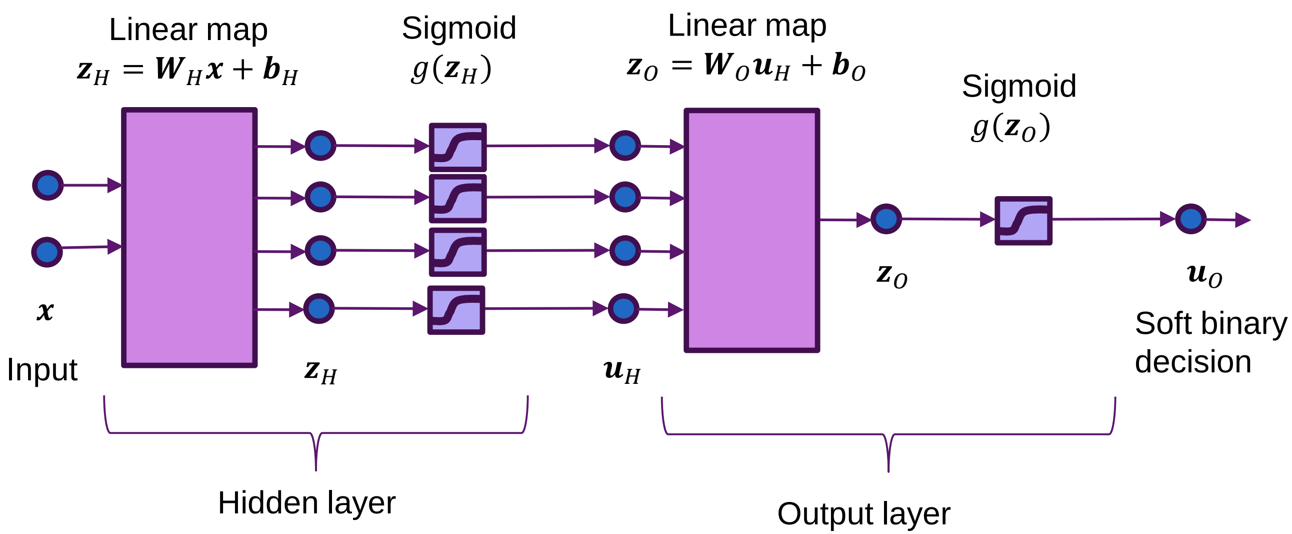
- Backpropagation: Compute gradients backwards
  - Use tensor dot products and chain rule
- First compute all derivatives of all the variables
  - $\partial L/\partial z_O = \langle \partial L/\partial \hat{y}, \partial \hat{y}/\partial z_O \rangle$
  - $\partial L/\partial u_H = \langle \partial L/\partial z_O, \partial z_O/\partial u_H \rangle$
  - $\partial L/\partial z_H = \langle \partial L/\partial u_H, \partial u_H/\partial z_H \rangle$
  - ($\partial \hat{y}/\partial z_O$ and $\partial u_H/\partial z_H$ is element wise)
- Then compute gradient of parameters:
  - $\partial L/\partial W_O = \langle \partial L/\partial z_O, \partial z_O/\partial W_O \rangle$
  - $\partial L/\partial b_O = \langle \partial L/\partial z_O, \partial z_O/\partial b_O \rangle$
  - $\partial L/\partial W_H = \langle \partial L/\partial z_H, \partial z_H/\partial W_H \rangle$
  - $\partial L/\partial b_H = \langle \partial L/\partial z_H, \partial z_H/\partial b_H \rangle$
  -

Example: Last layer of a Binary Classifier

- How to compute $\partial L/\partial W_O, \partial L/\partial b_O$?

$$L(\theta) = -\sum_{i=1}^{N} y_i \ln \hat{y}_i + (1 - y_i) \ln (1 - \hat{y}_i)$$

$$\hat{y}_i = \frac{1}{1+e^{-z_{O,i}}}; \quad \boldsymbol{z}_O = \boldsymbol{W}_O \boldsymbol{u}_H + \boldsymbol{b}_O$$



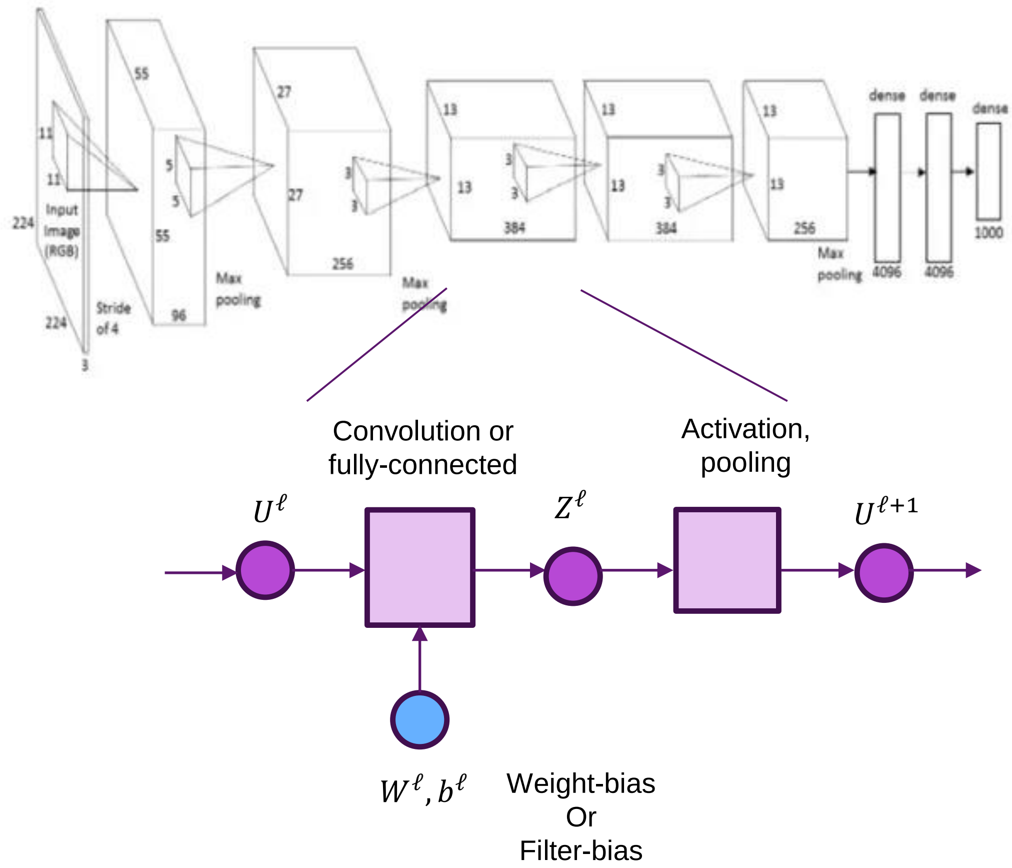
This part could be convolutional layers

Example: Last layer of a Binary Classifier

- Go through on the board

Indexing Multi-Layer Networks

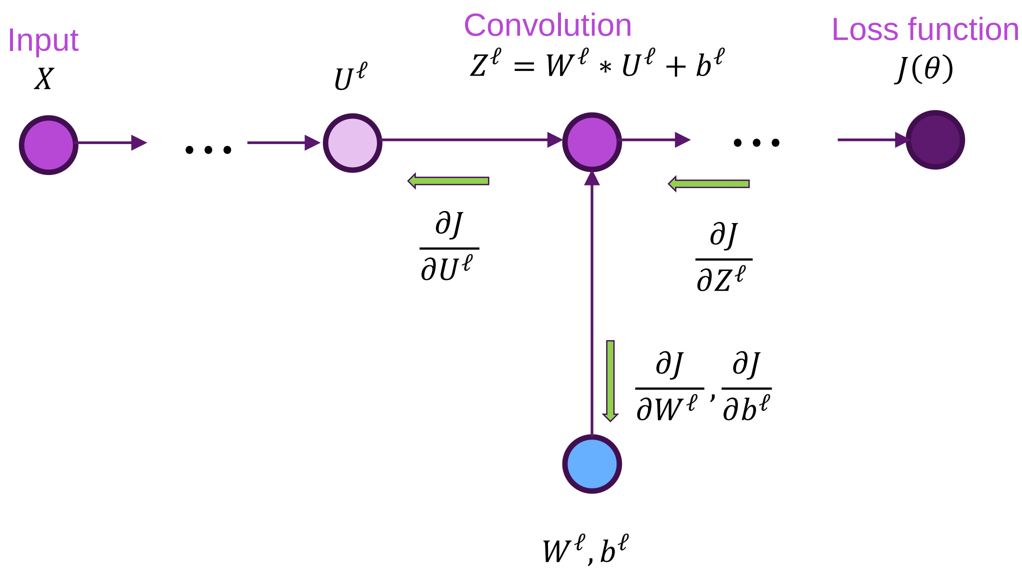
- Similar to two-layer NNs
  - But must keep track of layers
- Consider batch of image inputs:
  - $X[i,j,k,n]$, (sample,row,col,channel)
- Input tensor at layer ℓ:
  - $U^{\ell}[i,j,k,n]$ for convolutional layer
  - $U^{\ell}[i,n]$ for fully connected layer
- Output tensor from linear transform:
  - $Z^{\ell}[i,j,k,n]$ or $Z^{\ell}[i,n]$
- Output tensor after activation / pooling:
  - $U^{\ell+1}[i,j,k,n]$ or $U^{\ell+1}[i,n]$



Back-Propagation in Convolutional Layers

- Convolutional layer in forward path
  $$Z^\ell = W^\ell * U^\ell + b^\ell$$
- During back-propagation:
  - Obtain gradient tensor from upstream layers $\frac{\partial J}{\partial Z^\ell}$
  - Need to compute downstream gradients:

$$\frac{\partial J}{\partial W^\ell}, \quad \frac{\partial J}{\partial b^\ell}, \quad \frac{\partial J}{\partial U^\ell}$$



Gradient With Respect to Filter Weights

- Write convolution as:

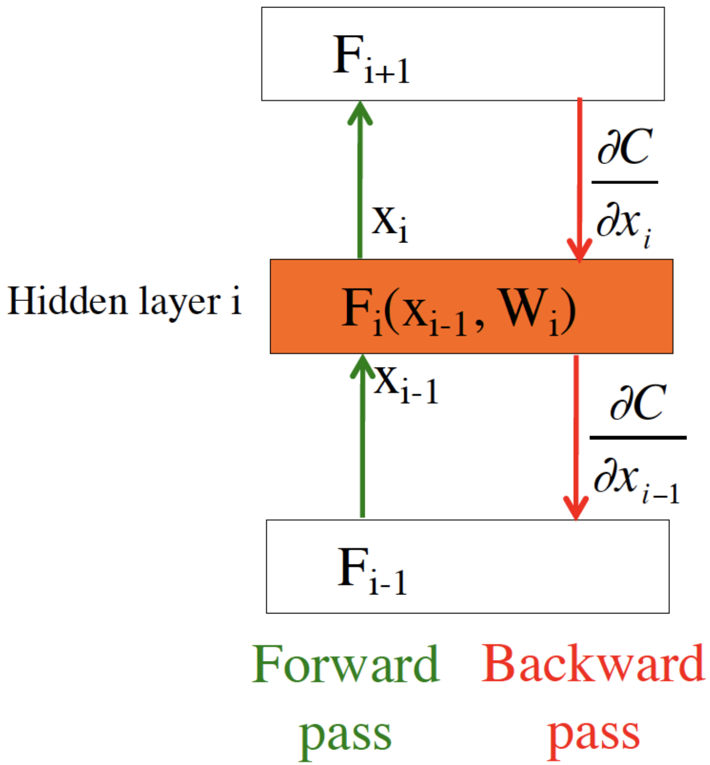
$$Z[i_1, i_2, m] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} \sum_{n=0}^{N_{in}-1} W[k_1, k_2, n, m] U[i_1 + k_1, i_2 + k_2, n] + b[m]$$

  - Drop layer index ℓ and sample index i
- Gradient wrt filter weights: $\frac{\partial Z[i_1,i_2,m]}{\partial W[k_1,k_2,n,m]} = U[i_1 + k_1, i_2 + k_2, n]$
  - *Note that the same filter is used for all pixels, need to sum gradients* $\frac{\partial Z[i_1,i_2,m]}{\partial W[k_1,k_2,n,m]}$ *for all* i_1, i_2:

$$\frac{\partial J}{\partial W[k_1,k_2,n,m]} = \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \frac{\partial Z[i_1,i_2,m]}{\partial W[k_1,k_2,n,m]} \frac{\partial J}{\partial Z[i_1,i_2,m]} = \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} U[i_1 + k_1, i_2 + k_2, n] \frac{\partial J}{\partial Z[i_1,i_2,m]}$$

- Gradient wrt weights can be computed via convolution
  - Convolve input U with gradient tensor $\frac{\partial J}{\partial Z[i_1,i_2,m]}$
- Similar computations for gradients with respect to $\frac{\partial J}{\partial b}$
  - Homework!

Backpropagation: layer i



- Layer i has two inputs (during training)

$$x_{i-1} \qquad \frac{\partial C}{\partial x_i}$$

- For layer i, we need the derivatives:

$$\frac{\partial F_i(x_{i-1}, w_i)}{\partial x_{i-1}} \qquad \frac{\partial F_i(x_{i-1}, w_i)}{\partial w_i}$$

- We compute the outputs

$$x_i = F_i(x_{i-1}, w_i)$$

$$\frac{\partial C}{\partial x_{i-1}} = \frac{\partial C}{\partial x_i} \cdot \frac{\partial F_i(x_{i-1}, w_i)}{\partial x_{i-1}}$$

- The weight update equation is:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial x_i} \cdot \frac{\partial F_i(x_{i-1}, w_i)}{\partial w_i}$$

$$w_i^{k+1} \leftarrow w_i^k + \eta_t \frac{\partial E}{\partial w_i}$$

(sum over all training examples to get E)

From Fergus: https://cs.nyu.edu/~fergus/teaching/vision/2_neural_nets.pdf

Backpropagation: summary

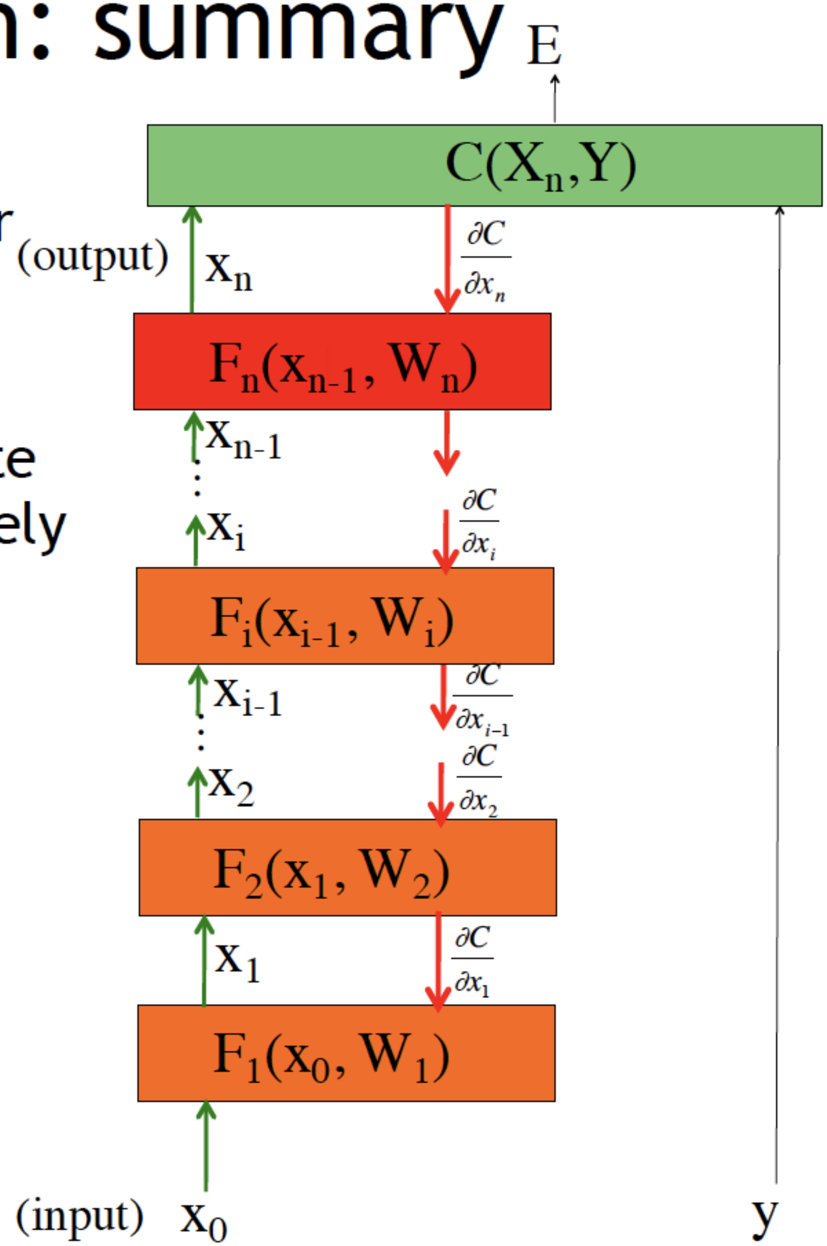
- Forward pass: for each training example. Compute the outputs for all layers

$$x_i = F_i(x_{i-1}, w_i)$$

- Backwards pass: compute cost derivatives iteratively from top to bottom:

$$\frac{\partial C}{\partial x_{i-1}} = \frac{\partial C}{\partial x_i} \cdot \frac{\partial F_i(x_{i-1}, w_i)}{\partial x_{i-1}}$$

- Compute gradients and update weights.



From Fergus: https://cs.nyu.edu/~fergus/teaching/vision/2_neural_nets.pdf

Outline (Part II)

- Neural Nets and Conv Nets and Model Training (Review)
- Gradient calculation
- Some important extensions of conv. layers
- Popular classification models and transfer learning

Residual Connections (ResNET)

- Really, really deep convnets don’t train well
  - Gradient of final loss does not propagate back to earlier layers
- Key idea: introduce “pass through” into each layer for back propagation

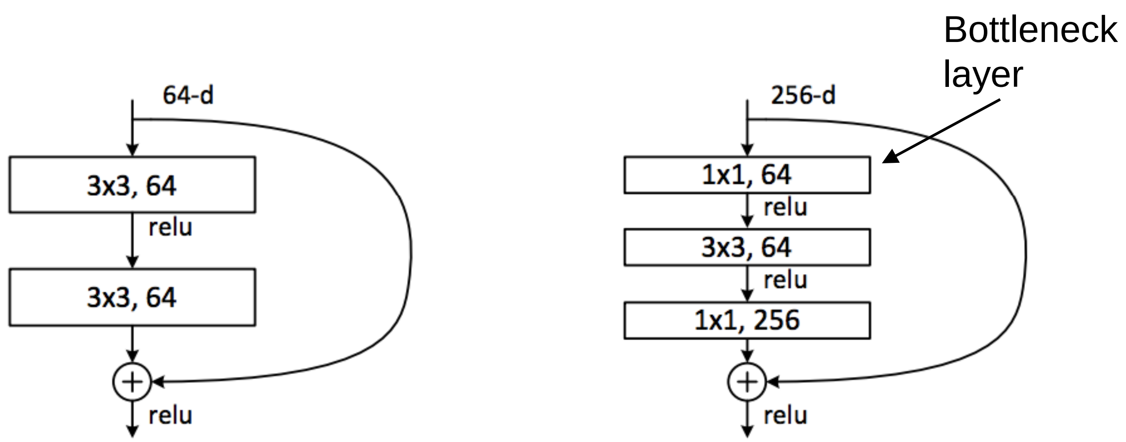
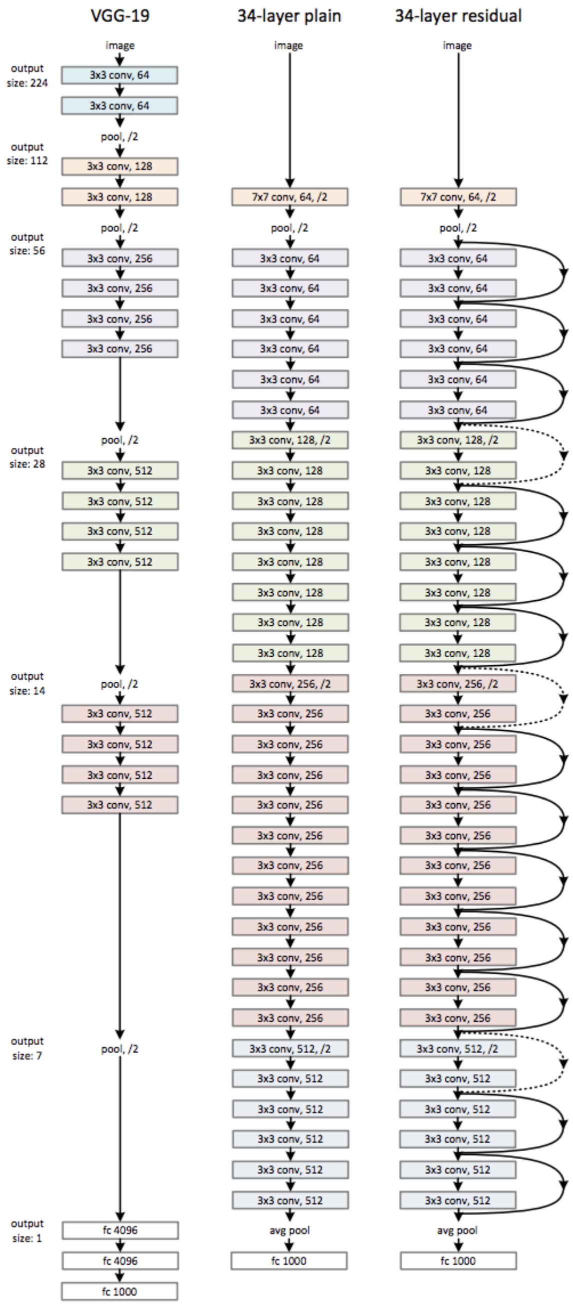


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.

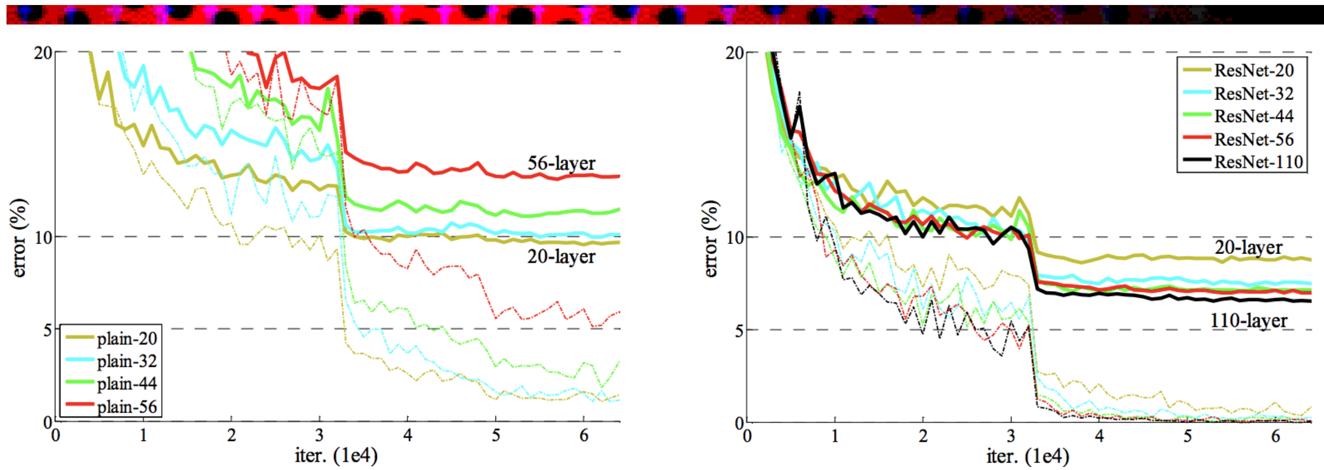
He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016. http://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

Some Important Extensions

- Residual connections
- Dense connections
- Dilated convolution



Benefit of residual connection



W/o residual layer: deeper networks perform worse even for the training data.
W residual layer: deeper networks perform better!

Using shortcut 2 is theoretically optimal
Demystifying ResNet
https://arxiv.org/abs/1611.01186

Revolution of Depth

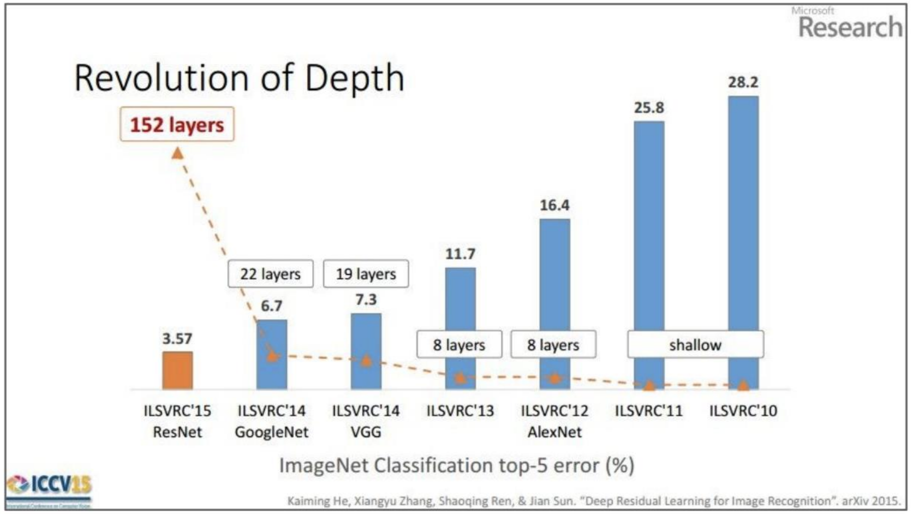
Case Studies

D	E
16 weight layers	19 weight layers
conv3-64 conv3-64	conv3-64 conv3-64
conv3-128 conv3-128	conv3-128 conv3-128
conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool	
FC-4096	
FC-4096	
FC-1000	
soft-max	

VGG (2014)

GoogLeNet (2014)

ResNet (2015)



From: http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf

A variation of residual connection: Concatenation (DenseNet)

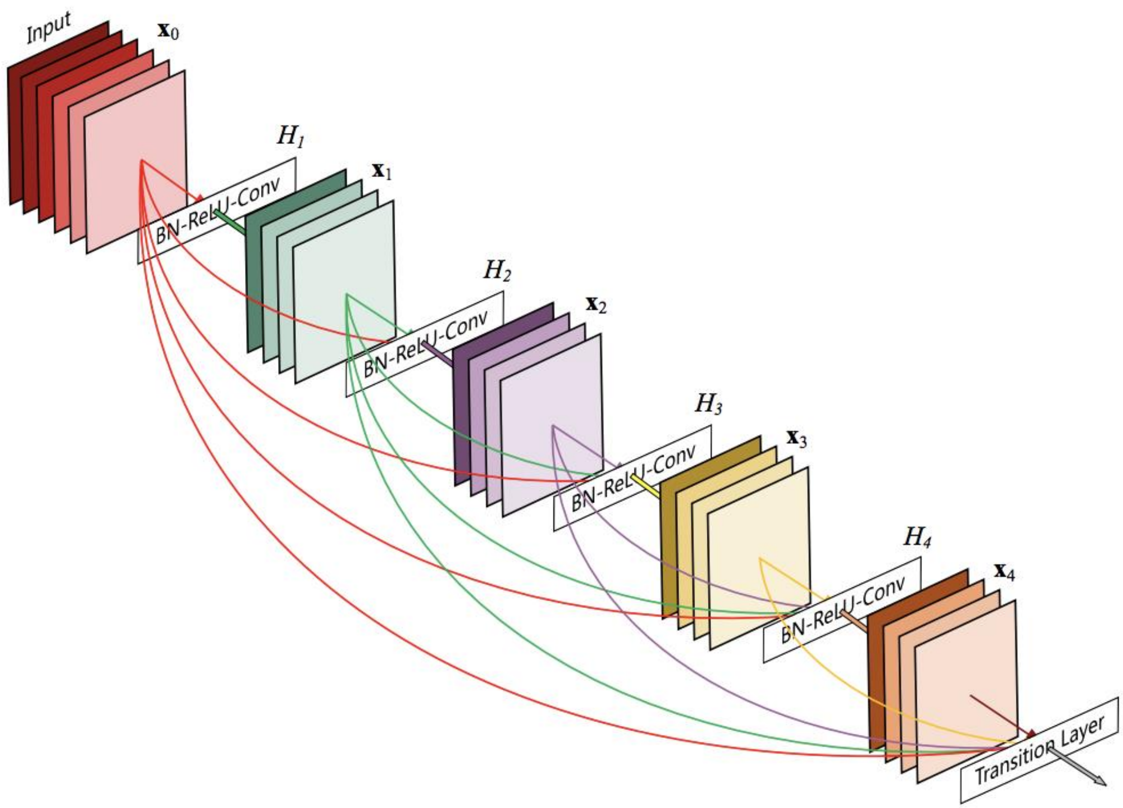


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- Feature maps of all preceding layers are concatenated and used as input for the current layer.
- Facilitate gradient back propagation, as with residual connection
- Strengthen feature forward propagation and reuse

From: Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely connected convolutional networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700-4708. 2017.

Stacking Dense Blocks

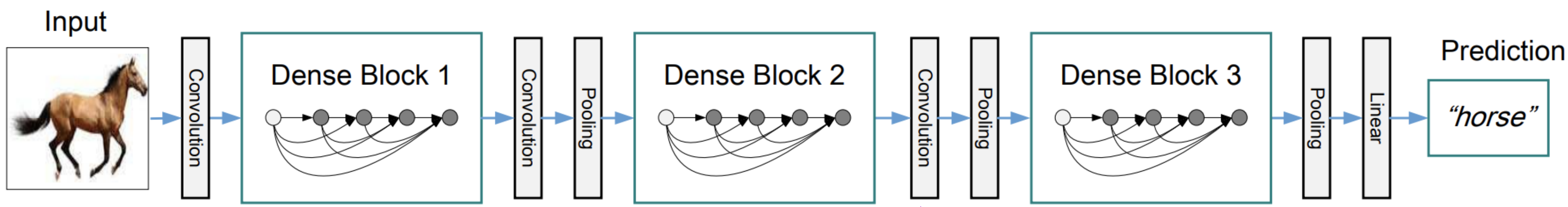
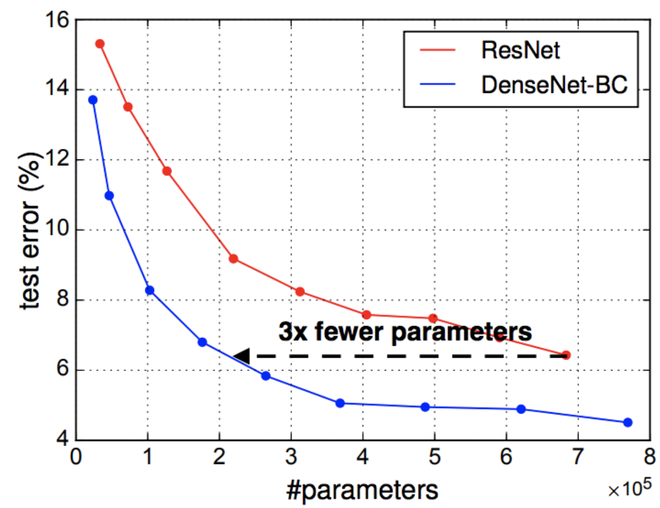


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Use bottleneck layer (1x1 conv) to reduce the number of feature maps between blocks

- Can use fewer layers to achieve same performance as ResNET



From: Huang, Gao, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. "Densely connected convolutional networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700-4708. 2017.

Dilated Convolution

- Large perceptive field is important to incorporate global information
- How to increase the perceptive field
  - Larger filter
  - More layers of small filters
  - Dilated conv.

- Receptive field of the first layer is the filter size
- Receptive field (w.r.t. input image) of a deeper layer depends on all previous layers' filter size and strides

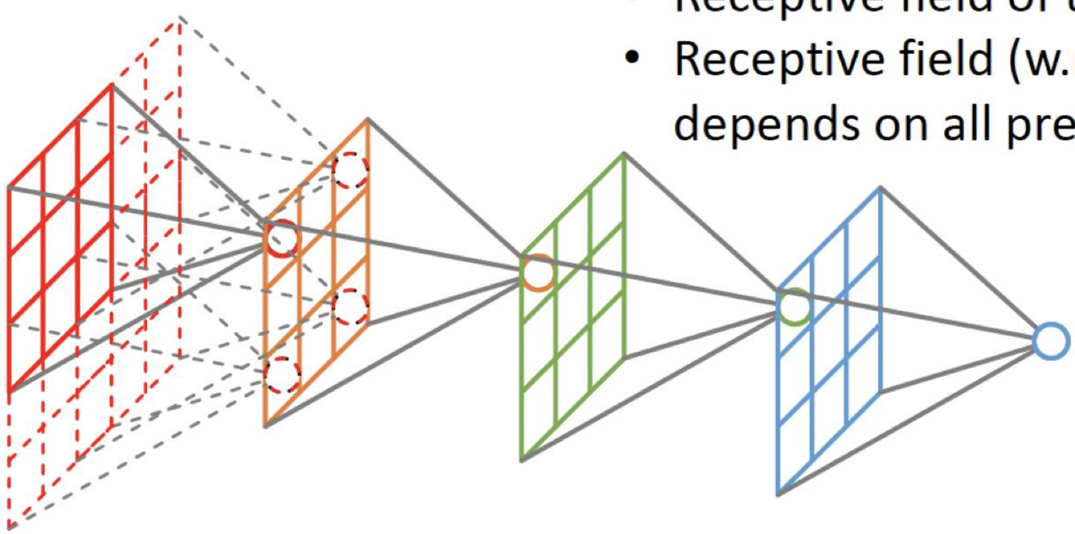


Figure from Fergus: https://cs.nyu.edu/~fergus/teaching/vision/3_convnets.pdf

Dilated Conv in 1D

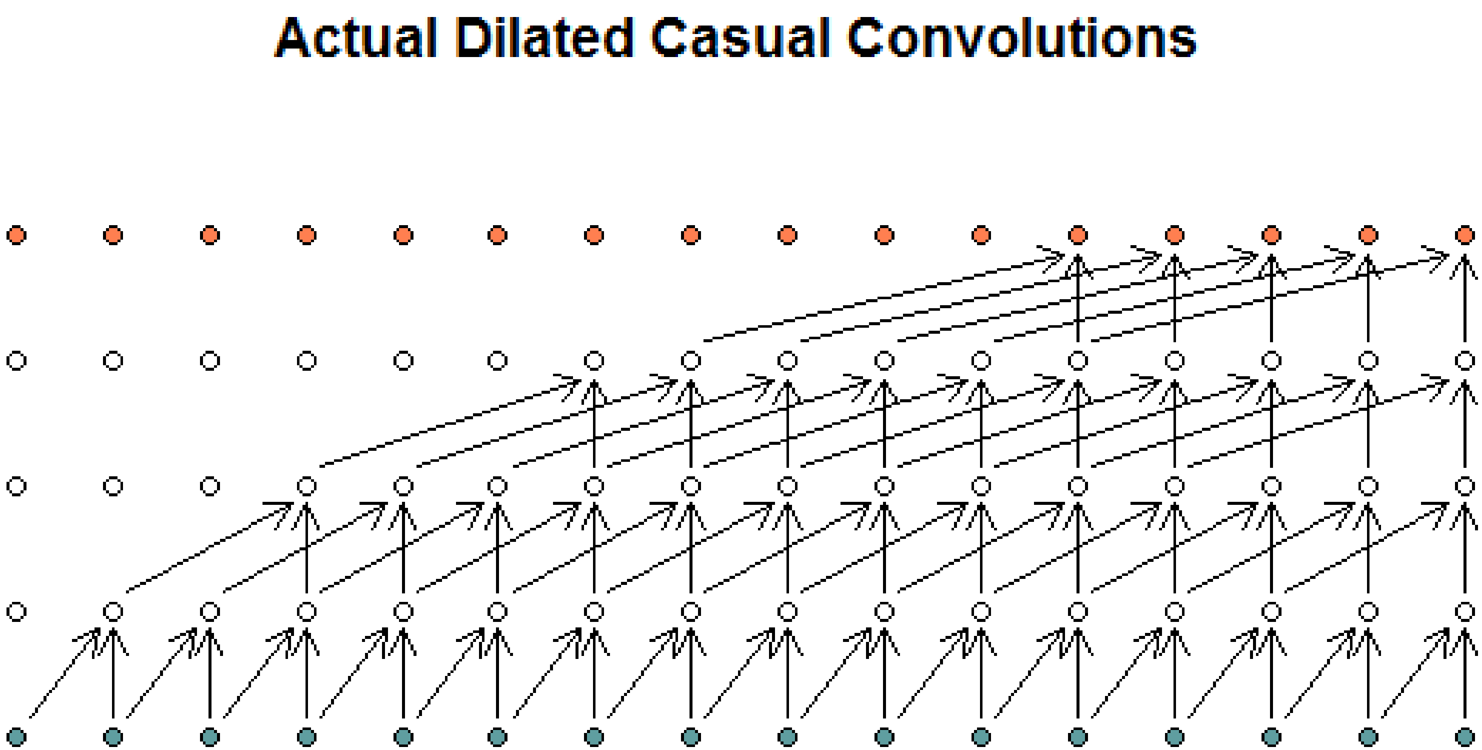


Figure from https://i.stack.imgur.com/RmJSu.png

Multiscale processing while maintaining original resolution!
Used for speech waveform generation.

Dilated Conv. In 2D

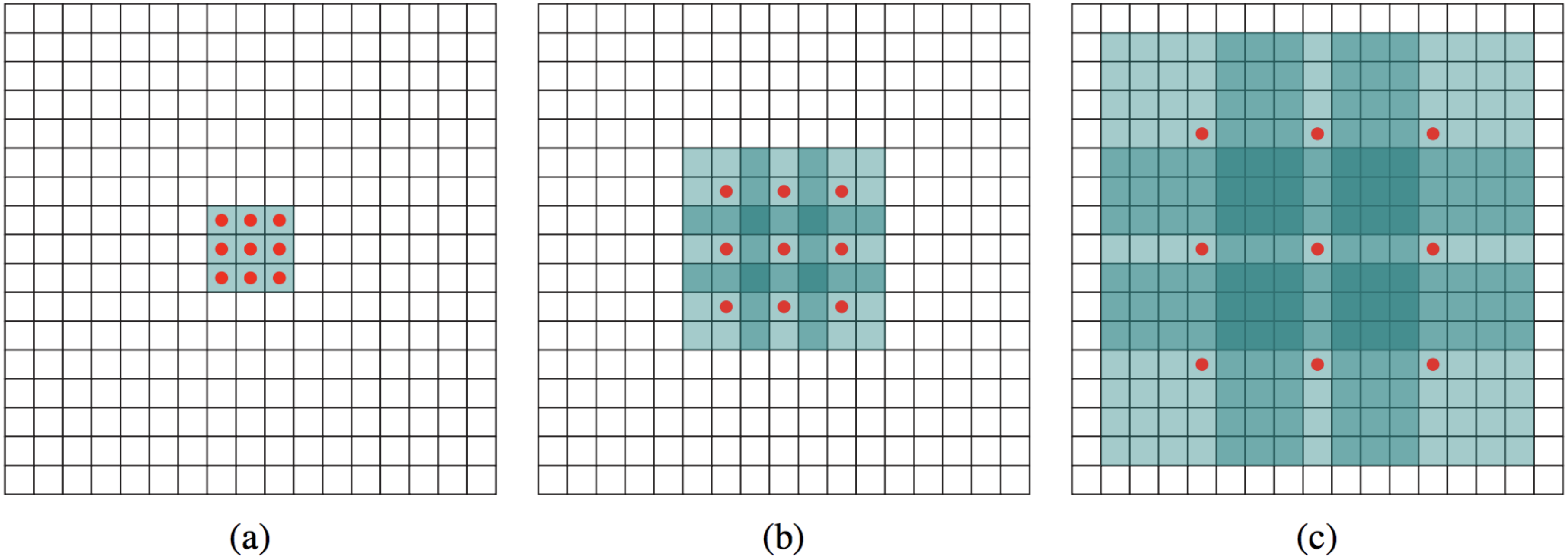


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

Yu, Fisher, and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions." *arXiv preprint arXiv:1511.07122* (2015).

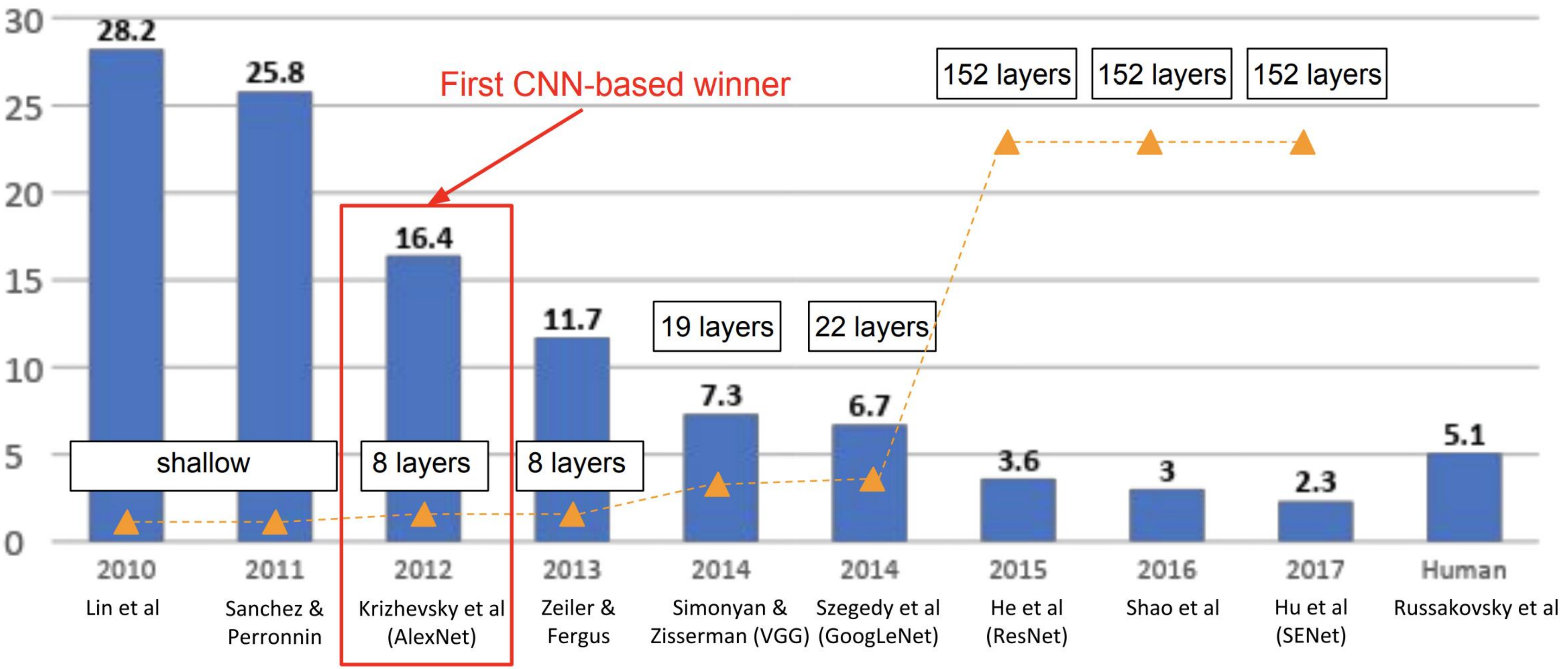
Multiscale processing while maintaining original resolution!
Good for dense prediction: image to image

Outline (Part II)

- Neural Nets and Conv Nets and Model Training (Review)
- Gradient calculation
- Some important extensions of conv. layers
- Popular classification models and transfer learning

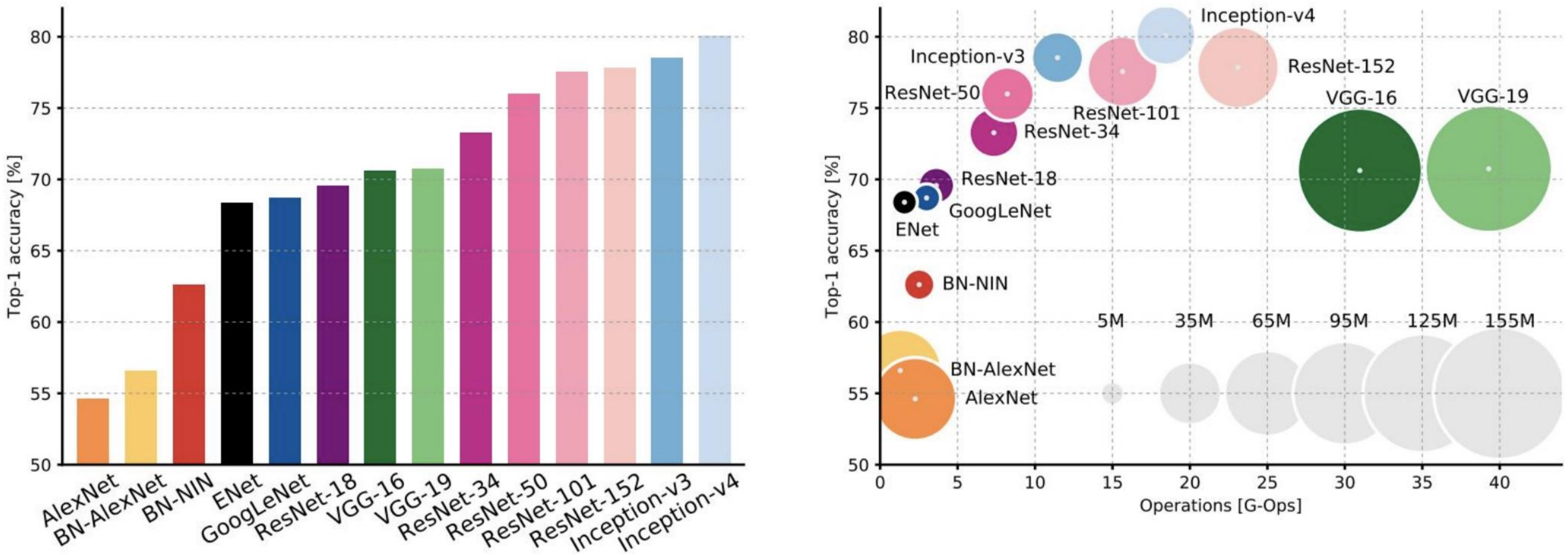
Well-Known Models

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf

Performance vs. Complexity



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

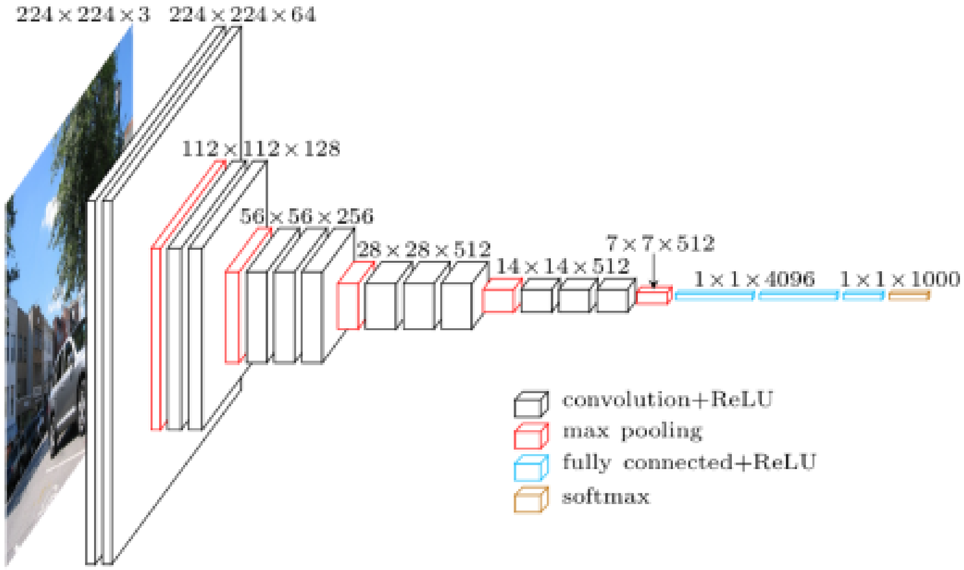
http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf

Transfer Learning

- For image classification or other applications, training from scratch takes tremendous resources
- Instead, can refine the VGG or other well trained networks
- Can use VGG convolutional layers, and retrain only the fully connected layers (possibly some later convolutional layers) for different problems.
- Or can use VGG conv layers as the “initial model” and further refine.
- Computer assignment: load VGG model, and fix all conv. layers, retrain additional fully connected layers for binary classification, try and compare different training tricks
  - Using Flickr API (courtesy of Sundeep Rangan) for downloading images for a given keyword

VGG16

- From the Visual Geometry Group
  - Oxford, UK
- Won ImageNet ILSVRC-2014
- Remains a very good network
- Lower layers are often used as feature extraction layers for other tasks



Model	top-5 classification error on ILSVRC-2012 (%)	
	validation set	test set
16-layer	7.5%	7.4%
19-layer	7.5%	7.3%
model fusion	7.1%	7.0%

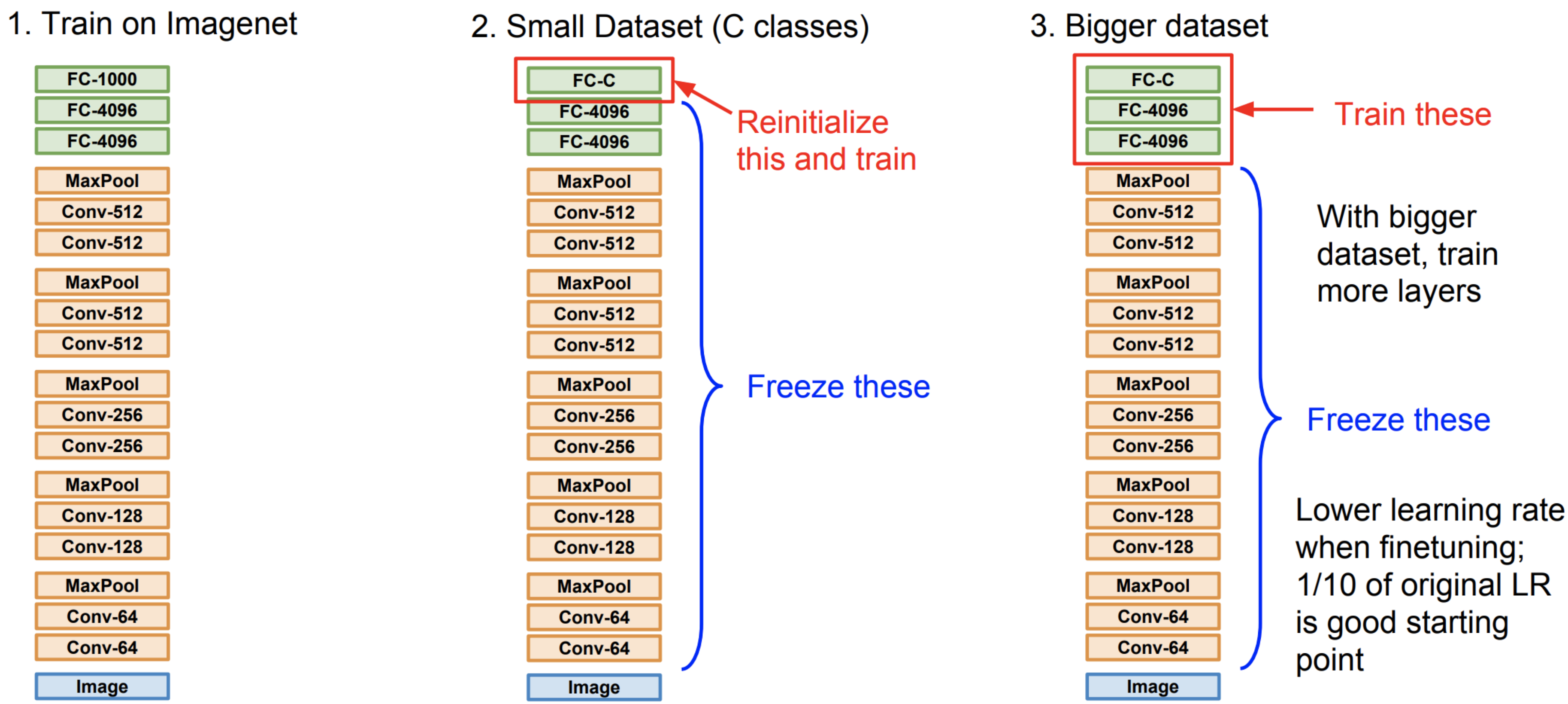
http://www.robots.ox.ac.uk/~vgg/research/very_deep/

K. Simonyan, A. Zisserman
Very Deep Convolutional Networks for Large-Scale Image Recognition
arXiv technical report, 2014

Transfer Learning

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



From http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

Takeaway for your projects and beyond:

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision

From http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture07.pdf

Summary

- Gradient Calculation through backpropagation
  - Tensor gradient, Tensor chain rule
- Residual and dense connections to ease gradient back propagation
- Dilated convolution for increasing perceptive field
- Transfer learning

Recommended Readings

- For tensor gradient calculation and backpropagation:
  - Lecture material of Sundeep Rangan
  - https://github.com/sdrangan/introml/blob/master/sequence.md
  - Unit on neural net and convolution networks
- For vision applications:
  - Stanford course by Feifei Li, et al: CS231n: Convolutional Neural Networks for Visual Recognition, Spring 2018: http://cs231n.stanford.edu/
  - Popular network case studies: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture09.pdf
  - Learning GPU and PyTorch and TensorFlow: http://cs231n.stanford.edu/slides/2018/cs231n_2018_lecture08.pdf
  - Video available for previous offerings:
    - https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYvhttps://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv