# Lecture 2:
# Medical Images: Classification

# Administrative

- A0 released, due Tue 10/4 11:59pm (setup assignment)
- A1 will also be released Tue, due in 2 weeks (Tue 10/18)
  - You will need to download several datasets to do the assignment. Make sure to start early!
  - 3 parts:
    - Medical image classification
    - Medical image segmentation in 2D
    - Medical image segmentation in 3D, with semi-supervised learning
- “Deep Learning Fundamentals” Review Session this Fri 1:30pm, Alway M112
- Numpy/Tensorflow Review Session next Fri 10/7, helpful for A1

# Administrative

- Office Hours starting next week
- For homework questions go to TA office hours
- Prof. Yeung's OH are for class and project questions only

# Agenda

Today: Medical Images: Classification

- Convolutional neural networks for image classification
- Data considerations for image classification models
- Evaluating image classification models
- Case studies of CNNs for medical image classification
  - More on transfer learning and how much data needs for deep learning
  - More on recent CNN architectures

# Agenda

Today: Medical Images: Classification

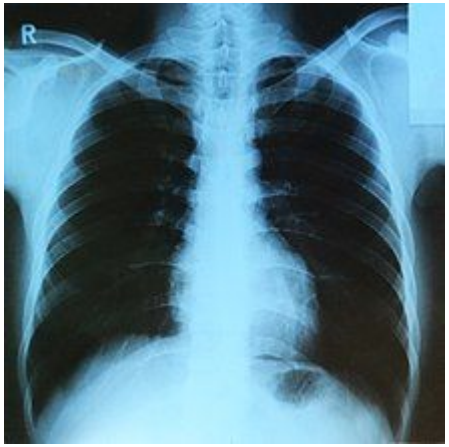
- Convolutional neural networks for image classification
- Data considerations for image classification models
- Evaluating image classification models
- Case studies of CNNs for medical image classification
  - More on transfer learning and how much data needs for deep learning
  - More on recent CNN architectures

Next Mon: Medical Images: Advanced Vision Models (Detection and Segmentation)

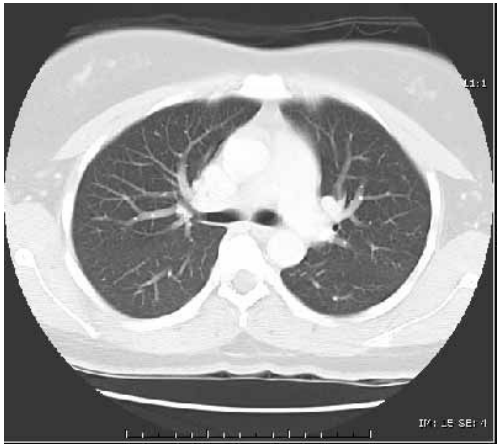
Next Wed: Medical Images: Advanced Vision Models (3D and Video)

# From last time: examples of medical image data

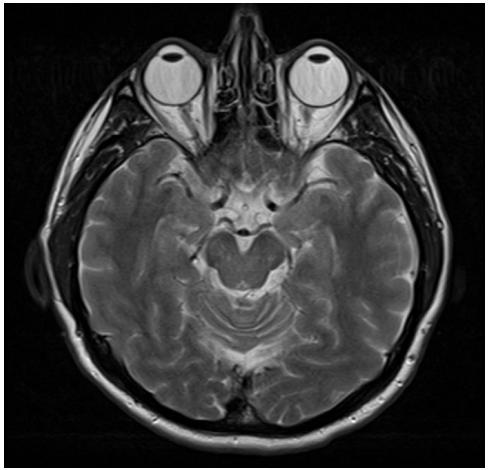
E.g.:




X-rays (invented 1895).



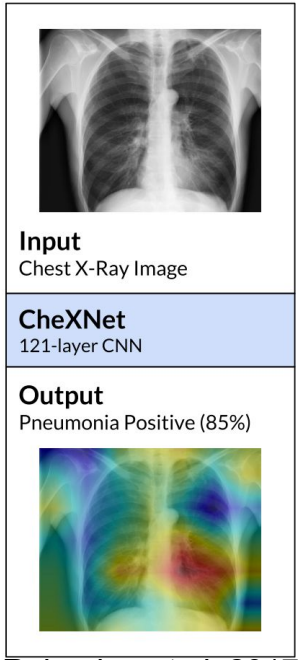
CT (invented 1972).



MRI (invented 1977).

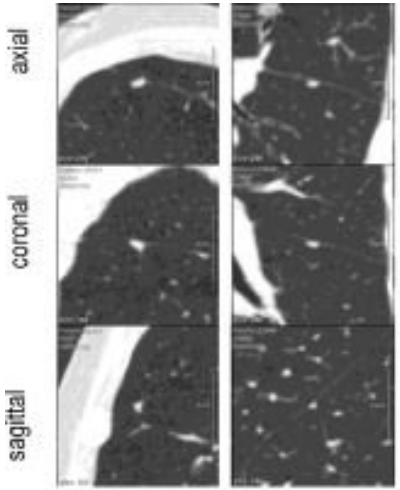
# Examples of medical image classification tasks

Is an x-ray positive for pneumonia or not?



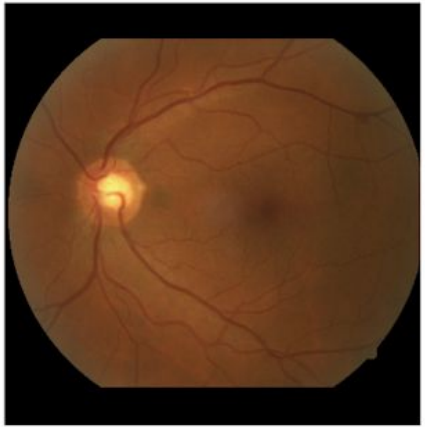

Rajpurkar et al. 2017

Is a CT scan of a lung nodule benign or not?



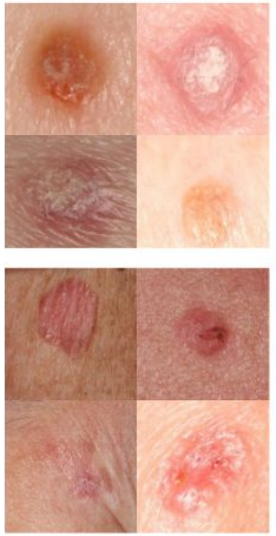

Ciompi et al. 2015

Is this moderate or worse diabetic retinopathy?



Gulshan et al. 2016

What types of skin lesions are these?



Esteva et al. 2017

# Deep Learning Models for Image Classification

# Machine learning framework

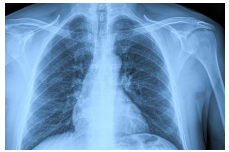
Data-driven learning of a mapping from input to output

# Machine learning framework

Data-driven learning of a mapping from input to output

**Traditional machine learning approaches**

**Input** → **Feature extractor** → **Machine learning model** → **Output**

(e.g.,  )

(e.g., color and texture histograms)

(e.g., support vector machines and random forests)

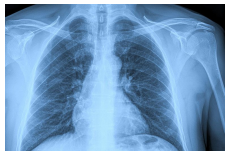
(e.g., presence or not or disease)

# Deep learning (a type of machine learning)

# Deep learning (a type of machine learning)

**Traditional machine learning**

**Input** → **Feature extractor** → **Machine learning model** → **Output**

(e.g.,  )

(e.g., color and texture histograms)

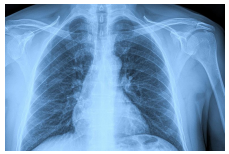
(e.g., support vector machines and random forests)

(e.g., presence or not or disease)

# Deep learning (a type of machine learning)

## Traditional machine learning

**Input** → **Feature extractor** → **Machine learning model** → **Output**

(e.g.,  )

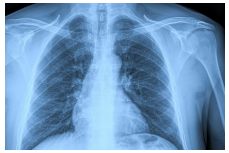
(e.g., color and texture histograms)

(e.g., support vector machines and random forests)

(e.g., presence or not or disease)

## Deep learning

**Input** → **Deep Learning Model** → **Output**

(e.g.,  )

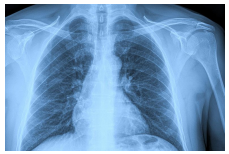
(e.g., convolutional and recurrent neural networks)

(e.g., presence or not or disease)

# Deep learning (a type of machine learning)

## Traditional machine learning

**Input** → **Feature extractor** → **Machine learning model** → **Output**

(e.g.,  )

(e.g., color and texture histograms)

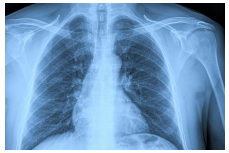
(e.g., support vector machines and random forests)

(e.g., presence or not or disease)

## Deep learning

Directly learns what are useful (and better!) features from the training data

**Input** → **Deep Learning Model** → **Output**

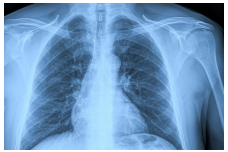
(e.g.,  )

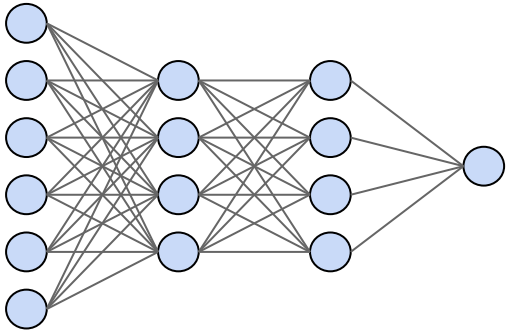
(e.g., convolutional and recurrent neural networks)

(e.g., presence or not or disease)

# How do deep learning models perform feature extraction?

**Input**

(e.g.,  )

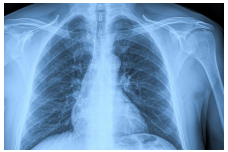


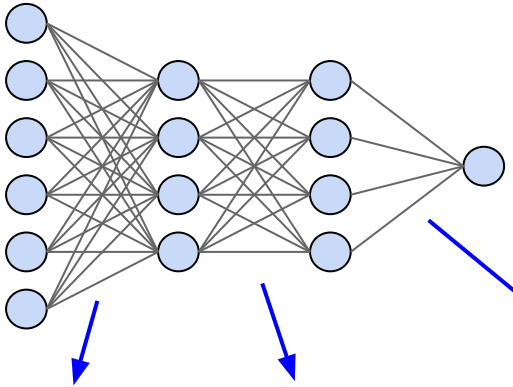
**Output**

(e.g., presence or not or disease)

# How do deep learning models perform feature extraction?

**Input**

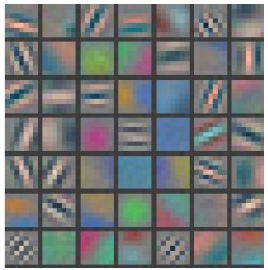
(e.g.,  )



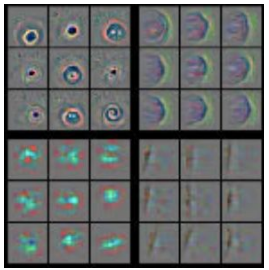
**Output**

(e.g., presence or not or disease)

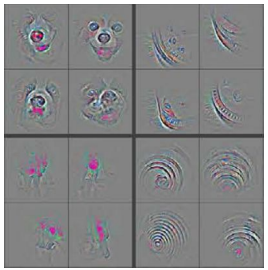
Hierarchical structure of neural networks allows compositional extraction of increasingly complex features



Low-level features



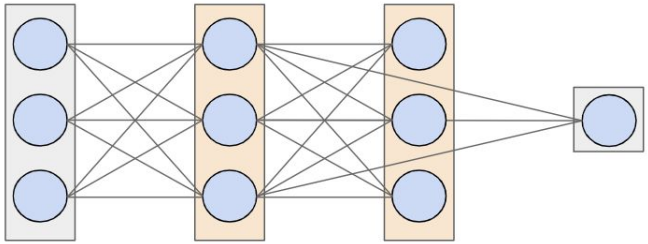
Mid-level features



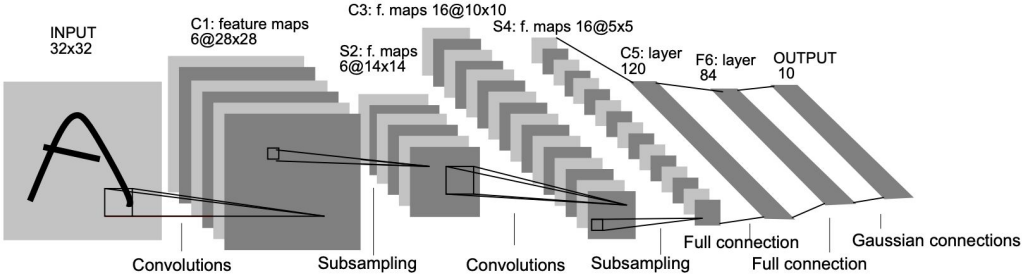
High-level features

Feature visualizations from Zeiler and Fergus 2013

# Different classes of neural networks

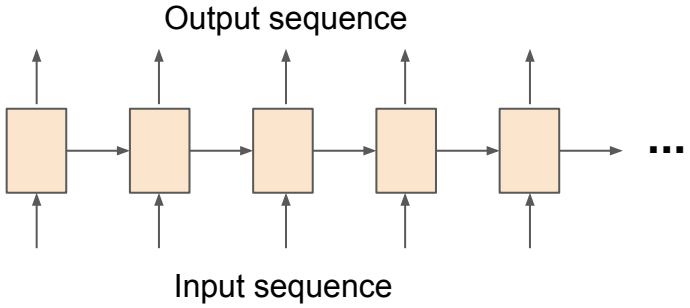


**Fully connected neural networks**
(linear layers, good for “feature vector” inputs)

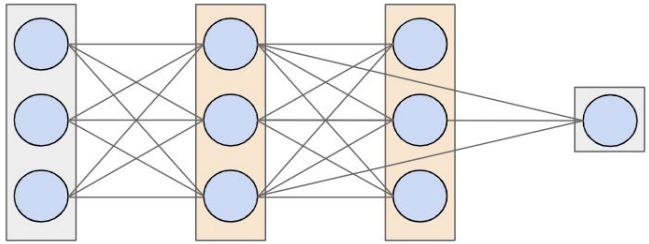



**Convolutional neural networks**
(convolutional layers, good for image inputs)

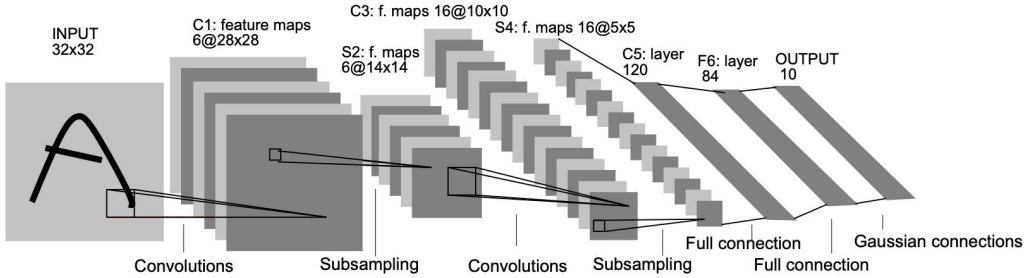
**Recurrent neural networks**
(linear layers modeling recurrence relation across sequence, good for sequence inputs)



# Different classes of neural networks

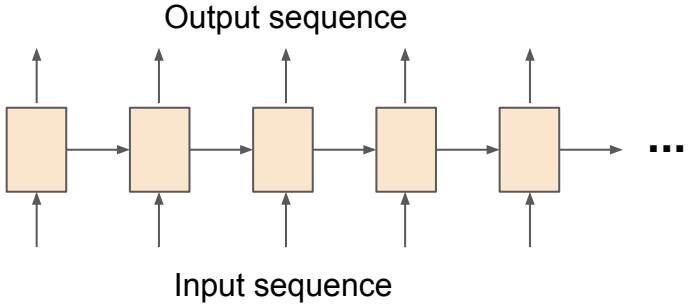


**Fully connected neural networks**
(linear layers, good for “feature vector” inputs)



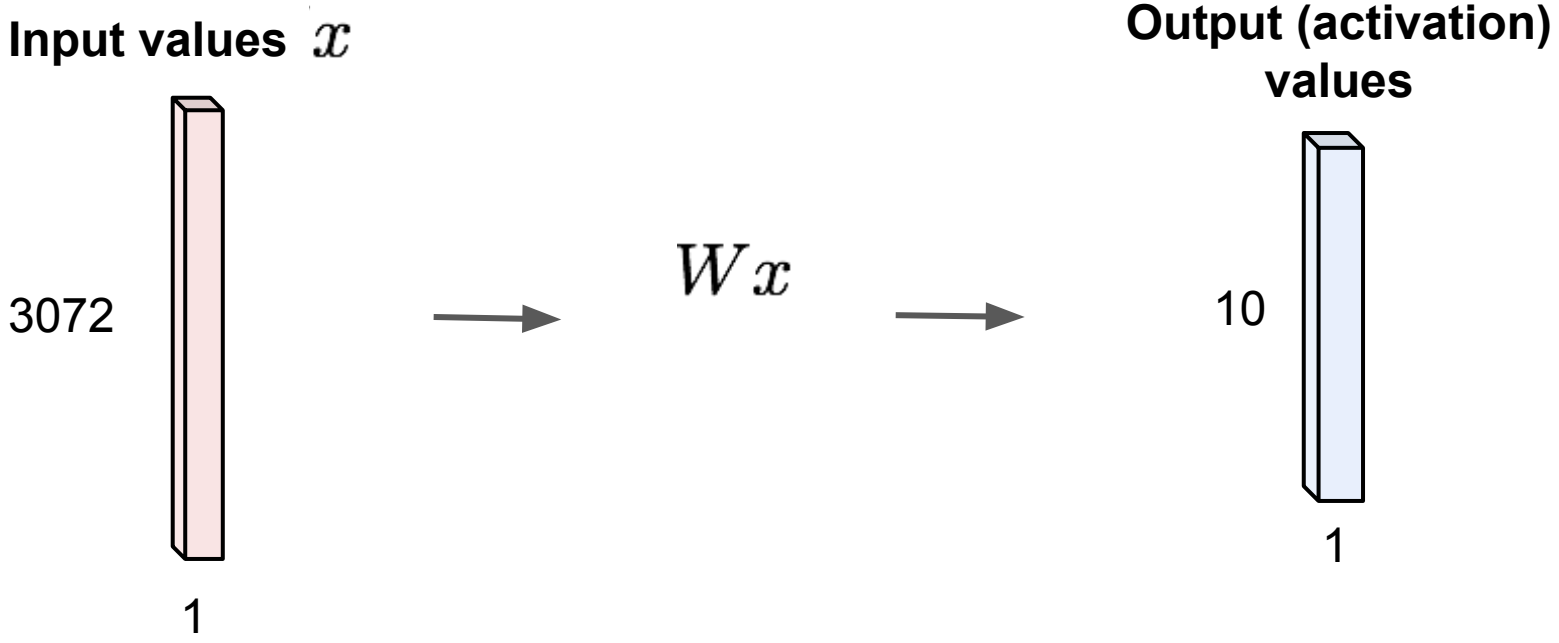

**Convolutional neural networks**
(convolutional layers, good for image inputs)

**Recurrent neural networks**
(linear layers modeling recurrence relation across sequence, good for sequence inputs)

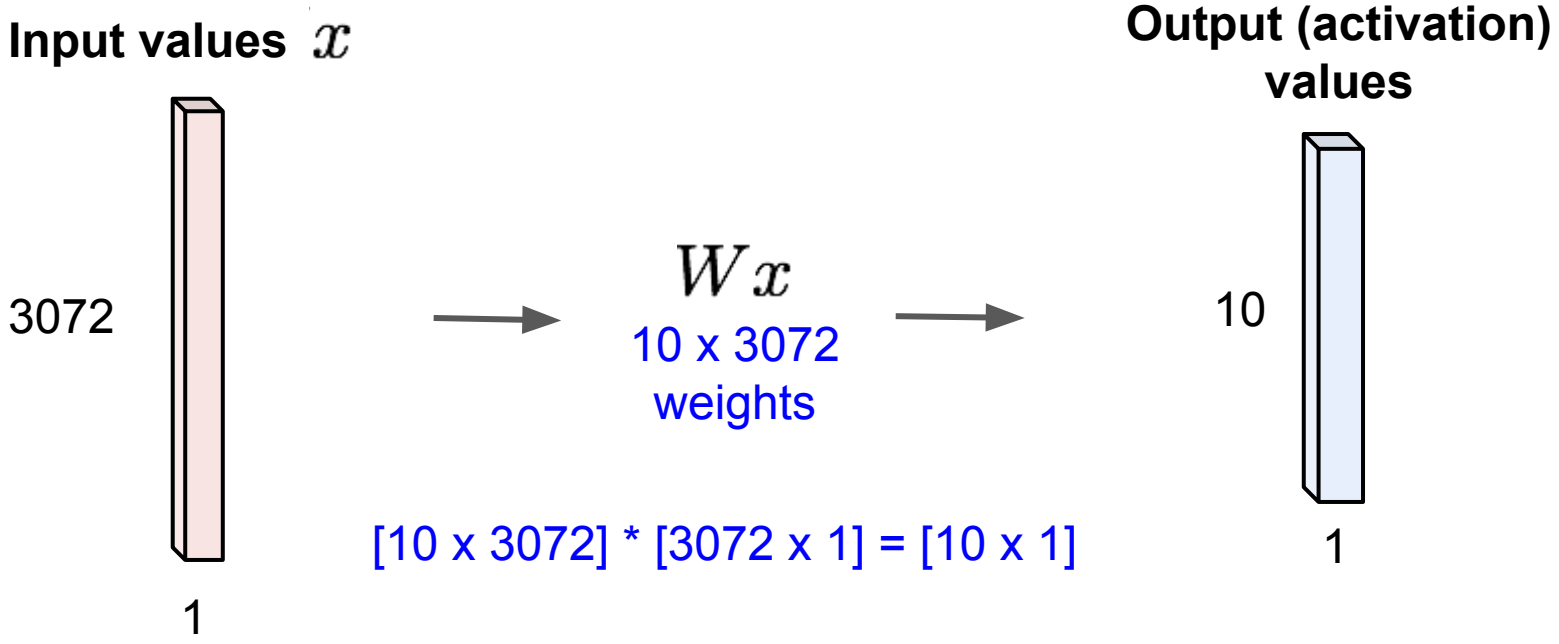


# Review: fully connected neural network layers

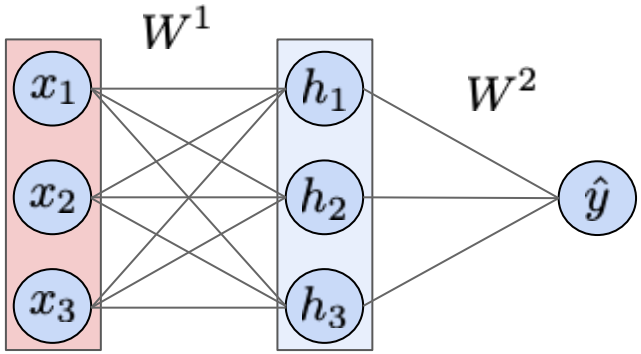
# Review: fully connected neural network layers



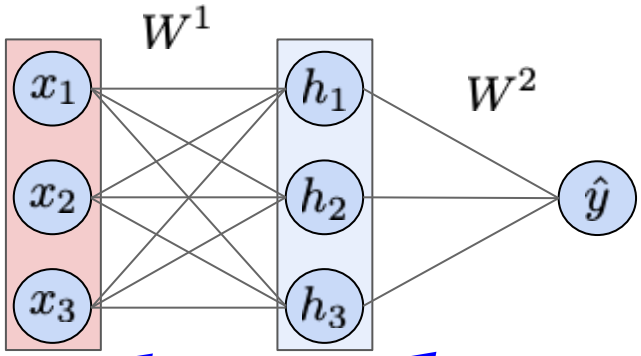
# Review: fully connected neural network layers



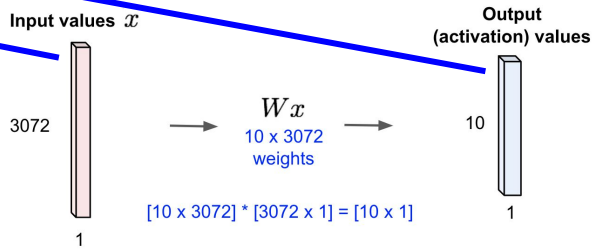
# Example: Simple two-layer fully-connected neural network



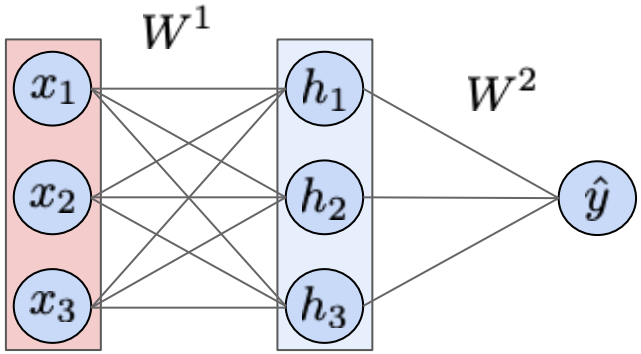
# Example: Simple two-layer fully-connected neural network




Each layer has the same structure we just saw, but this is a different example with different dimensions

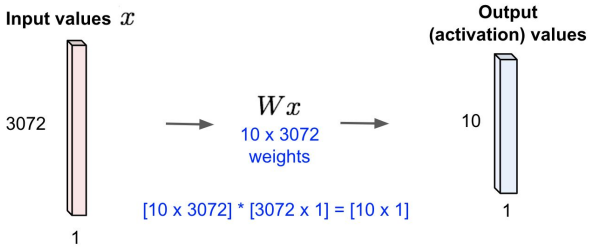


# Example: Simple two-layer fully-connected neural network

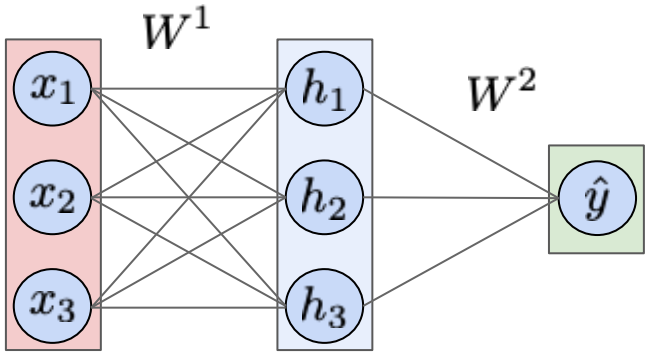



$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \end{bmatrix}$$

Each layer has the same structure we just saw, but this is a different example with different dimensions



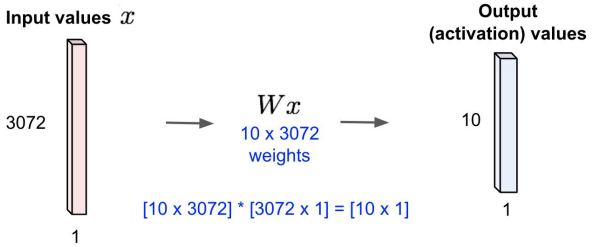
# Example: Simple two-layer fully-connected neural network



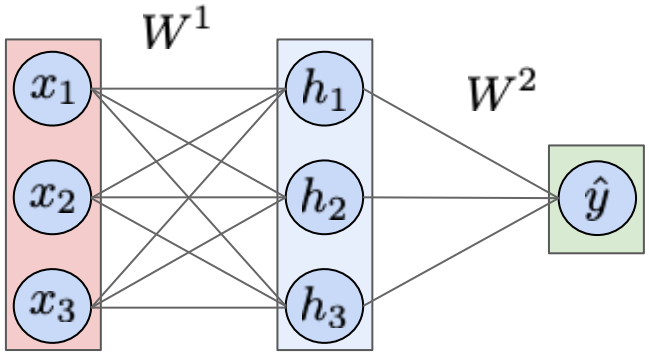

$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \end{bmatrix}$$

Each layer has the same structure we just saw, but this is a different example with different dimensions



# Example: Simple two-layer fully-connected neural network

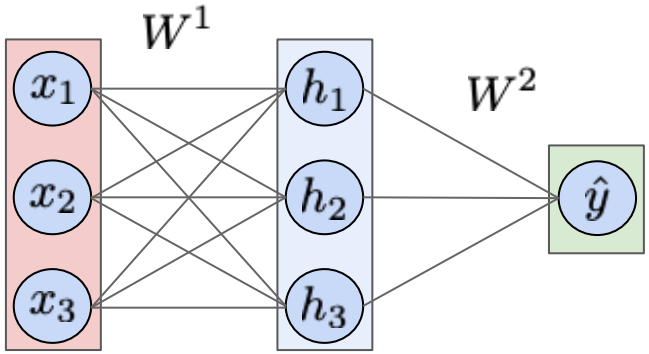



**Output:** $\hat{y} = W^2(\sigma(W^1 x + b^1)) + b^2$

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{bmatrix} \quad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \end{bmatrix} \quad b^2 = \begin{bmatrix} b_1^2 \end{bmatrix}$$

# Example: Simple two-layer fully-connected neural network



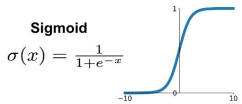

**Output:** $\hat{y} = W^2(\sigma(W^1 x + b^1)) + b^2$

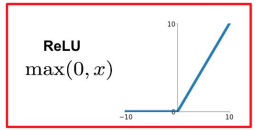
Common activation functions

You can find these in Keras: https://keras.io/layers/advanced-activations/

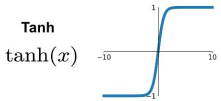
You will see these extensively, typically after linear or convolutional layers.
They add nonlinearity to allow the model to express complex nonlinear functions.

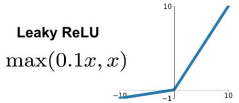
Typical in modern CNNs and MLPs




and many more...

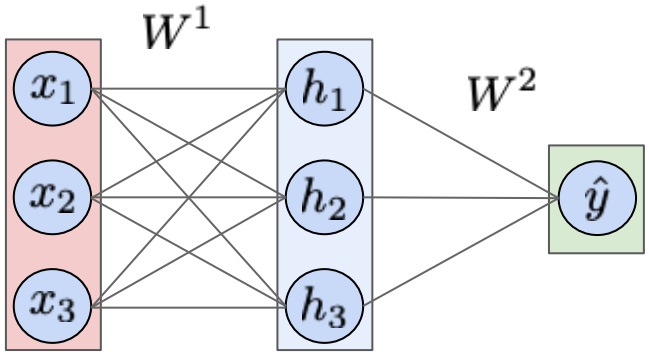



$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{bmatrix} \quad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \end{bmatrix} \quad b^2 = \begin{bmatrix} b_1^2 \end{bmatrix}$$

# Example: Simple two-layer fully-connected neural network




**Output:** $\hat{y} = W^2(\sigma(W^1 x + b^1)) + b^2$

**Neural network parameters:**

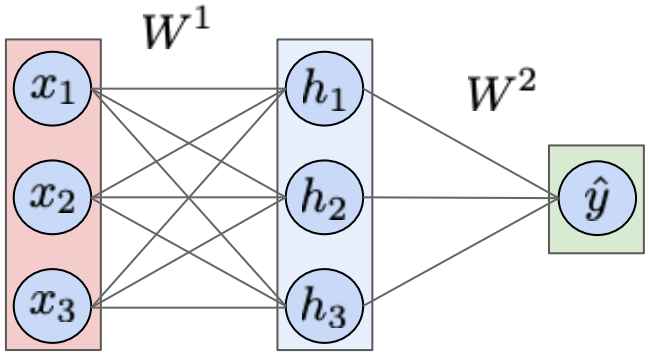
$$W = \{W^1, b^1, W^2, b^2\}$$

"weights" "biases"

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{bmatrix} \quad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \end{bmatrix} \quad b^2 = \begin{bmatrix} b_1^2 \end{bmatrix}$$

# Example: Simple two-layer fully-connected neural network




**Output:** $\hat{y} = W^2(\sigma(W^1 x + b^1)) + b^2$

**Neural network parameters:**

$$W = \{W^1, b^1, W^2, b^2\}$$

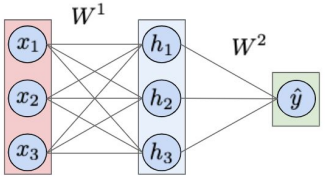
"weights" "biases"

Often refer to all parameters together as just "weights". Bias is implicitly assumed.

$$W^1 = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{bmatrix} \quad b^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 \end{bmatrix} \quad b^2 = \begin{bmatrix} b_1^2 \end{bmatrix}$$

# Commonly used for feature vector inputs




Let us consider the task of **regression**: predicting a single real-valued output from input data

**Model input:** data vector $x = [x_1, x_2, ..., x_N]$ **Model output:** prediction (single number) $\hat{y}$

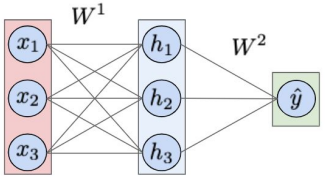
Example: predicting hospital length-of-stay from clinical variables in the electronic health record

$x =$ [age, weight, …, temperature, oxygen saturation] $\hat{y} =$ length-of-stay (days)

Example: predicting expression level of a target gene from the expression levels of N landmark genes

$x \in \mathcal{R}^N =$ expression levels of N landmark genes $\hat{y} =$ expression level of target gene

# Can also be applied for classification




The task of **classification** predicts a categorical output from input data

**Model input:** data vector $x = [x_1, x_2, ..., x_N]$

**Model output:** prediction of 1-of-K classes

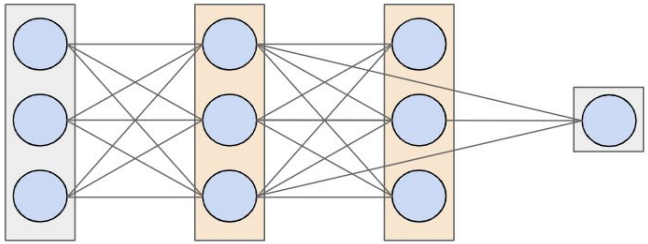
$$\hat{y} \in \{1, ..., K\}$$

Example: predicting in-hospital mortality from clinical variables in the electronic health record

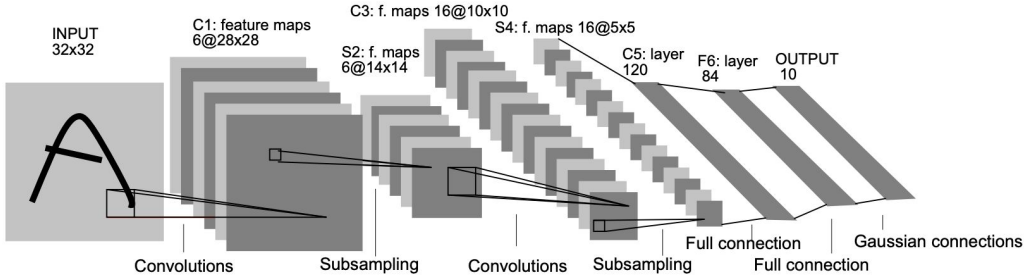
$x =$ [age, weight, …, temperature, oxygen saturation]

$\hat{y} \in \{0, 1\}$ for occurrence of in-hospital mortality

# Different classes of neural networks

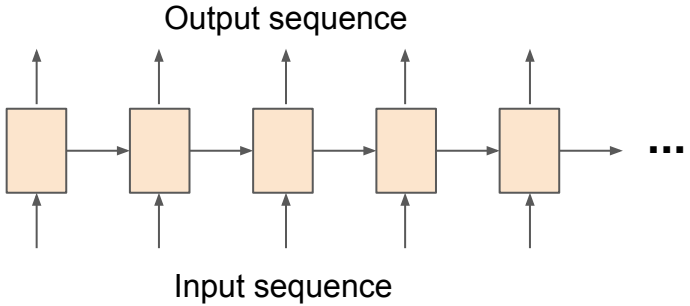


**Fully connected neural networks**
(linear layers, good for “feature vector” inputs)

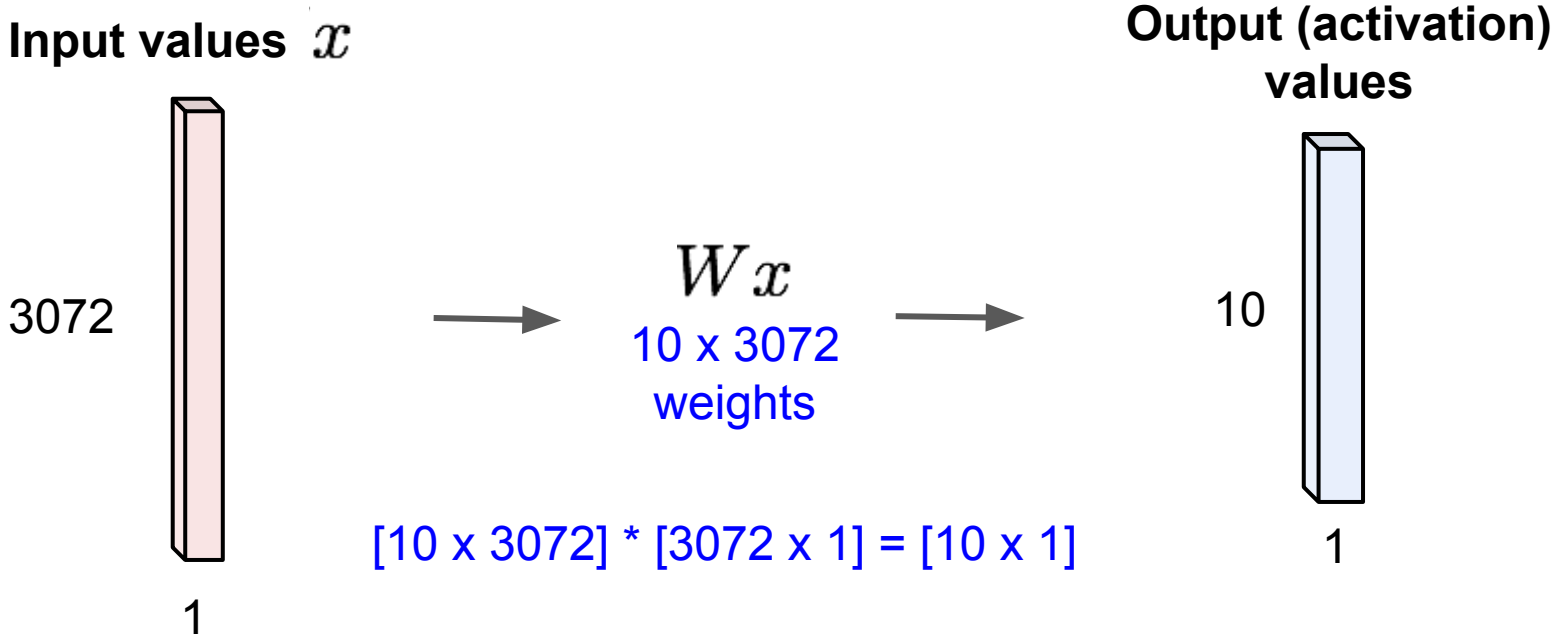



**Convolutional neural networks**
(convolutional layers, good for image inputs)

**Recurrent neural networks**
(linear layers modeling recurrence relation across sequence, good for sequence inputs)

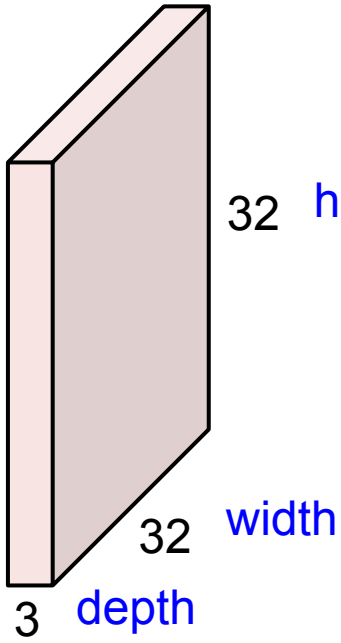


# Previously: Fully connected layer



# Now: Convolutional layer

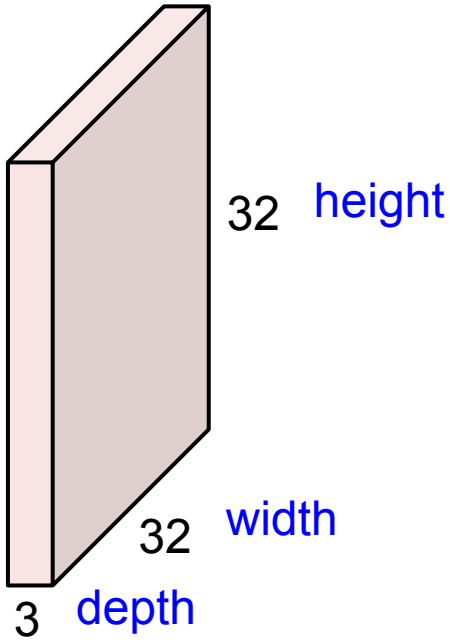
32x32x3 image -> preserve spatial structure




Slide credit: CS231n

# Convolutional layer

32x32x3 image -> preserve spatial structure



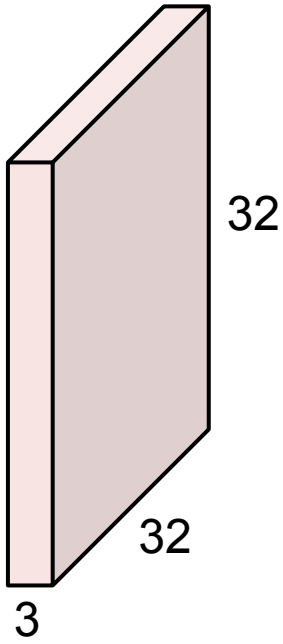

Input now has spatial height and width dimensions!

In contrast to fully-connected layers, want to preserve spatial structure when processing with a convolutional layer

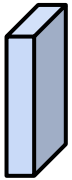
Slide credit: CS231n

# Convolutional layer

32x32x3 image




5x5x3 filter (weights)



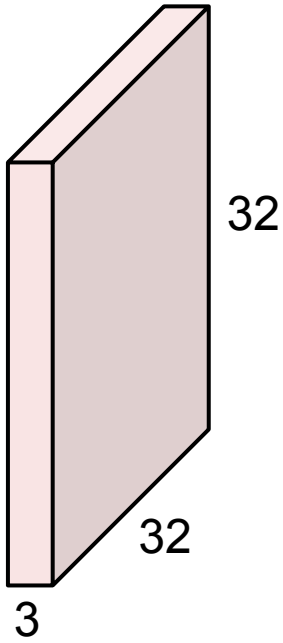
**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS231n

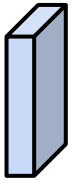
# Convolutional layer

Filters always extend the full depth of the input volume

32x32x3 image



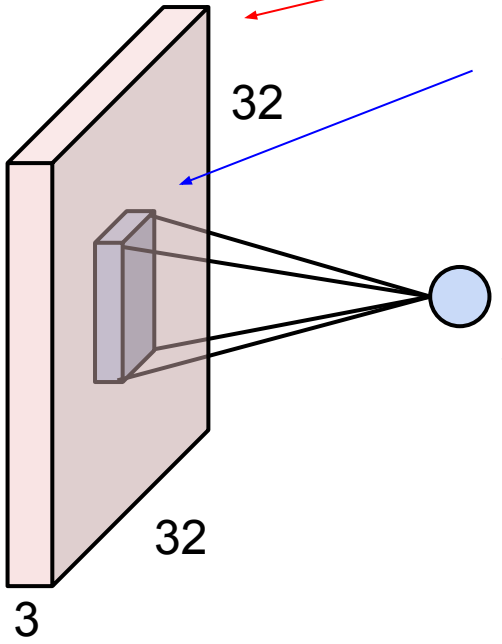

5x5x3 filter (weights)



**Convolve** the filter with the image i.e. “slide over the image spatially, computing dot products”

Slide credit: CS231n

# Convolutional layer




32x32x3 image

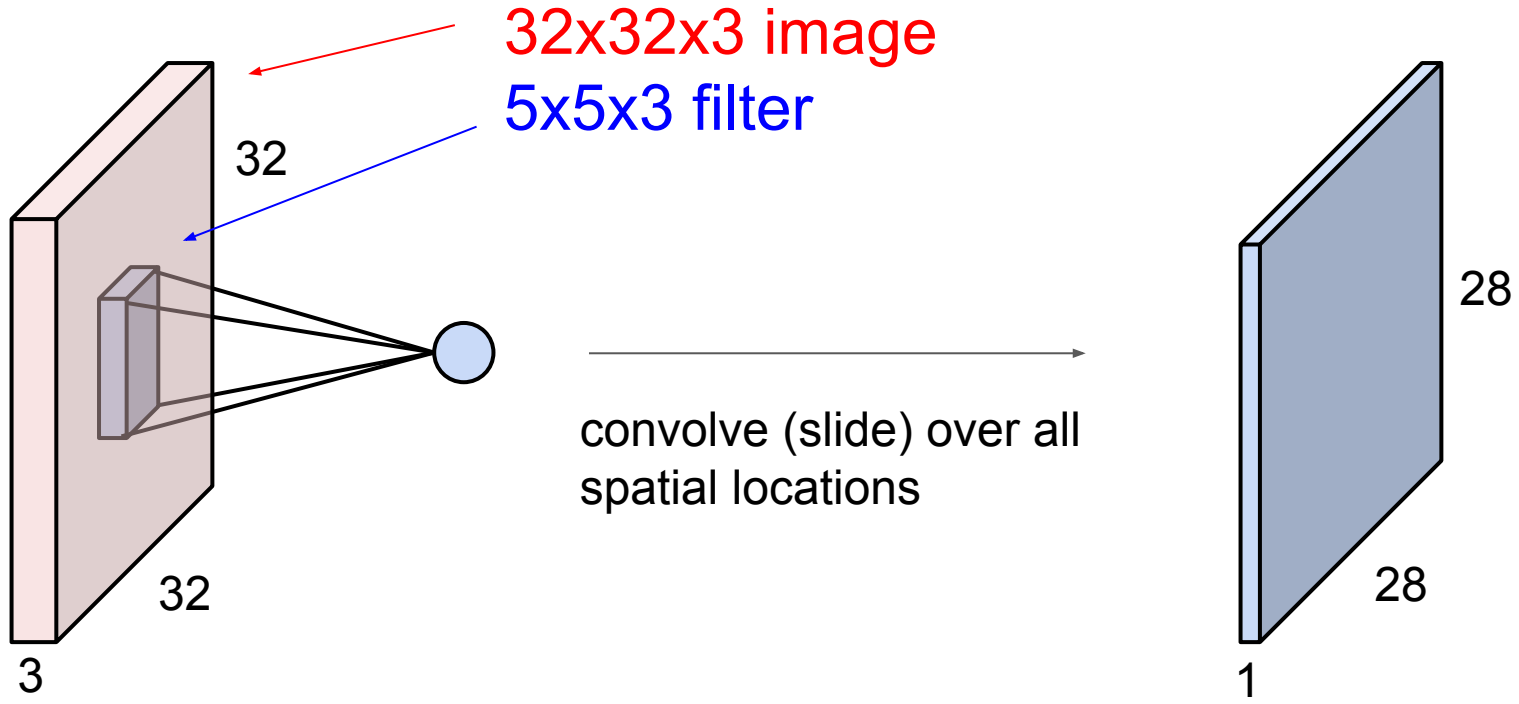
5x5x3 filter $w$

**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)

$$w^T x + b$$

Slide credit: CS231n

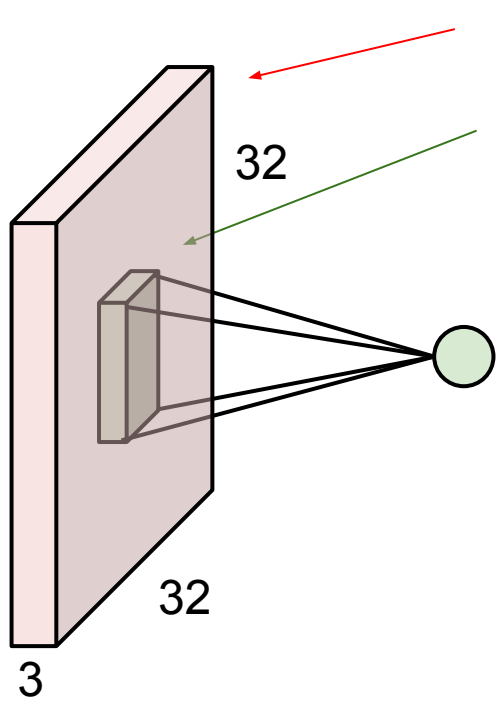
# Convolutional layer



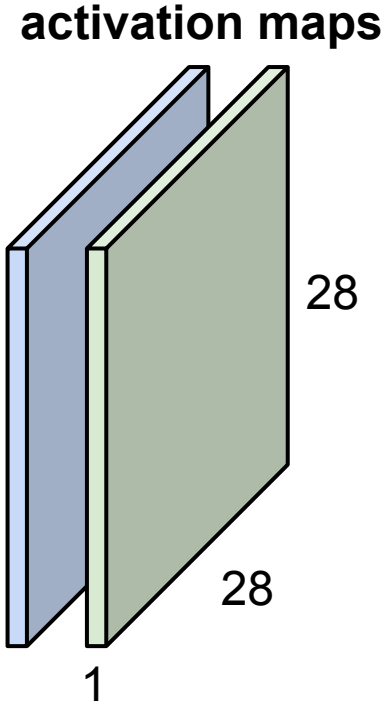

Slide credit: CS231n

consider a second, green filter

# Convolutional layer

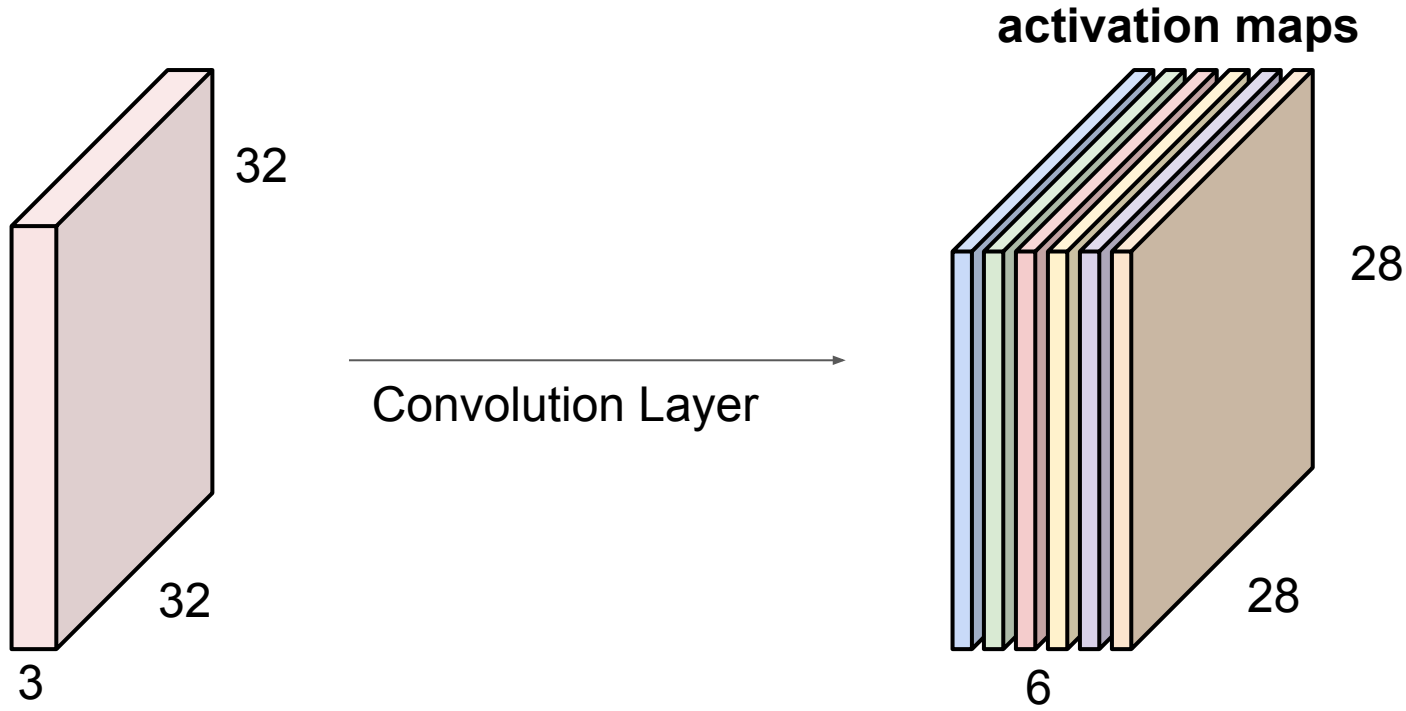



convolve (slide) over all spatial locations




Slide credit: CS231n

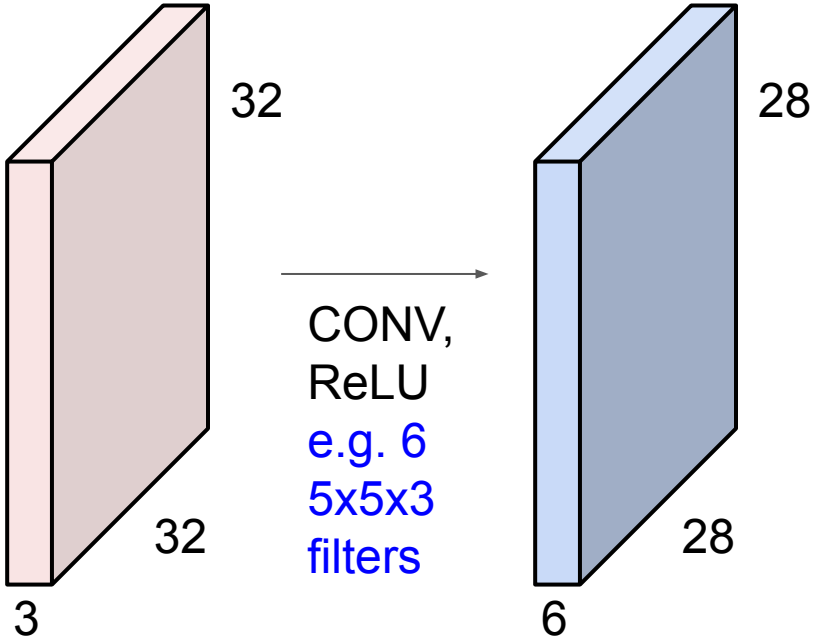
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:




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

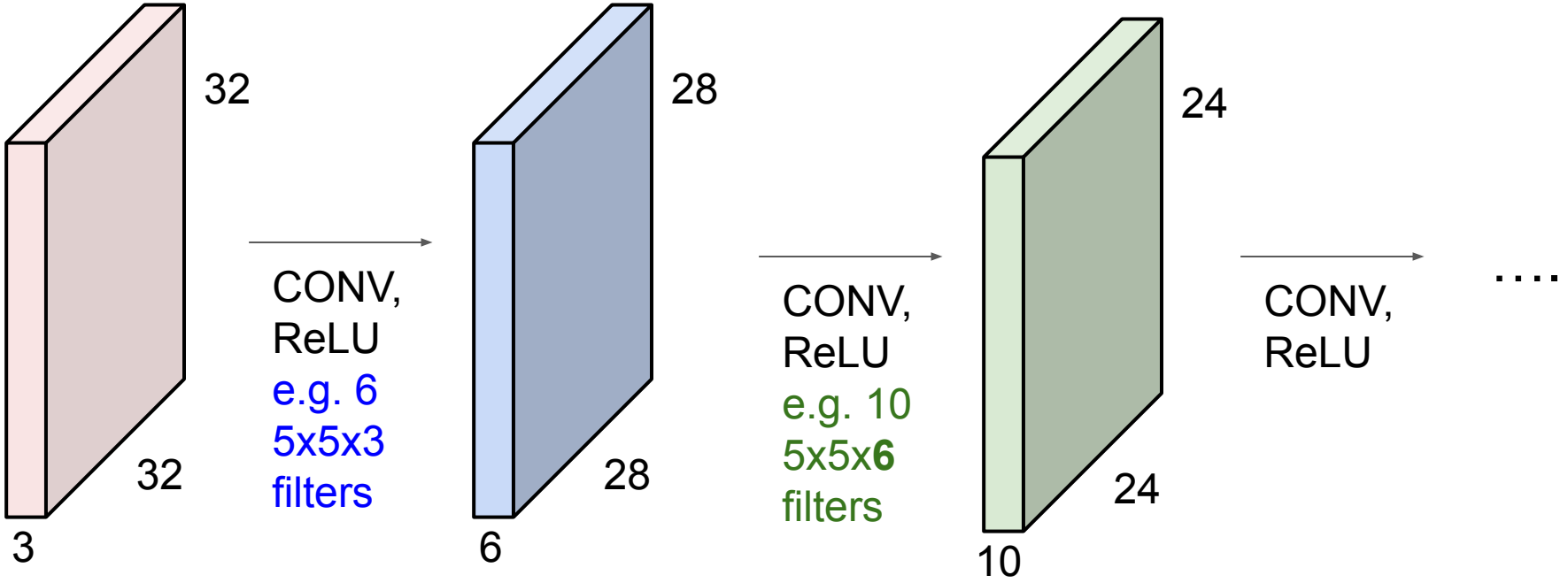
Slide credit: CS231n

**Preview:** ConvNet (or CNN) is a sequence of Convolution Layers, interspersed with activation functions




Slide credit: CS231n

**Preview:** ConvNet (or CNN) is a sequence of Convolution Layers, interspersed with activation functions




Slide credit: CS231n

**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.

Slide credit: CS231n

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

**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.

Slide credit: CS231n

# In Keras

**Conv2D** [source]

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, d
```

2D convolution layer (e.g. spatial convolution over images).

# In Keras

Padding options: ‘valid’ does not pad, use ‘same’ to pad such that input and output spatial dimensions are the same size

one filter =>
one activation map

example 5x5 filters
(32 total)

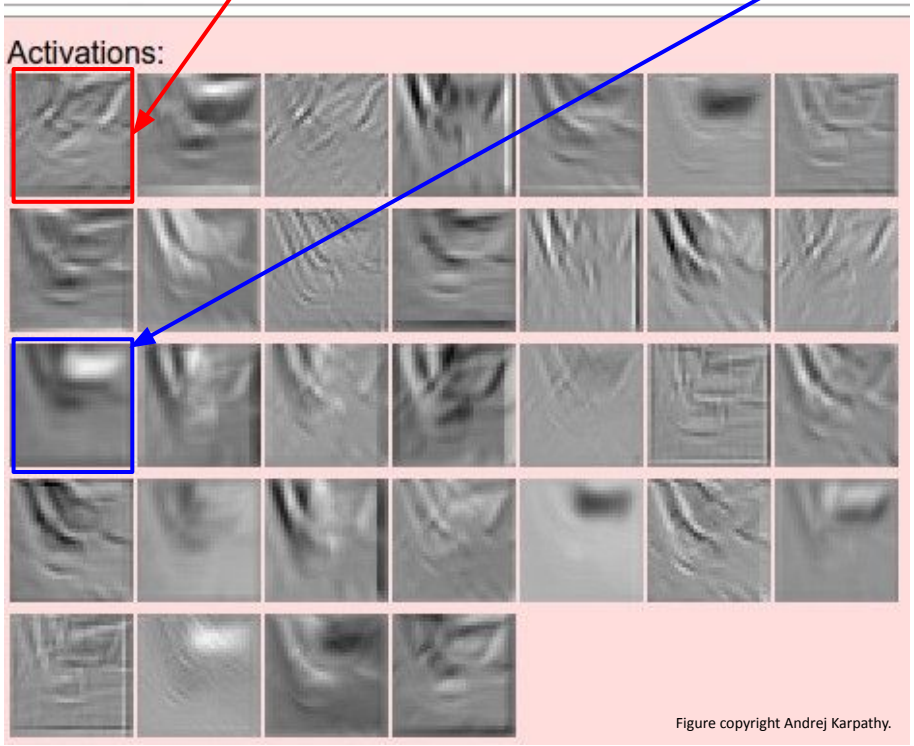
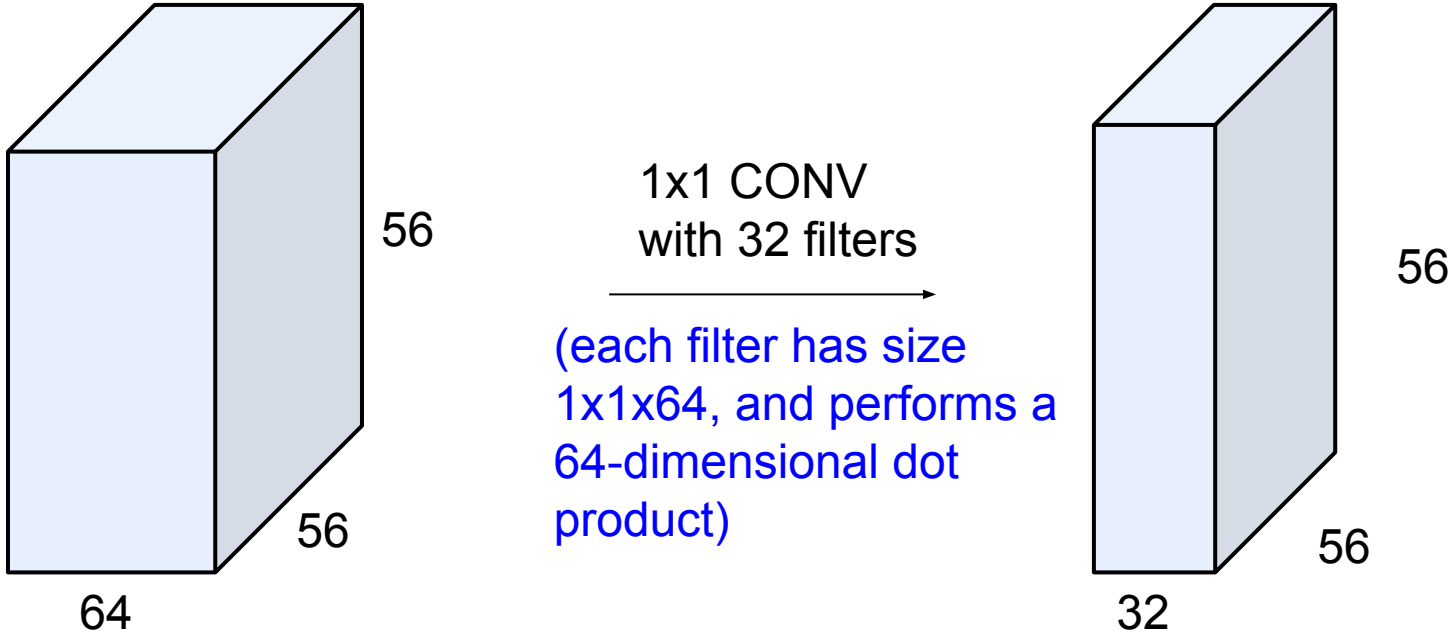



Figure copyright Andrej Karpathy.

Slide credit: CS231n

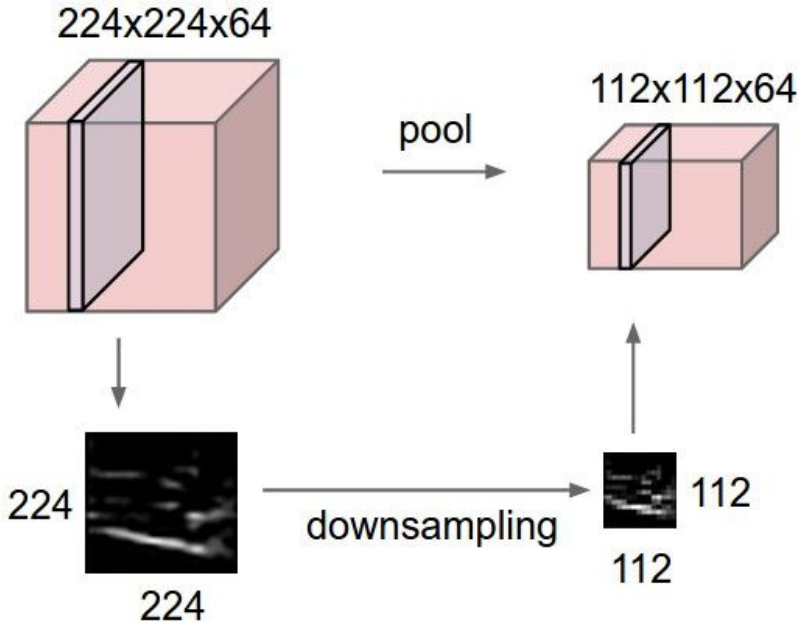
(btw, 1x1 convolution layers make perfect sense -> performs **dimensionality reduction** in the depth dimension)




Slide credit: CS231n

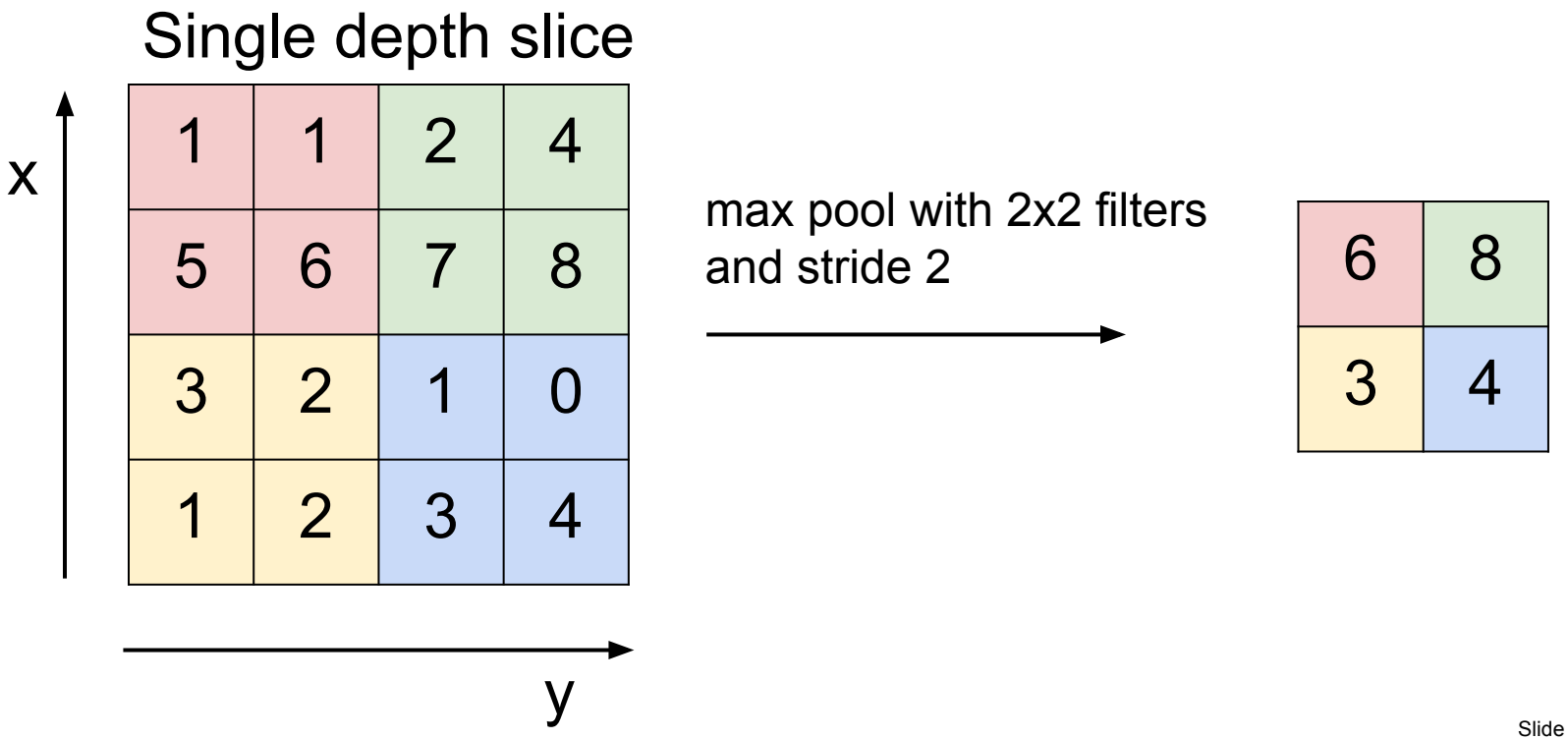
# Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:




Slide credit: CS231n

# Max pooling




Slide credit: CS231n

# Pooling layer: practical implementation

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent $F$,
  - the stride $S$,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

In Keras:

**MaxPooling2D** [source]

```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
```

Slide credit: CS231n

# Pooling layer: practical implementation

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent $F$,
  - the stride $S$,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

In Keras:

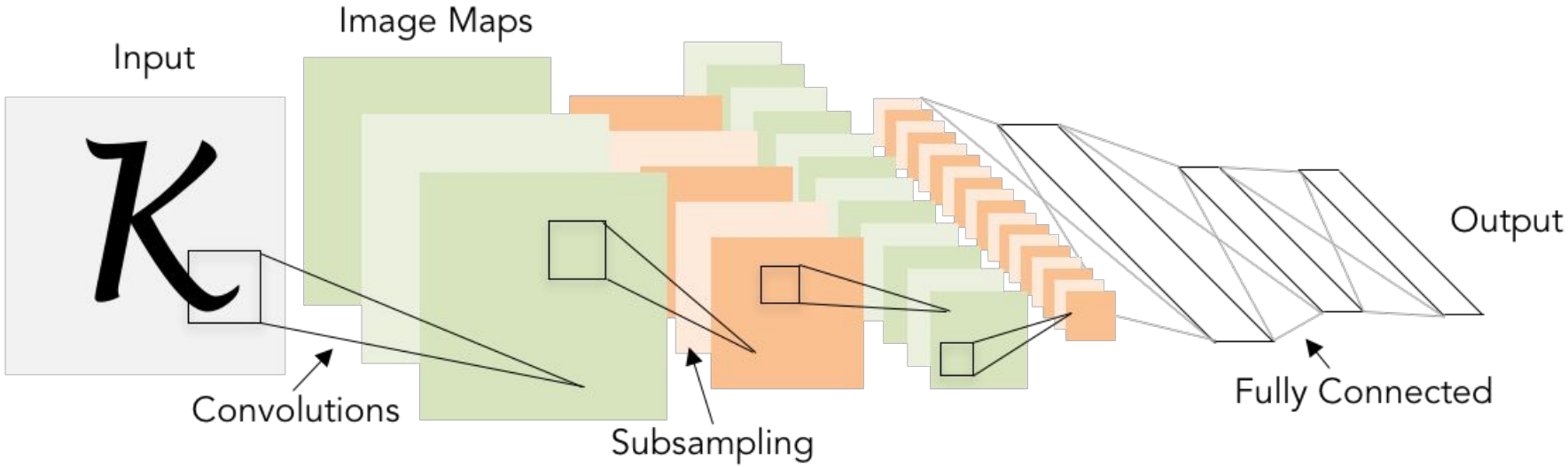
**MaxPooling2D** [source]

```
keras.layers.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid', data_format=None)
```

Slide credit: CS231n

# LeNet-5

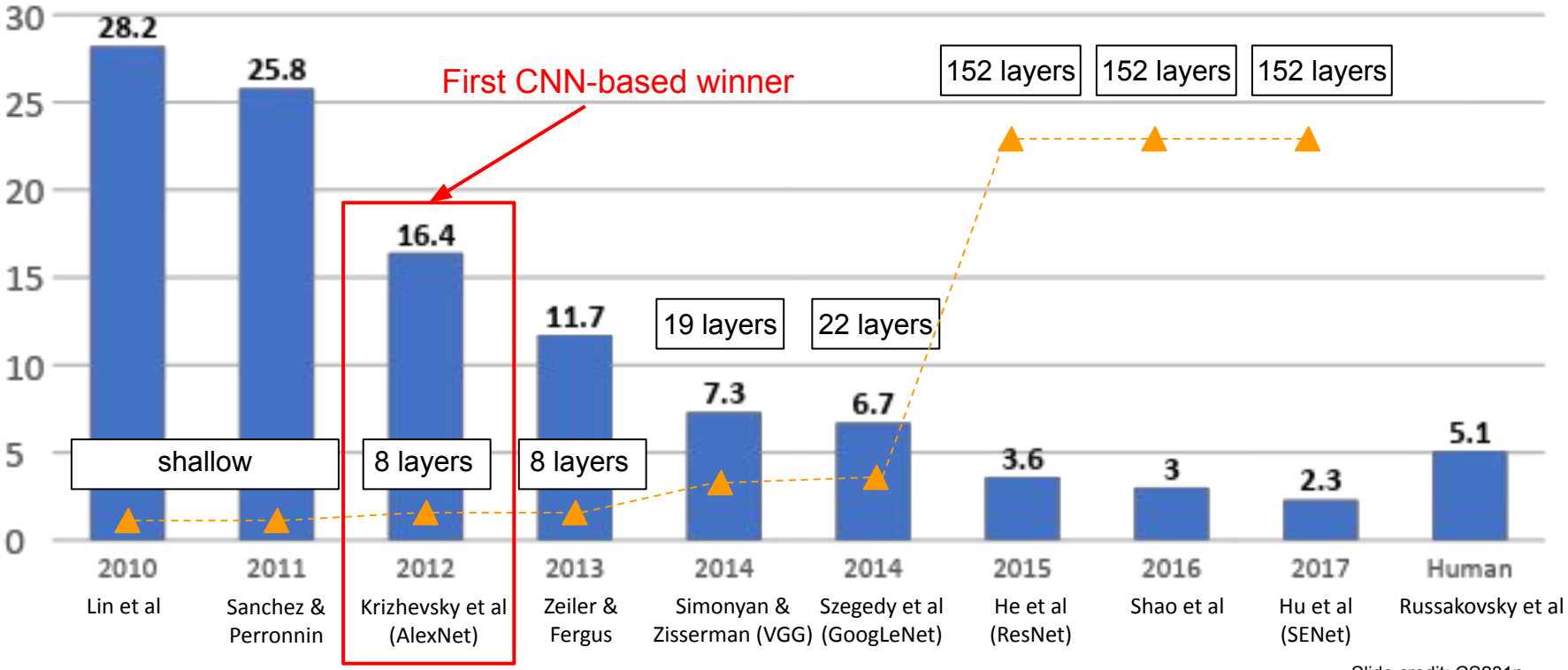
[LeCun et al., 1998]




Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC-FC]

Slide credit: CS231n

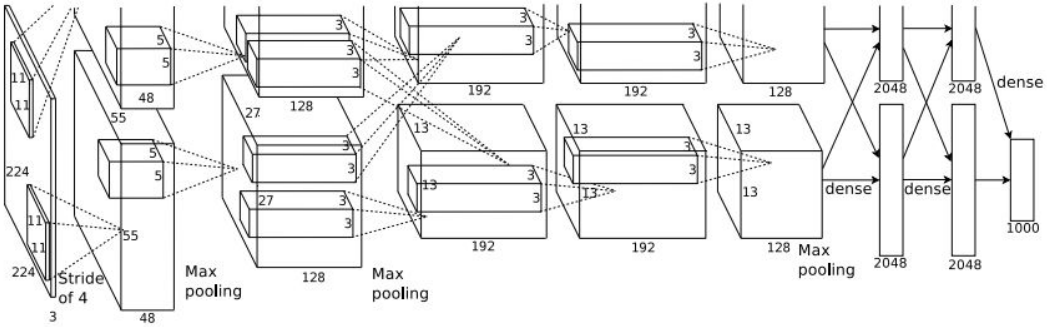
# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners




Slide credit: CS231n

# AlexNet

*[Krizhevsky et al. 2012]*

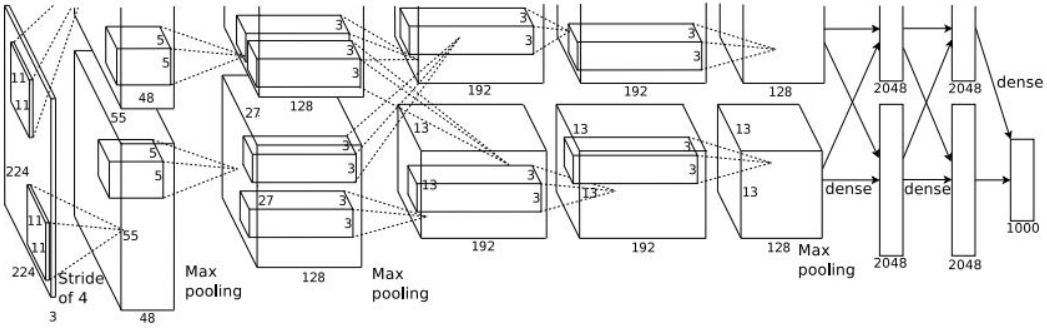



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)


Slide credit: CS231n

# AlexNet

*[Krizhevsky et al. 2012]*




Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

**Details (discussed further in review session):**
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4 (regularization)
- 7 CNN ensemble: 18.2% -> 15.4%


Slide credit: CS231n

# Common loss functions

**Regression**

$$L_{regression} = \frac{1}{M} \sum_{i} (\hat{y}^i - y^i)^2$$

Label is a continuous value.

# Common loss functions

**Regression** Minimize squared difference between prediction output and target

$$L_{regression} = \frac{1}{M} \sum_{i} (\hat{y}^i - y^i)^2$$

Label is a continuous value.

# Common loss functions

**Regression**

Minimize squared difference between prediction output and target

$$L_{regression} = \frac{1}{M}\sum_{i}(\hat{y}^i - y^i)^2$$

Label is a continuous value.

**Binary Cross-Entropy**

$$L_{BCE} = \frac{1}{M}\sum_{i} -(y_i log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

Label is binary in {0,1}. Prediction is a real number in (0,1) and is the probability of the label being 1. It is usually the output of a sigmoid operation after the final layer.

# Common loss functions

Equivalent to the negative log of the probability of the correct ground truth class being predicted. Think about what the expression looks like when y_i = 1 vs. 0.

**Regression**

Minimize squared difference between prediction output and target

$$L_{regression} = \frac{1}{M}\sum_{i}(\hat{y}^i - y^i)^2$$

Label is a continuous value.

**Binary Cross-Entropy**

$$L_{BCE} = \frac{1}{M}\sum_{i} -(y_i log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

Label is binary in {0,1}. Prediction is a real number in (0,1) and is the probability of the label being 1. It is usually the output of a sigmoid operation after the final layer.

# Common loss functions

Equivalent to the negative log of the probability of the correct ground truth class being predicted. Think about what the expression looks like when y_i = 1 vs. 0.

**Regression**

Minimize squared difference between prediction output and target

$$L_{regression} = \frac{1}{M}\sum_i (\hat{y}^i - y^i)^2$$

Label is a continuous value.

**Binary Cross-Entropy**

$$L_{BCE} = \frac{1}{M}\sum_i -(y_i log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i))$$

Label is binary in {0,1}. Prediction is a real number in (0,1) and is the probability of the label being 1. It is usually the output of a sigmoid operation after the final layer.

**Softmax**

$$L_{Softmax} = \frac{1}{M}\sum_i -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

Label is 1 of K classes in {0, …, K}. Extension of binary cross-entropy loss to multiple classes. s_j corresponds to the score (e.g. output of final layer) for each class; the fraction in the log provides a normalized probability for each class.

# Common loss functions

**Regression** Minimize squared difference between prediction output and target

$$L_{regression} = \frac{1}{M}\sum_i (\hat{y}^i - y^i)^2$$

Label is a continuous value.

**Softmax** Negative log of the probability of the true class y_i, as with the BCE loss.

$$L_{Softmax} = \frac{1}{M}\sum_i -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Label is 1 of K classes in {0, …, K}. Extension of binary cross-entropy loss to multiple classes. s_j corresponds to the score (e.g. output of final layer) for each class; the fraction in the log provides a normalized probability for each class.

Equivalent to the negative log of the probability of the correct ground truth class being predicted. Think about what the expression looks like when y_i = 1 vs. 0.

**Binary Cross-Entropy**

$$L_{BCE} = \frac{1}{M}\sum_i -(y_i log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i))$$

Label is binary in {0,1}. Prediction is a real number in (0,1) and is the probability of the label being 1. It is usually the output of a sigmoid operation after the final layer.

# Common loss functions

Equivalent to the negative log of the probability of the correct ground truth class being predicted. Think about what the expression looks like when y_i = 1 vs. 0.

**Regression**

Minimize squared difference between prediction output and target

$$L_{regression} = \frac{1}{M}\sum_i (\hat{y}^i - y^i)^2$$

Label is a continuous value.

**Binary Cross-Entropy**

$$L_{BCE} = \frac{1}{M}\sum_i -(y_i log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i))$$

Label is binary in {0,1}. Prediction is a real number in (0,1) and is the probability of the label being 1. It is usually the output of a sigmoid operation after the final layer.

**Softmax**

Negative log of the probability of the true class y_i, as with the BCE loss.

$$L_{Softmax} = \frac{1}{M}\sum_i -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

Label is 1 of K classes in {0, …, K}. Extension of binary cross-entropy loss to multiple classes. s_j corresponds to the score (e.g. output of final layer) for each class; the fraction in the log provides a normalized probability for each class.

**SVM**

$$L_{SVM} = \frac{1}{M}\sum_i \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Label is 1 of K classes in {0, …, K}. Same use case as softmax, but different way of encouraging the model to produce outputs that we “like”. In practice, softmax is more popular and provides a nice probabilistic interpretation.

# Common loss functions

**Regression** Minimize squared difference between prediction output and target

$$L_{regression} = \frac{1}{M}\sum_i (\hat{y}^i - y^i)^2$$

Label is a continuous value.

**Softmax** Negative log of the probability of the true class y_i, as with the BCE loss.

$$L_{Softmax} = \frac{1}{M}\sum_i -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Label is 1 of K classes in {0, …, K}. Extension of binary cross-entropy loss to multiple classes. s_j corresponds to the score (e.g. output of final layer) for each class; the fraction in the log provides a normalized probability for each class.

Equivalent to the negative log of the probability of the correct ground truth class being predicted. Think about what the expression looks like when y_i = 1 vs. 0.

**Binary Cross-Entropy**

$$L_{BCE} = \frac{1}{M}\sum_i -(y_i log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i))$$

Label is binary in {0,1}. Prediction is a real number in (0,1) and is the probability of the label being 1. It is usually the output of a sigmoid operation after the final layer.

**SVM** Incurs lowest loss of 0 (what we want) if the score for the true class y_i is greater than the score for each incorrect class j by a margin of 1

$$L_{SVM} = \frac{1}{M}\sum_i \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Label is 1 of K classes in {0, …, K}. Same use case as softmax, but different way of encouraging the model to produce outputs that we “like”. In practice, softmax is more popular and provides a nice probabilistic interpretation.

# Common loss functions

You will find these in tensorflow!

In Keras:

**mean_squared_error**

```
keras.losses.mean_squared_error(y_true, y_pred)
```

→ Mean squared error (MSE) is another name for regression loss

**categorical_crossentropy**

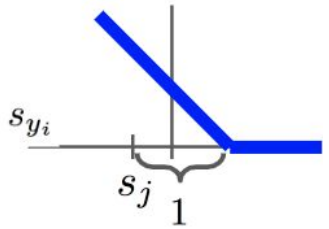
```
keras.losses.categorical_crossentropy(y_true, y_pred, from_logits=False, label_smoothing=0)
```

→ Covers both BCE and Softmax loss (remember softmax is a multiclass extension of BCE)

**hinge**

```
keras.losses.hinge(y_true, y_pred)
```

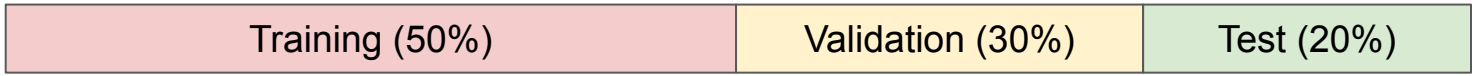
→ Hinge is another name for SVM loss, due to the loss function shape.




https://keras.io/losses/

# Data Considerations for Image Classification Models

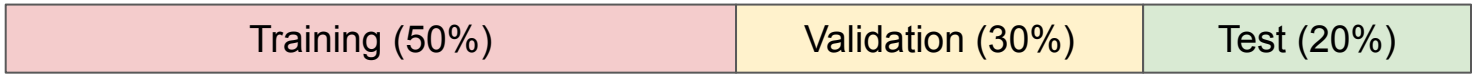
# Training, validation, and test sets




Held-out evaluation set for selecting best hyperparameters during training

Do not use until final evaluation

# Training, validation, and test sets




Held-out evaluation set for selecting best hyperparameters during training

Do not use until final evaluation

Other splits e.g. 60/20/20 also popular.
Balance sufficient data for training vs. informative performance estimate on validation / testing.

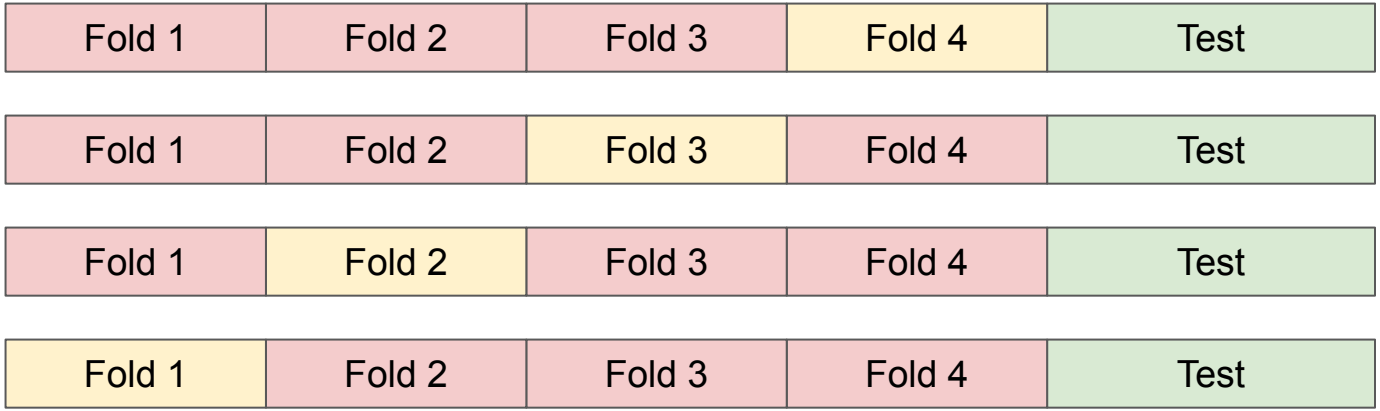
# Maximizing training data for the final model


Once hyperparameters are selected using the validation set, common to merge training and validation sets into a larger “trainval” set to train a final model using the hyperparameters.

OK since we can use non-test data however we want during model development!

# K-fold cross validation: for small datasets

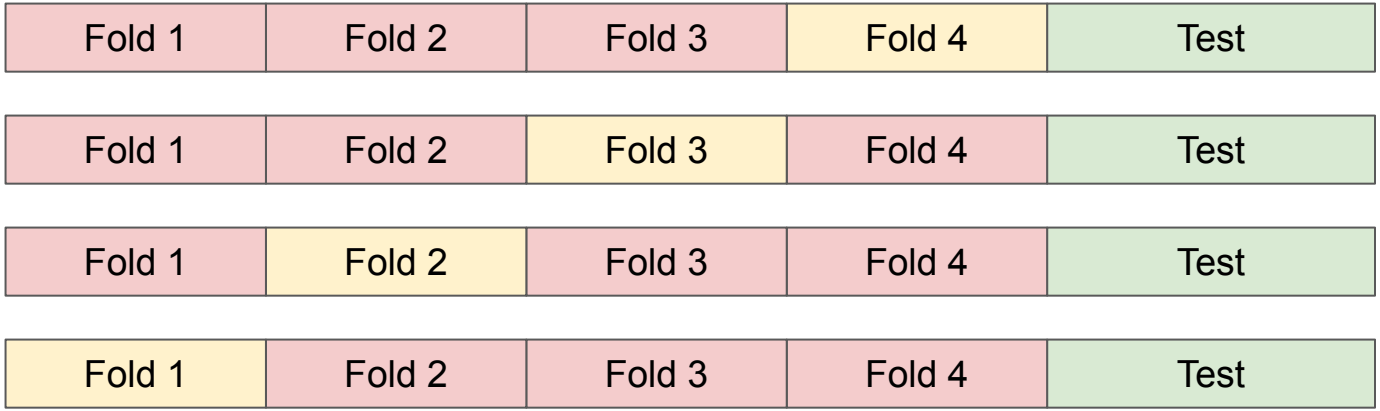
Sometimes we have small labeled datasets in healthcare… in this case K-fold cross validation (which is more computationally expensive) may be worthwhile.




Train model K times with a different fold as the validation set each time; then average the validation set results. Allows more data to be used for each training of the model, while still using enough data to get accurate validation result.

# K-fold cross validation: for small datasets

Sometimes we have small labeled datasets in healthcare… in this case K-fold cross validation (which is more computationally expensive) may be worthwhile.




Train model K times with a different fold as the validation set each time; then average the validation set results. Allows more data to be used for each training of the model, while still using enough data to get accurate validation result.

Can also apply same concept to test-time evaluation.

# Data preprocessing

**Min-max scaling:**

x_scaled = (x_orig - x_min) / (x_max - x_min)

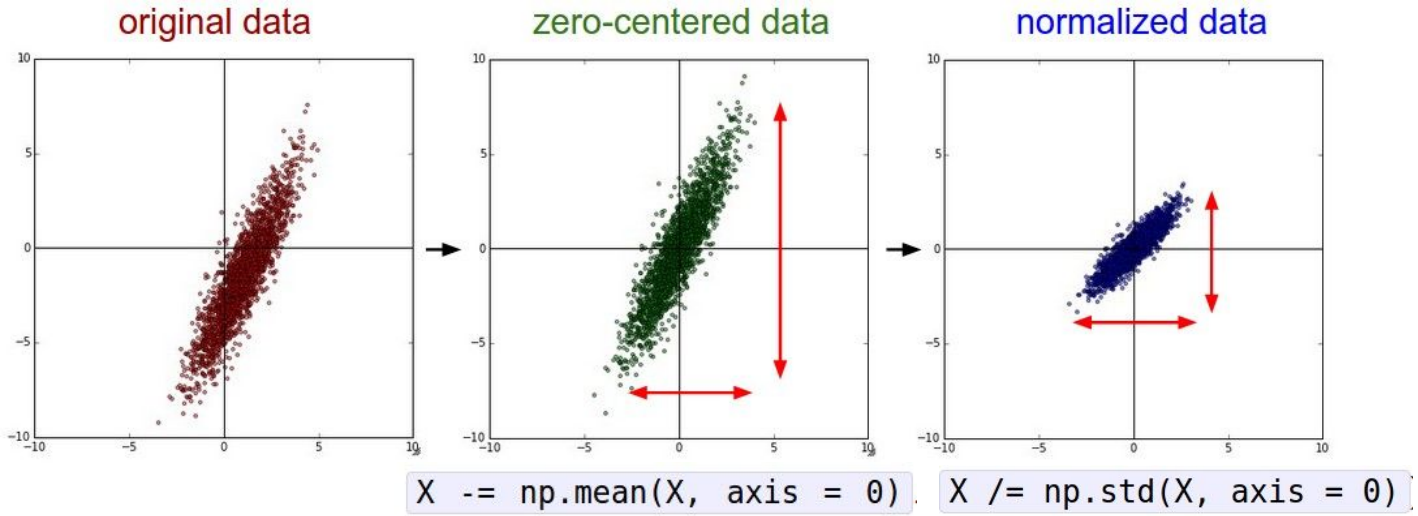
where x_min and x_max are min and max values in the original data

- Maps original range of data to [0,1] range
- Neural networks generally expect small numbers as input (not too extreme relative to scale of initialized weights)

Slide credit: CS231n

# Data preprocessing

Common to also normalize mean and variance of features, such that features are treated equally. Most common: make all features zero-mean, unit variance.




Slide credit: CS231n

# Data preprocessing: for images

For images, common to perform simpler normalization:

e.g. consider a dataset with [32,32,3] images

- Subtract the mean image (used in original AlexNet model)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (used in original VGG model)
  (mean along each channel = 3 numbers)
- Subtract per-channel mean and
  divide by per-channel std (used in original ResNet model)
  (mean along each channel = 3 numbers)

Slide credit: CS231n

# Evaluating image classification models

# Evaluation metrics

**Confusion matrix**

| Ground Truth \ Prediction | 0 | 1 |
|---|---|---|
| 0 | TN | FP |
| 1 | FN | TP |

**Accuracy:** (TP + TN) / total

# Evaluation metrics

**Confusion matrix**

| Ground Truth \ Prediction | 0 | 1 |
|---|---|---|
| 0 | TN | FP |
| 1 | FN | TP |

**Accuracy:** (TP + TN) / total

Q: When might evaluating purely accuracy be problematic?

# Evaluation metrics

**Confusion matrix**

| Ground Truth \ Prediction | 0 | 1 |
|---|---|---|
| 0 | TN | FP |
| 1 | FN | TP |

**Accuracy:** (TP + TN) / total

Q: When might evaluating purely accuracy be problematic?

A: Imbalanced datasets.

# Evaluation metrics

**Confusion matrix**

| Ground Truth \ Prediction | 0 | 1 |
|---|---|---|
| 0 | TN | FP |
| 1 | FN | TP |

**Accuracy:** (TP + TN) / total

**Sensitivity / Recall** (true positive rate)**:**
TP / total positives

**Specificity** (true negative rate)**:**
TN / total negatives

**Precision** (positive predictive value)**:**
TP / total predicted positives

**Negative predictive value:**
TN / total predicted negatives

# Evaluation metrics

We can trade-off different values of these metrics as we vary our classifier's score threshold to predict a positive

**Confusion matrix**

| Ground Truth \ Prediction | 0 | 1 |
|---|---|---|
| 0 | TN | FP |
| 1 | FN | TP |

**Accuracy:** (TP + TN) / total

**Sensitivity / Recall** (true positive rate)**:**
TP / total positives

**Specificity** (true negative rate)**:**
TN / total negatives

**Precision** (positive predictive value)**:**
TP / total predicted positives

**Negative predictive value:**
TN / total predicted negatives

# Evaluation metrics

Q: As prediction threshold increases, how does that generally affect sensitivity? Specificity?

**Confusion matrix**

| Ground Truth | Prediction 0 | Prediction 1 |
|---|---|---|
| 0 | TN | FP |
| 1 | FN | TP |

**Accuracy:** (TP + TN) / total

**Sensitivity / Recall** (true positive rate)**:**
TP / total positives

**Specificity** (true negative rate)**:**
TN / total negatives

**Precision** (positive predictive value)**:**
TP / total predicted positives

**Negative predictive value:**
TN / total predicted negatives

# Evaluation metrics

Q: As prediction threshold increases, how does that generally affect sensitivity? Specificity?
A: Sensitivity goes down, specificity up

**Confusion matrix**

| Ground Truth \ Prediction | 0 | 1 |
|---|---|---|
| 0 | TN | FP |
| 1 | FN | TP |

**Accuracy:** (TP + TN) / total

**Sensitivity / Recall** (true positive rate)**:**
TP / total positives

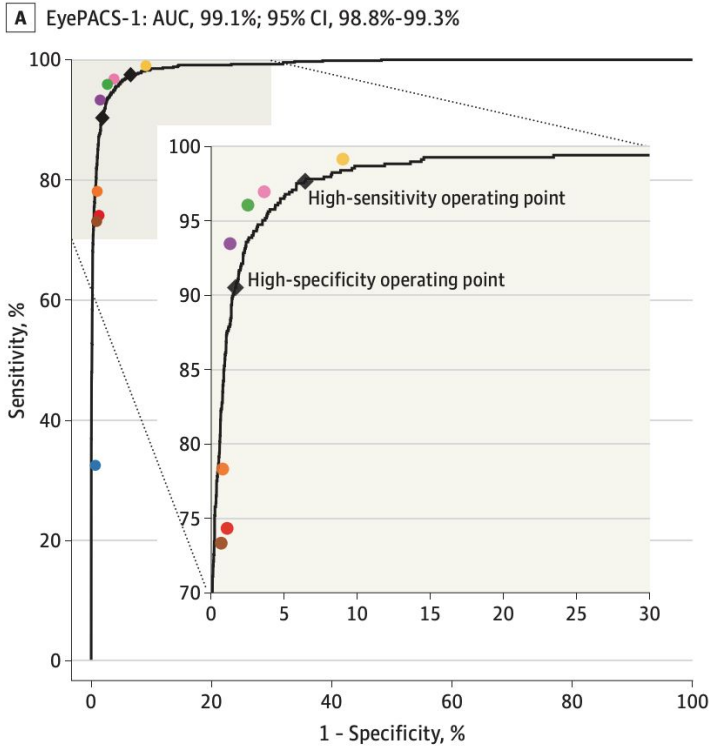
**Specificity** (true negative rate)**:**
TN / total negatives

**Precision** (positive predictive value)**:**
TP / total predicted positives

**Negative predictive value:**
TN / total predicted negatives

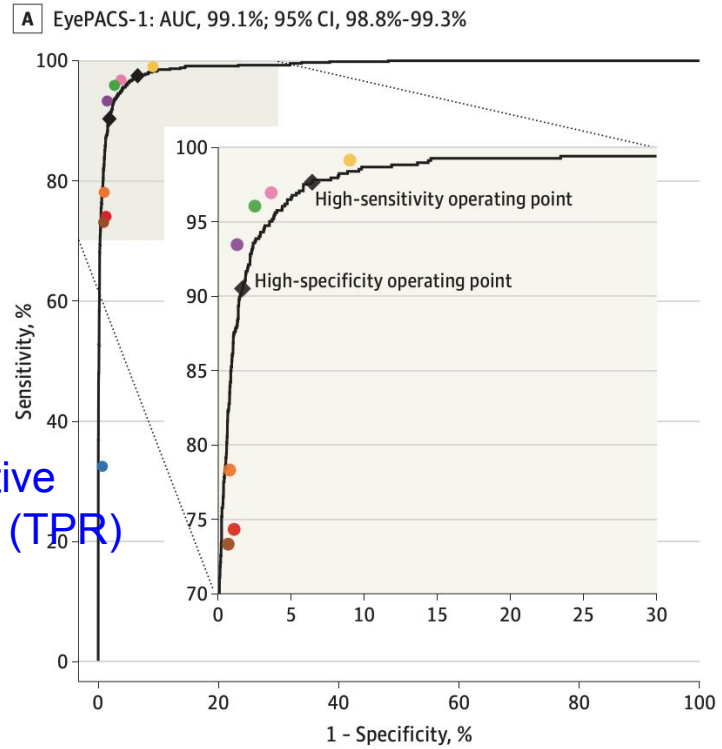
# Evaluation metrics

- **Receiver Operating Characteristic (ROC) curve**:
  - Plots sensitivity and specificity (specifically, 1 - specificity) as prediction threshold is varied
  - Gives trade-off between sensitivity and specificity
  - Also report summary statistic AUC (area under the curve)



# Evaluation metrics

- **Receiver Operating Characteristic (ROC) curve**:
  - Plots sensitivity and specificity (specifically, 1 - specificity) as prediction threshold is varied
  - Gives trade-off between sensitivity and specificity
  - Also report summary statistic AUC (area under the curve)



# Evaluation metrics

- Sometimes also see **precision recall curve**
  - More informative when dataset is heavily imbalanced (specificity = true negative rate less meaningful in this case)

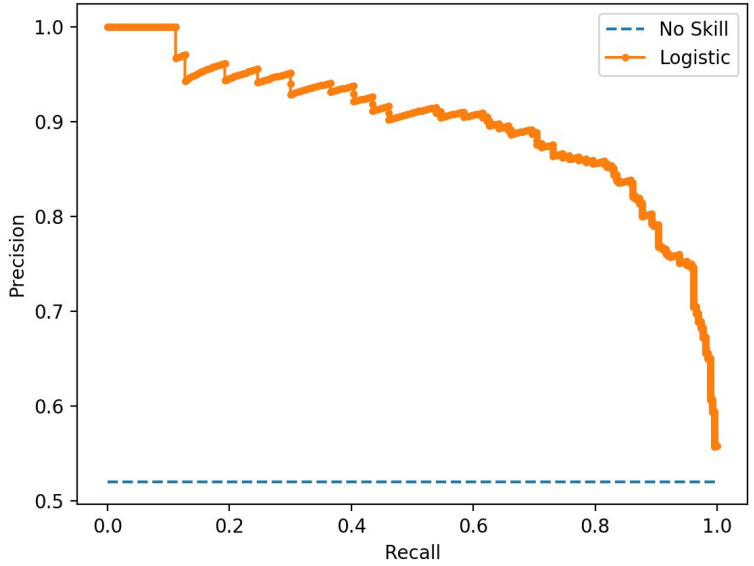



Figure credit: https://3qeqpr26caki16dnhd19sv6by6v-wpengine.netdna-ssl.com/wp-content/uploads/2018/08/Precision-Recall-Plot-for-a-No-Skill-Classifier-and-a-Logistic-Regression-Model4.png

# Evaluation metrics

- Selecting optimal trade-off points
  - Maximize **Youden's Index**
    - J = sensitivity + specificity - 1
    - Gives equal weight to optimizing true positives and true negatives

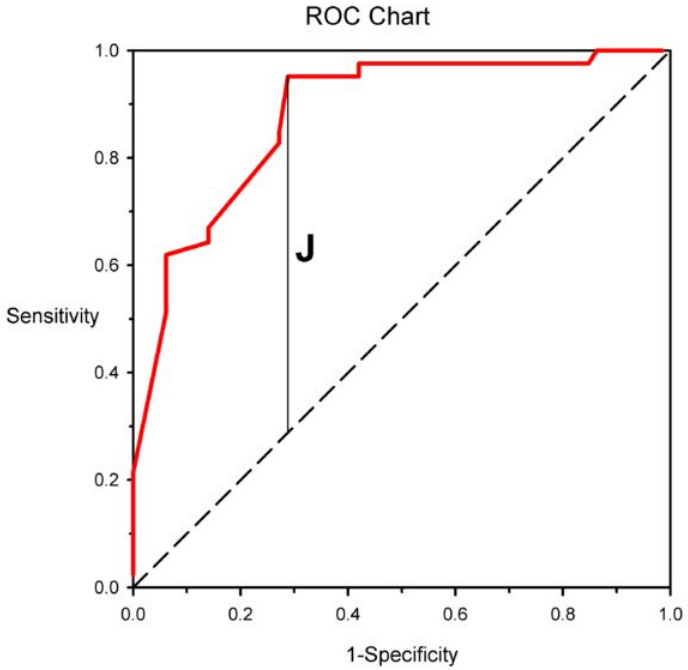



Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

# Evaluation metrics

- Selecting optimal trade-off points
  - Maximize **Youden's Index**
    - J = sensitivity + specificity - 1
    - Gives equal weight to optimizing true positives and true negatives

Also equal to distance above chance line for a balanced dataset: sensitivity - (1 - specificity) = sensitivity + specificity - 1

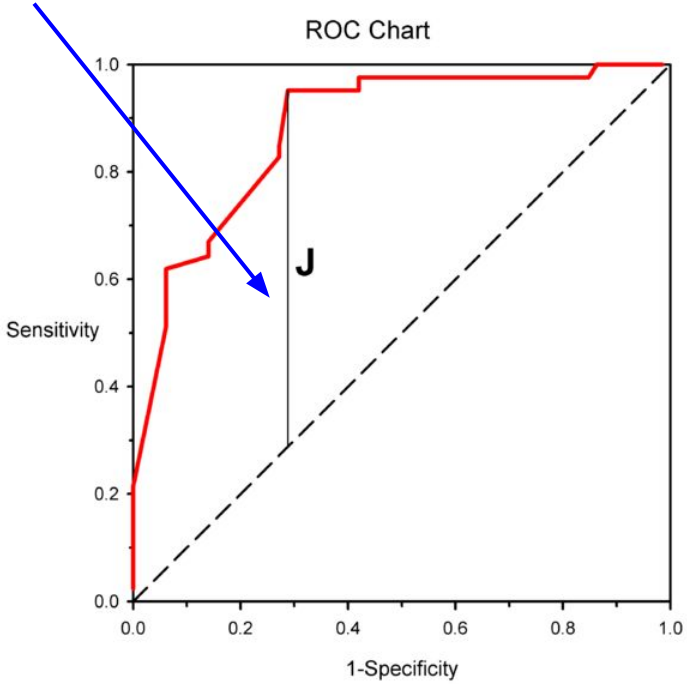



Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

# Evaluation metrics

- Selecting optimal trade-off points
  - Maximize **Youden's Index**
    - J = sensitivity + specificity - 1
    - Gives equal weight to optimizing true positives and true negatives
  - Sometimes also see F-measure (or F1 score)
    - F1 = 2*(precision*recall) / (precision + recall)
    - Harmonic mean of precision and recall

Also equal to distance above chance line for a balanced dataset: sensitivity - (1 - specificity) = sensitivity + specificity - 1

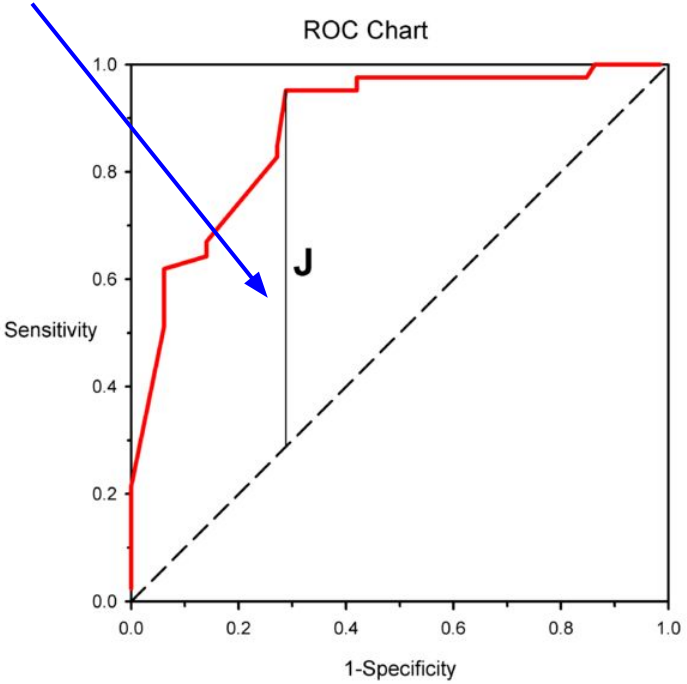



Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

# Evaluation metrics

- Selecting optimal trade-off points
  - Maximize **Youden's Index**
    - J = sensitivity + specificity - 1
    - Gives equal weight to optimizing true positives and true negatives
  - Sometimes also see F-measure (or F1 score)
    - F1 = 2*(precision*recall) / (precision + recall)
    - Harmonic mean of precision and recall

But selected trade-off points could also depend on application

Also equal to distance above chance line for a balanced dataset: sensitivity - (1 - specificity) = sensitivity + specificity - 1

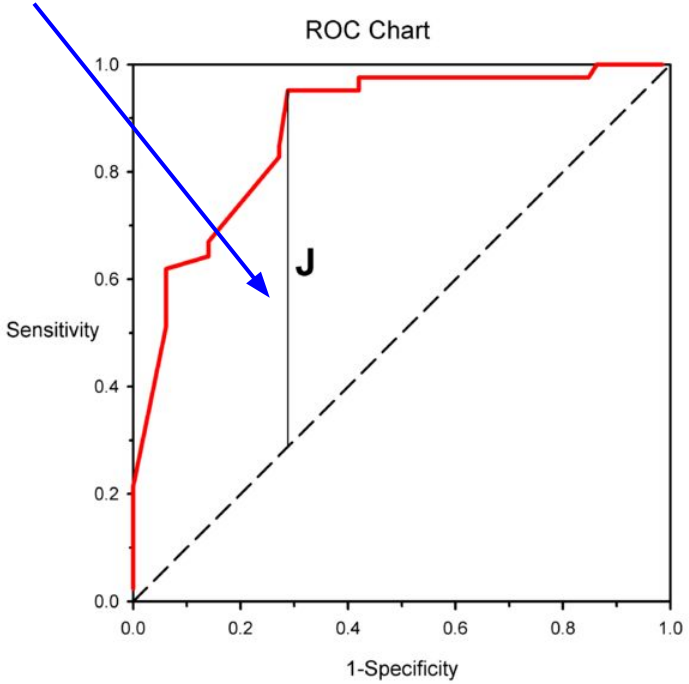


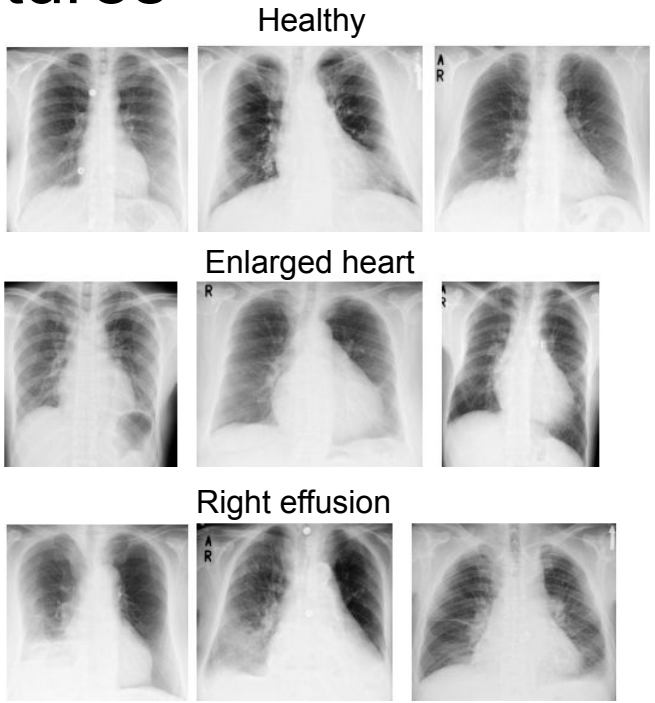

Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

# Case Studies of CNNs for Medical Imaging Classification

# Early steps of deep learning in medical imaging: using ImageNet CNN features

Bar et al. 2015

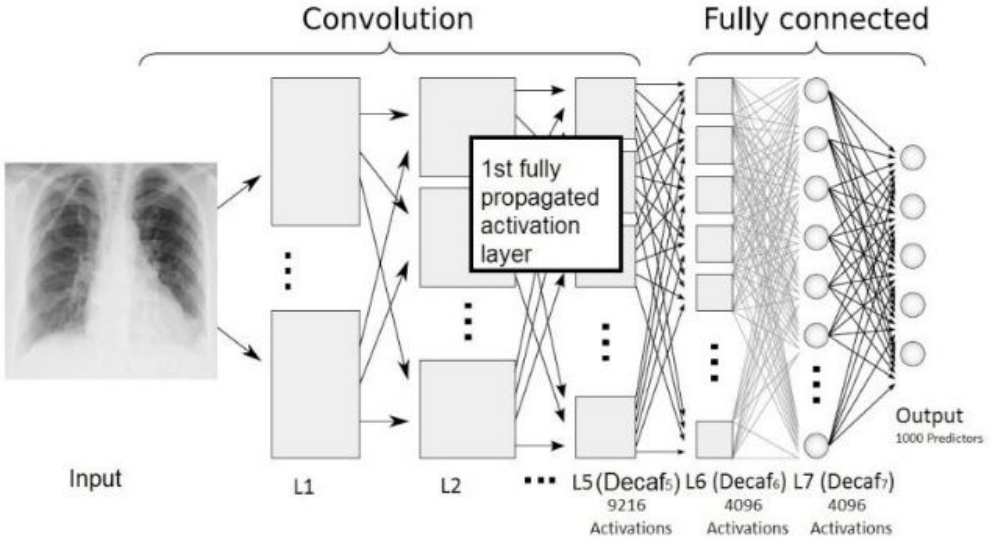
- Input: Chest **x-ray images**
- Output: Several binary classification tasks
  - Right pleural effusion or not
  - Enlarged heart or not
  - Healthy or abnormal
- Very small dataset: 93 frontal chest x-ray images




Bar et al. Deep learning with non-medical training used for chest pathology identification. SPIE, 2015.

# Bar et al. 2015

- Did not train a deep learning model on the medical data
- Instead, extracted features from an AlexNet trained on ImageNet
  - 5th, 6th, and 7th layers
- Used extracted features with an SVM classifier
- Performed zero-mean unit-variance normalization of all features
- Evaluated combination with other hand-crafted image features

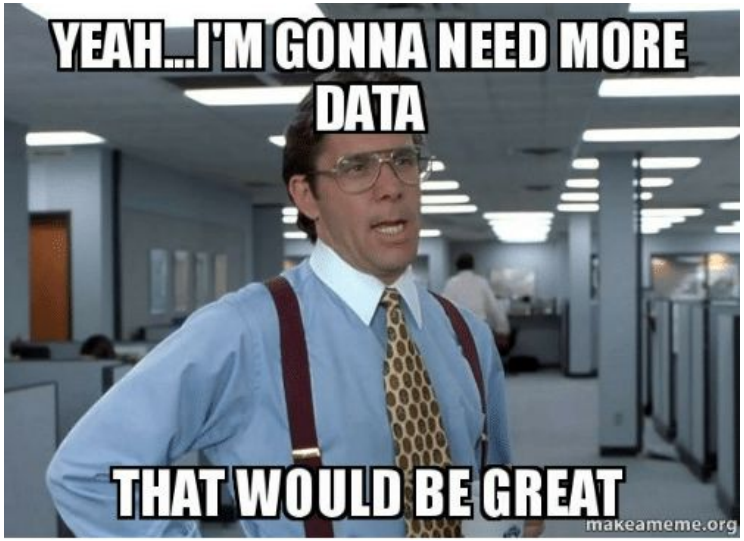



Bar et al. Deep learning with non-medical training used for chest pathology identification. SPIE, 2015.

# How much data do you need for deep learning?

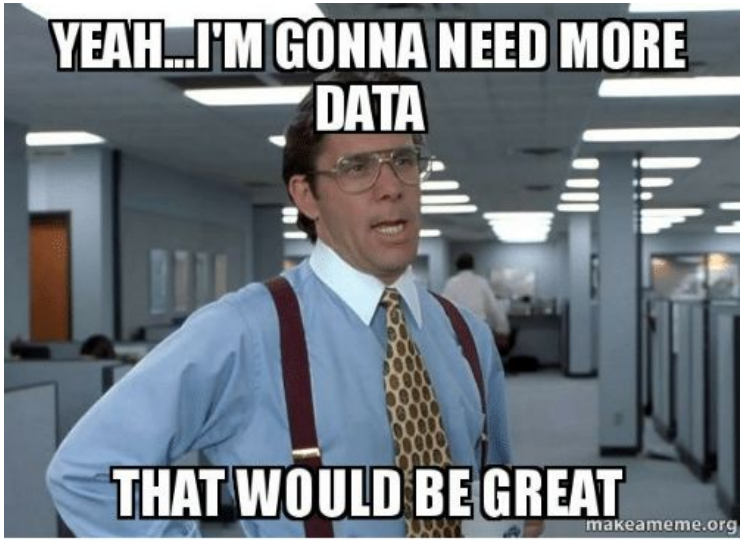
# How much data do you need for deep learning?

A: A lot.



# How much data do you need for deep learning?

A: A lot.

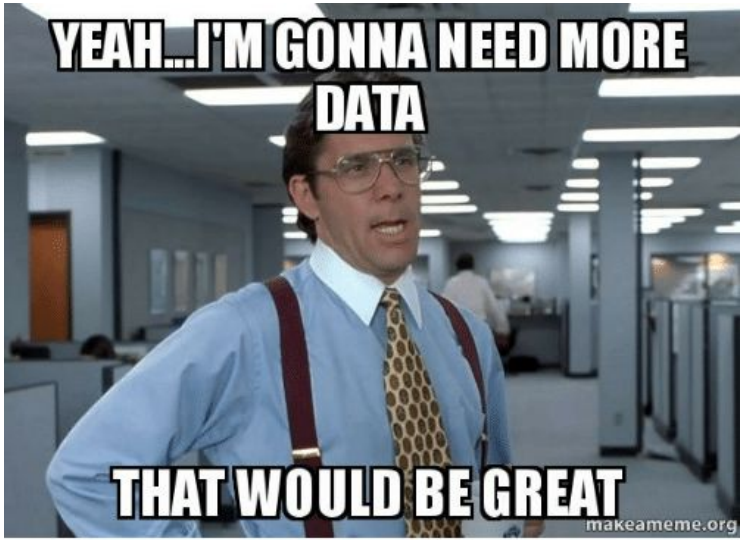



Premise of deep learning uses many parameters (e.g. millions) to fit complex functions -> if the dataset is too small, easiest solution that model ends up learning can be overfitting to memorizing the labels of the training examples

# How much data do you need for deep learning?

A: A lot.

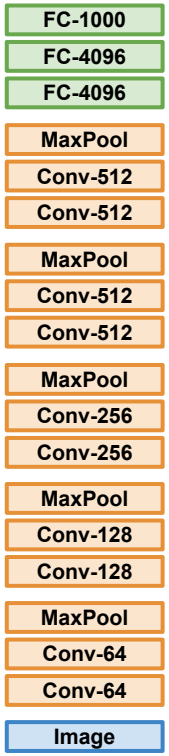
ImageNet dataset consists of 1M images: 1000 classes with 1000 images each




Premise of deep learning uses many parameters (e.g. millions) to fit complex functions -> if the dataset is too small, easiest solution that model ends up learning can be overfitting to memorizing the labels of the training examples

# Transfer learning: amplifying training data

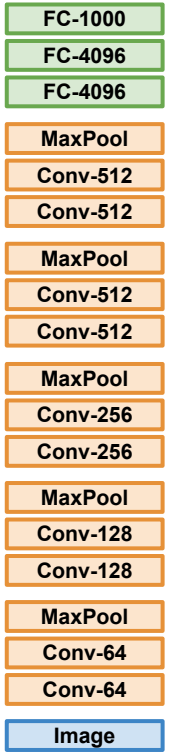
1. Train on big dataset
   (e.g. ImageNet)



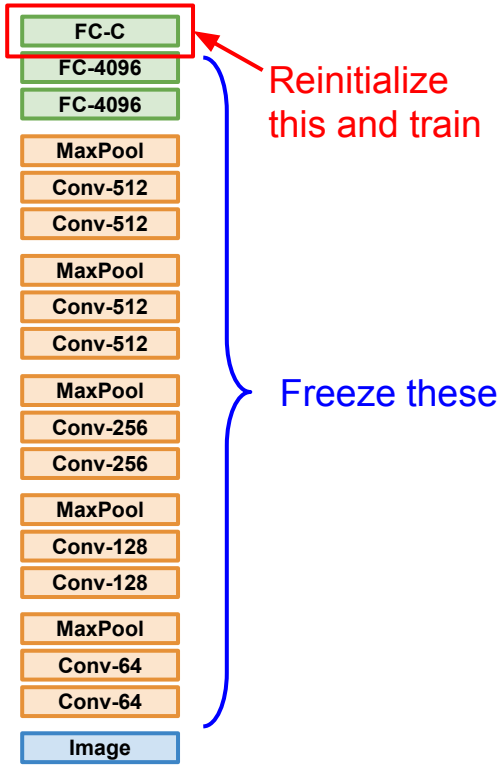

Slide credit: CS231n

# Transfer learning: amplifying training data

1. Train on big dataset
(e.g. ImageNet)




2. Small Dataset (C classes)



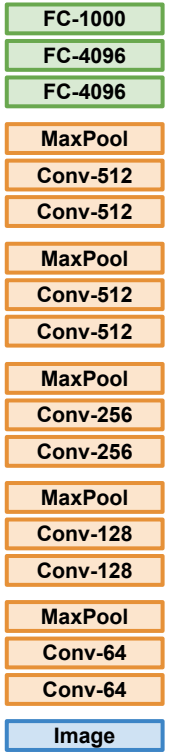

Reinitialize this and train

Freeze these

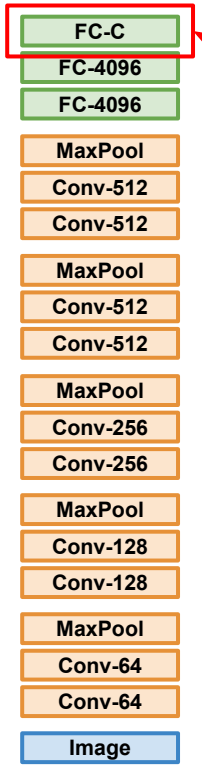
Slide credit: CS231n

# Transfer learning: amplifying training data

1. Train on big dataset (e.g. ImageNet)



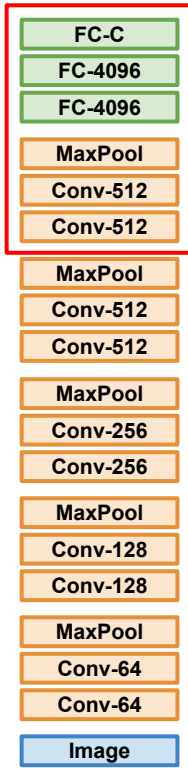

2. Small Dataset (C classes)




Reinitialize this and keep training on target dataset

Freeze these

3. Bigger dataset




With bigger dataset, train more layers

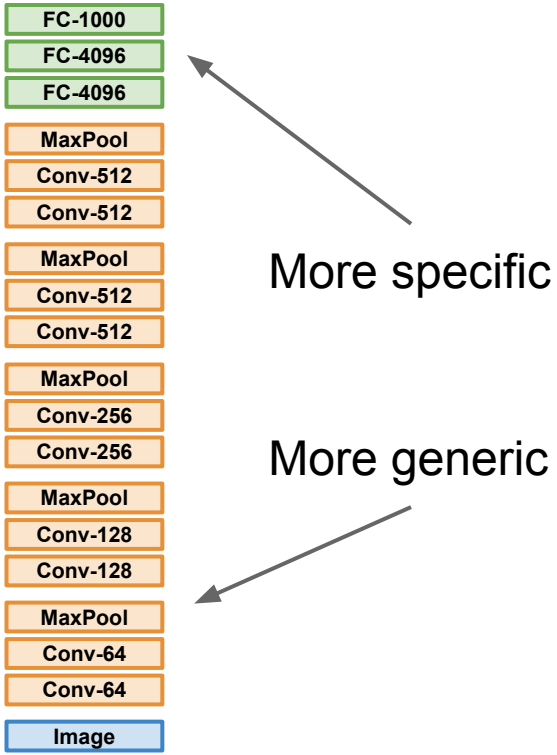
Train these

Lower learning rate when finetuning; 1/10 of original LR is good starting point

Freeze these

Slide credit: CS231n

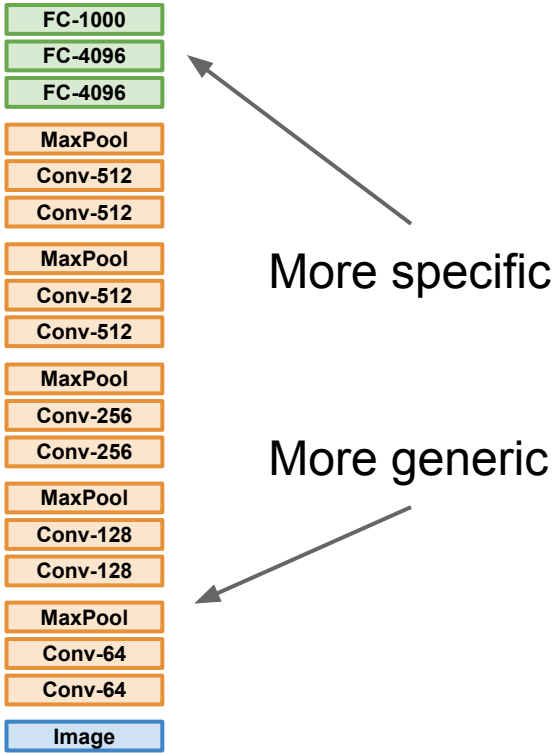
## Transfer learning from a large dataset to your dataset...




| | very similar dataset | very different dataset |
|---|---|---|
| **very little data** | | |
| **quite a lot of data** | | |

Slide credit: CS231n

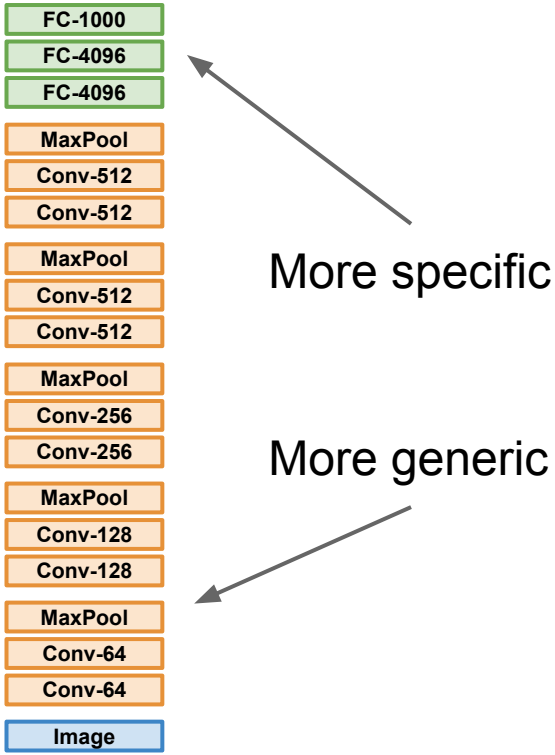
## Transfer learning from a large dataset to your dataset...




| | very similar dataset | very different dataset |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | |
| **quite a lot of data** | | |

Slide credit: CS231n

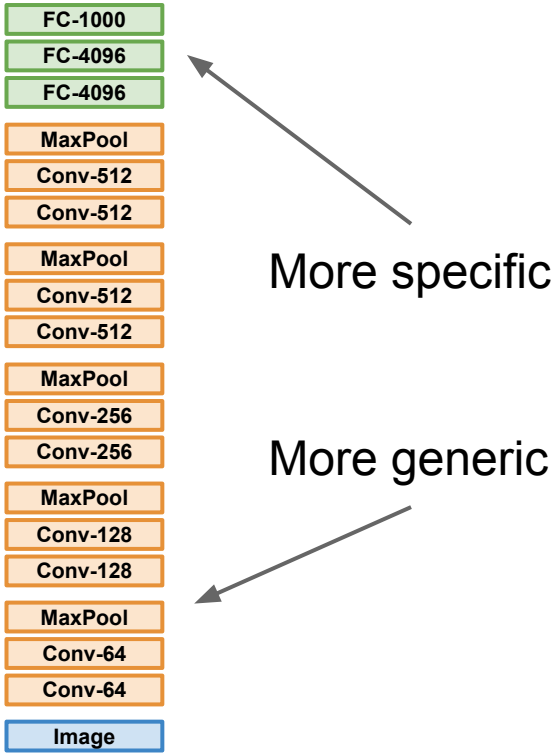
## Transfer learning from a large dataset to your dataset...




| | very similar dataset | very different dataset |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You're in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | | |

Slide credit: CS231n

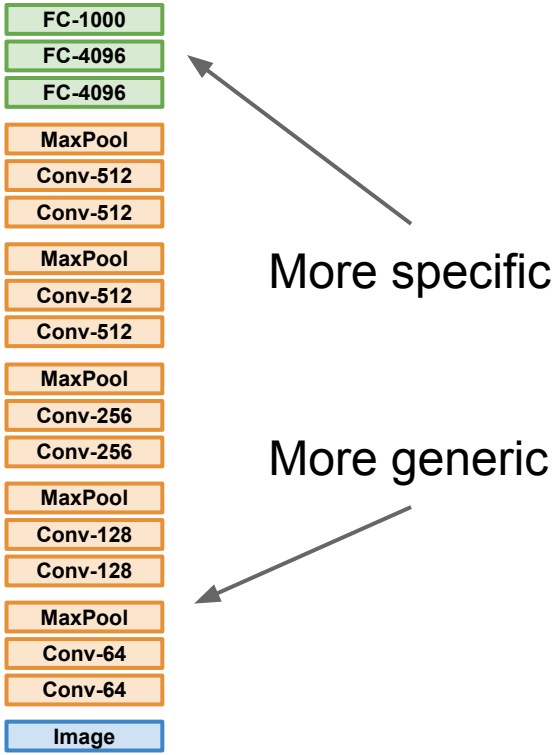
## Transfer learning from a large dataset to your dataset...




| | very similar dataset | very different dataset |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You're in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | |

Slide credit: CS231n

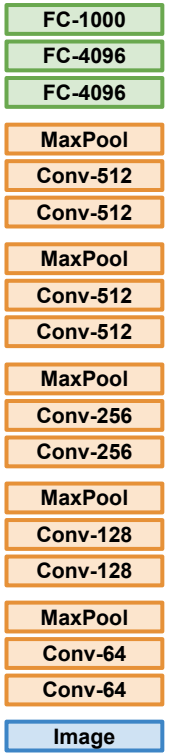
## Transfer learning from a large dataset to your dataset...




| | very similar dataset | very different dataset |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You're in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | Finetune a large number of layers |

Slide credit: CS231n

## Transfer learning from a large dataset to your dataset...




| | very similar dataset | very different dataset |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You're in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | Finetune a large number of layers |

Often good idea to try this first, try fine-tuning all layers of the network

Slide credit: CS231n

# How much data do you need for deep learning?

**Examples per class of your dataset**, in addition to transfer learning (take this with grain of salt, it really depends on the problem):

- Low dozens: generally too small to learn a meaningful model, using standard supervised deep learning
- High dozens to low hundreds: may see models with some predictive ability, unlikely to really wow or be “superhuman” though
- High hundreds to thousands: “happy regime” for deep learning

| | **very similar dataset** | **very different dataset** |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You’re in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | Finetune a large number of layers |

# How much data do you need for deep learning?

**Examples per class of your dataset**, in addition to transfer learning (take this with grain of salt, it really depends on the problem):

- Low dozens: generally too small to learn a meaningful model, using standard supervised deep learning
- High dozens to low hundreds: may see models with some predictive ability, unlikely to really wow or be “superhuman” though
- High hundreds to thousands: “happy regime” for deep learning

| | **very similar dataset** | **very different dataset** |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You’re in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | Finetune a large number of layers |

# How much data do you need for deep learning?

**Examples per class of your dataset**, in addition to transfer learning (take this with grain of salt, it really depends on the problem):

- Low dozens: generally too small to learn a meaningful model, using standard supervised deep learning
- High dozens to low hundreds: may see models with some predictive ability, unlikely to really wow or be “superhuman” though
- High hundreds to thousands: “happy regime” for deep learning

| | **very similar dataset** | **very different dataset** |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You’re in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | Finetune a large number of layers |

# How much data do you need for deep learning?

**Examples per class of your dataset**, in addition to transfer learning (take this with grain of salt, it really depends on the problem):

- Low dozens: generally too small to learn a meaningful model, using standard supervised deep learning
- High dozens to low hundreds: may see models with some predictive ability, unlikely to really wow or be “superhuman” though
- High hundreds to thousands: “happy regime” for deep learning

In general, deep learning is data hungry -- the more data the better

| | **very similar dataset** | **very different dataset** |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You’re in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | Finetune a large number of layers |

# How much data do you need for deep learning?

**Examples per class of your dataset**, in addition to transfer learning (take this with grain of salt, it really depends on the problem):

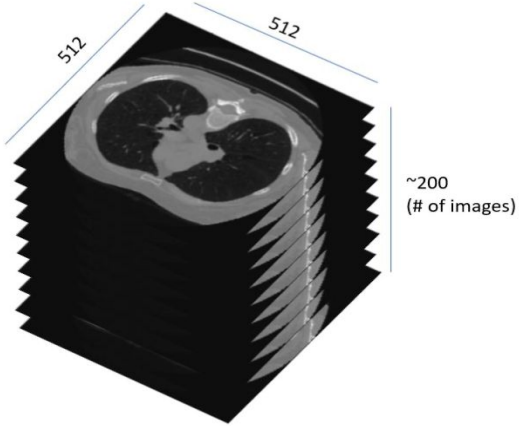
- Low dozens: generally too small to learn a meaningful model, using standard supervised deep learning
- High dozens to low hundreds: may see models with some predictive ability, unlikely to really wow or be “superhuman” though
- High hundreds to thousands: “happy regime” for deep learning

| | **very similar dataset** | **very different dataset** |
|---|---|---|
| **very little data** | Use Linear Classifier on top layer features | You’re in trouble… Try linear classifier on different layer features |
| **quite a lot of data** | Finetune a few layers | Finetune a large number of layers |

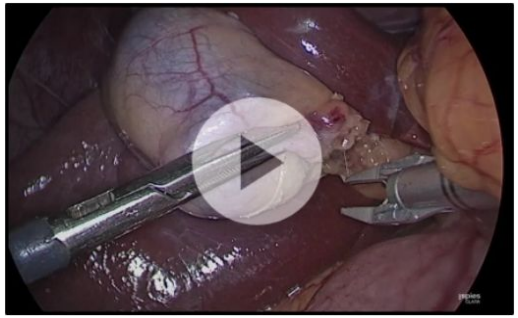
In general, deep learning is data hungry -- the more data the better

Almost always leverage transfer learning unless you have extremely different or huge (e.g. ImageNet-scale) dataset

# What counts as a data example?

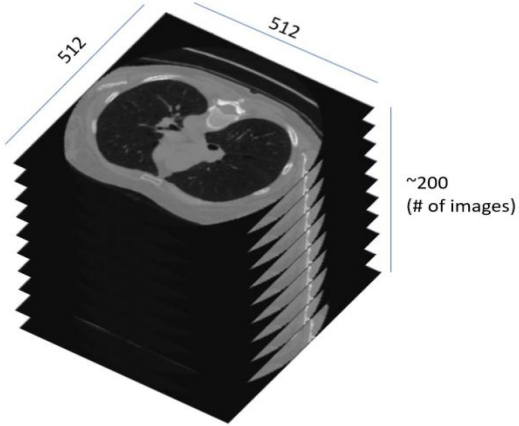



1 3D CT volume with 200 slices ≠ 200 data examples

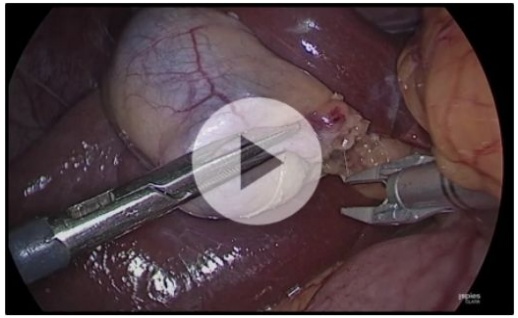


5 surgery videos with thousands of frames each ≠ thousands of data examples

# What counts as a data example?




1 3D CT volume with 200 slices ≠ 200 data examples

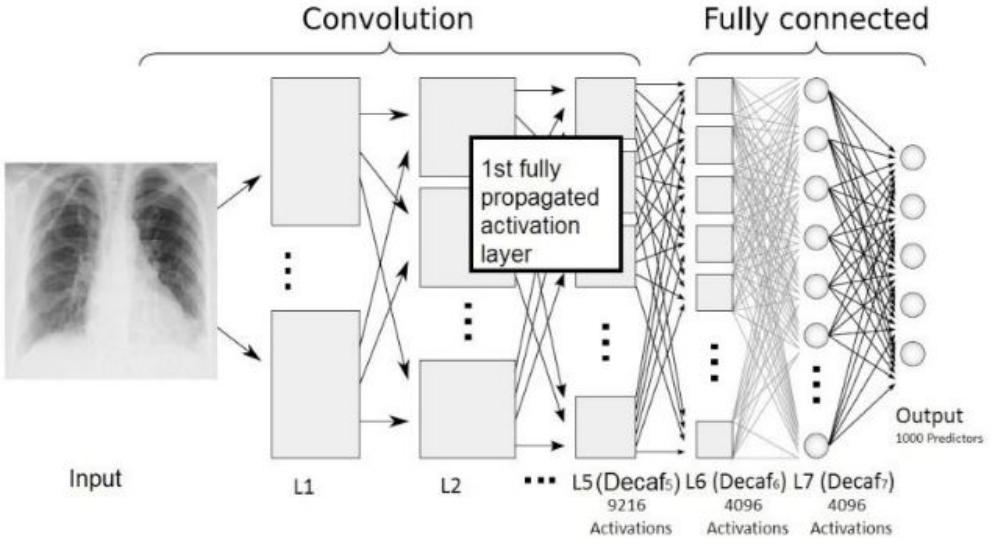


5 surgery videos with thousands of frames each ≠ thousands of data examples

Guidelines for amount of training data refers to # of unique instances representative of diversity expected during testing / deployment. E.g. # of independent CT scans or surgery videos. Additional correlated data (e.g. different slices of the same tumor or different suturing instances within the same video) provide relatively less incremental value in comparison.

# Bar et al. 2015

- Did not train a deep learning model on the medical data
- Instead, extracted features from an AlexNet trained on ImageNet
  - 5th, 6th, and 7th layers
- Used extracted features with an SVM classifier
- Performed zero-mean unit-variance normalization of all features
- Evaluated combination with other hand-crafted image features

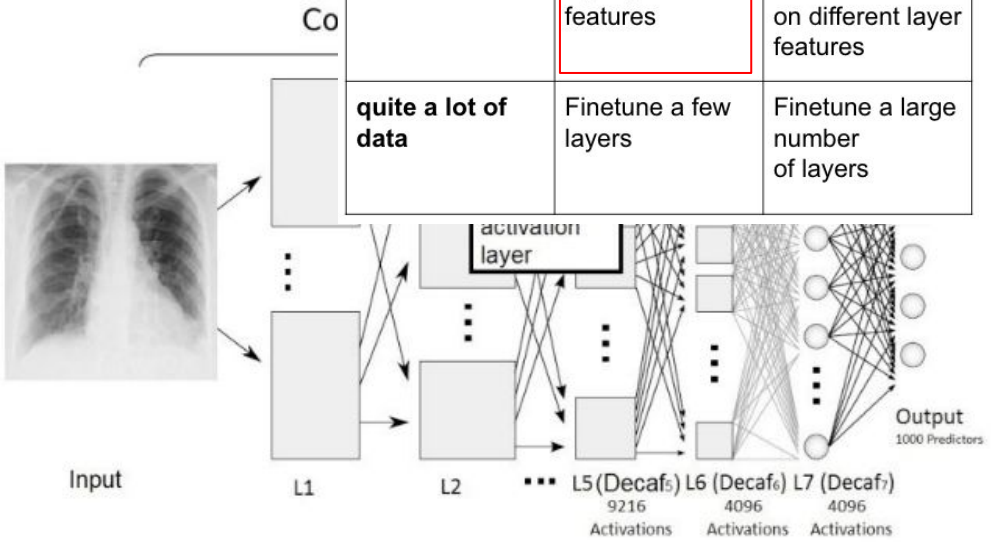



Bar et al. Deep learning with non-medical training used for chest pathology identification. SPIE, 2015.

# Bar et al. 2015

- Did not train a deep learning model on the medical data
- Instead, extracted features from an AlexNet trained on ImageNet
  - 5th, 6th, and 7th layers
- Used extracted features with an SVM classifier
- Performed zero-mean unit-variance normalization of all features
- Evaluated combination with other hand-crafted image features

| | very similar dataset | very different dataset |
|---|---|---|
| very little data | Use Linear Classifier on top layer features | You're in trouble… Try linear classifier on different layer features |
| quite a lot of data | Finetune a few layers | Finetune a large number of layers |




Bar et al. Deep learning with non-medical training used for chest pathology identification. SPIE, 2015.

# Bar et al. 2015

Table 1. Right Pleural Effusion Condition.

| | Low Level | | High Level | Deep | | | Fusion |
|---|---|---|---|---|---|---|---|
| | LBP | GIST | PiCoDes | Decaf L5 | Decaf L6 | Decaf L7 | PiCoDes+Decaf L5 |
| **Sensitivity** | 0.71 | 0.79 | 0.79 | 0.93 | 0.86 | 0.86 | **0.93** |
| **Specificity** | 0.77 | 0.92 | 0.91 | 0.84 | 0.86 | 0.80 | **0.84** |
| **AUC** | 0.75 | 0.93 | 0.91 | 0.92 | 0.91 | 0.84 | **0.93** |

Table 2. Healthy vs. Pathology.

| | Low Level | | High Level | Deep | | | Fusion |
|---|---|---|---|---|---|---|---|
| | LBP | GIST | PiCoDes | Decaf L5 | Decaf L6 | Decaf L7 | PiCoDes+Decaf L5 |
| **Sensitivity** | 0.65 | 0.68 | 0.59 | 0.73 | 0.89 | 0.76 | **0.81** |
| **Specificity** | 0.61 | 0.66 | 0.79 | 0.80 | 0.64 | 0.64 | **0.79** |
| **AUC** | 0.63 | 0.72 | 0.72 | 0.78 | 0.79 | 0.72 | **0.79** |

Table 3. Enlarged Heart Condition.

| | Low Level | | High Level | Deep | | | Fusion |
|---|---|---|---|---|---|---|---|
| | LBP | GIST | PiCoDes | Decaf L5 | Decaf L6 | Decaf L7 | PiCoDes+Decaf L5 |
| **Sensitivity** | 0.75 | 0.79 | 0.79 | 0.88 | 0.79 | 0.79 | **0.83** |
| **Specificity** | 0.78 | 0.81 | 0.84 | 0.78 | 0.88 | 0.77 | **0.84** |
| **AUC** | 0.80 | 0.82 | 0.87 | 0.87 | 0.84 | 0.79 | **0.89** |

Bar et al. Deep learning with non-medical training used for chest pathology identification. SPIE, 2015.

# Bar et al. 2015

Q: How might we interpret the AUC vs. CNN feature trends?

Table 1. Right Pleural Effusion Condition.

| | Low Level | | High Level | Deep | | | Fusion |
|---|---|---|---|---|---|---|---|
| | LBP | GIST | PiCoDes | Decaf L5 | Decaf L6 | Decaf L7 | PiCoDes+Decaf L5 |
| **Sensitivity** | 0.71 | 0.79 | 0.79 | 0.93 | 0.86 | 0.86 | **0.93** |
| **Specificity** | 0.77 | 0.92 | 0.91 | 0.84 | 0.86 | 0.80 | **0.84** |
| **AUC** | 0.75 | 0.93 | 0.91 | 0.92 | 0.91 | 0.84 | **0.93** |

Table 2. Healthy vs. Pathology.

| | Low Level | | High Level | Deep | | | Fusion |
|---|---|---|---|---|---|---|---|
| | LBP | GIST | PiCoDes | Decaf L5 | Decaf L6 | Decaf L7 | PiCoDes+Decaf L5 |
| **Sensitivity** | 0.65 | 0.68 | 0.59 | 0.73 | 0.89 | 0.76 | **0.81** |
| **Specificity** | 0.61 | 0.66 | 0.79 | 0.80 | 0.64 | 0.64 | **0.79** |
| **AUC** | 0.63 | 0.72 | 0.72 | 0.78 | 0.79 | 0.72 | **0.79** |

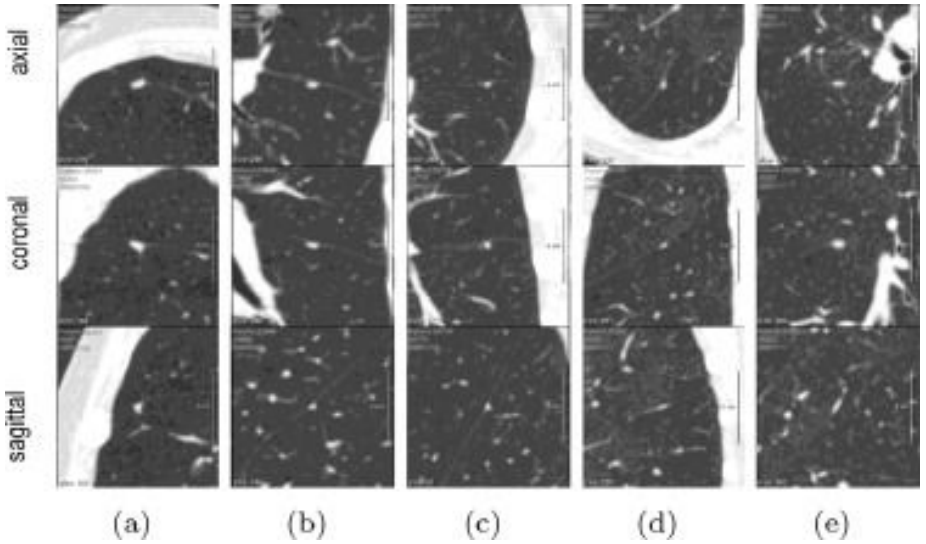
Table 3. Enlarged Heart Condition.

| | Low Level | | High Level | Deep | | | Fusion |
|---|---|---|---|---|---|---|---|
| | LBP | GIST | PiCoDes | Decaf L5 | Decaf L6 | Decaf L7 | PiCoDes+Decaf L5 |
| **Sensitivity** | 0.75 | 0.79 | 0.79 | 0.88 | 0.79 | 0.79 | **0.83** |
| **Specificity** | 0.78 | 0.81 | 0.84 | 0.78 | 0.88 | 0.77 | **0.84** |
| **AUC** | 0.80 | 0.82 | 0.87 | 0.87 | 0.84 | 0.79 | **0.89** |

Bar et al. Deep learning with non-medical training used for chest pathology identification. SPIE, 2015.

# Ciompi et al. 2015

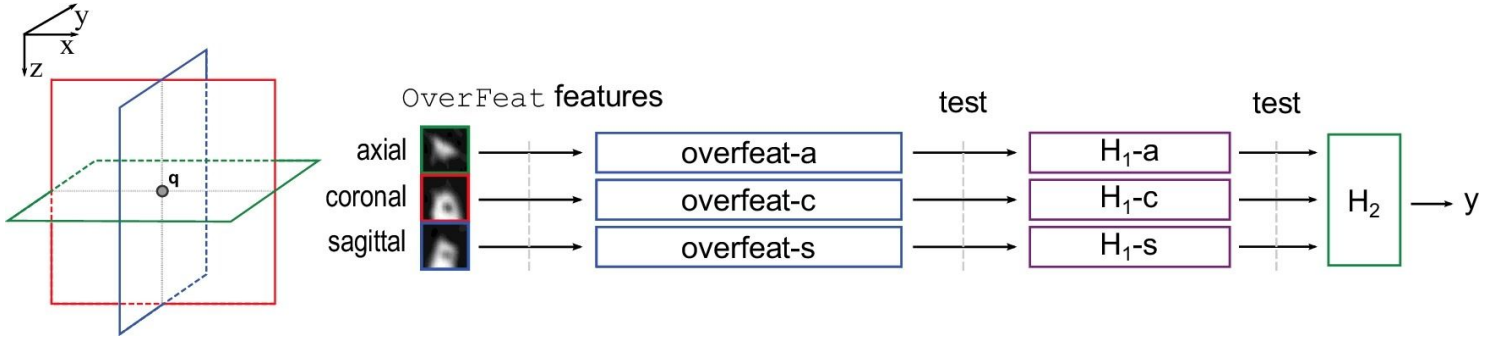
- Task: classification of lung nodules in **3D CT scans** as peri-fissural nodules (PFN, likely to be benign) or not
- Dataset: 568 nodules from 1729 scans at a single institution. (65 typical PFNs, 19 atypical PFNs, 484 non-PFNs).
- Data pre-processing: prescaling from CT hounsfield units (HU) into [0,255]. Replicate 3x across R,G,B channels to match input dimensions of ImageNet-trained CNNs.




Ciompi et al. Automatic classification of pulmonary peri-fissural nodules in computed tomography using an ensemble of 2D views and a convolutional neural network out-of-the-box. Medical Image Analysis, 2015.

# Ciompi et al. 2015

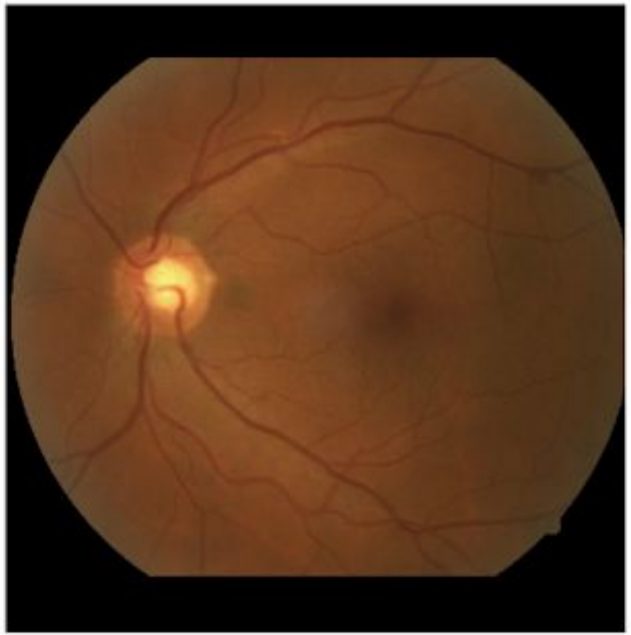
- Also extracted features from a deep learning model trained on ImageNet
  - Overfeat feature extractor (similar to AlexNet, but trained using additional losses for localization and detection)
  - To capture 3D information, extracted features from 3 different 2D views of each nodule, then input into 2-stage classifier (independent predictions on each view first, then outputs combined into second classifier).




Ciompi et al. Automatic classification of pulmonary peri-fissural nodules in computed tomography using an ensemble of 2D views and a convolutional neural network out-of-the-box. Medical Image Analysis, 2015.

# Gulshan et al. 2016

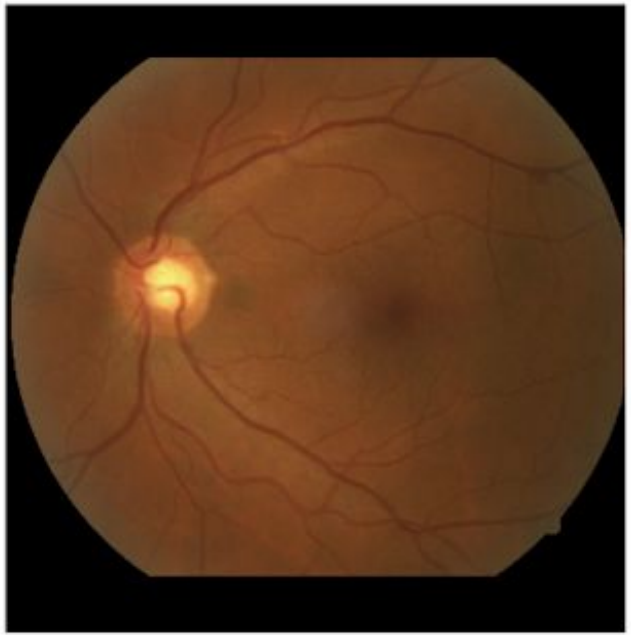
- **Task**: Binary classification of referable diabetic retinopathy from **retinal fundus photographs**
- **Input**: Retinal fundus photographs
- **Output**: Binary classification of referable diabetic retinopathy (y in {0,1})
  - Defined as moderate and worse diabetic retinopathy, referable diabetic macular edema, or both



Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

# Gulshan et al. 2016

- **Dataset**:
  - 128,175 images, each graded by 3-7 ophthalmologists.
  - 54 total graders, each paid to grade between 20 to 62508 images.
- **Data preprocessing**:
  - Circular mask of each image was detected and rescaled to be 299 pixels wide
- **Model**:
  - Inception-v3 CNN, with ImageNet pre-training
  - Multiple BCE losses corresponding to different binary prediction problems, which were then used for final determination of referable diabetic retinopathy

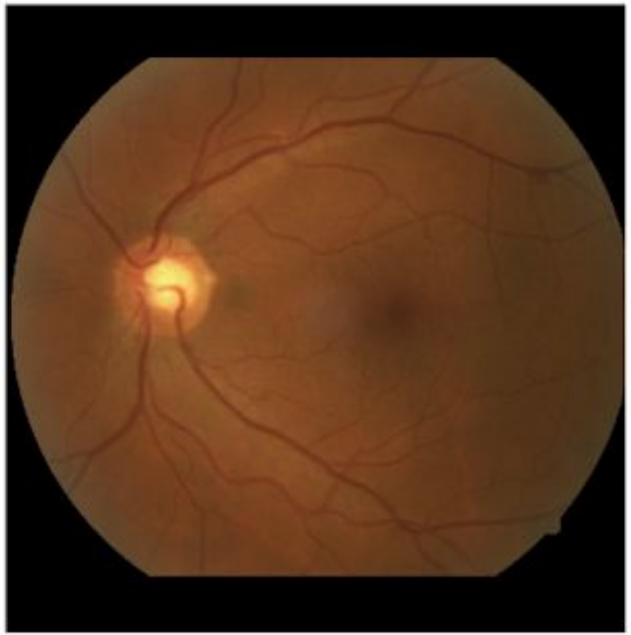


Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

# Gulshan et al. 2016

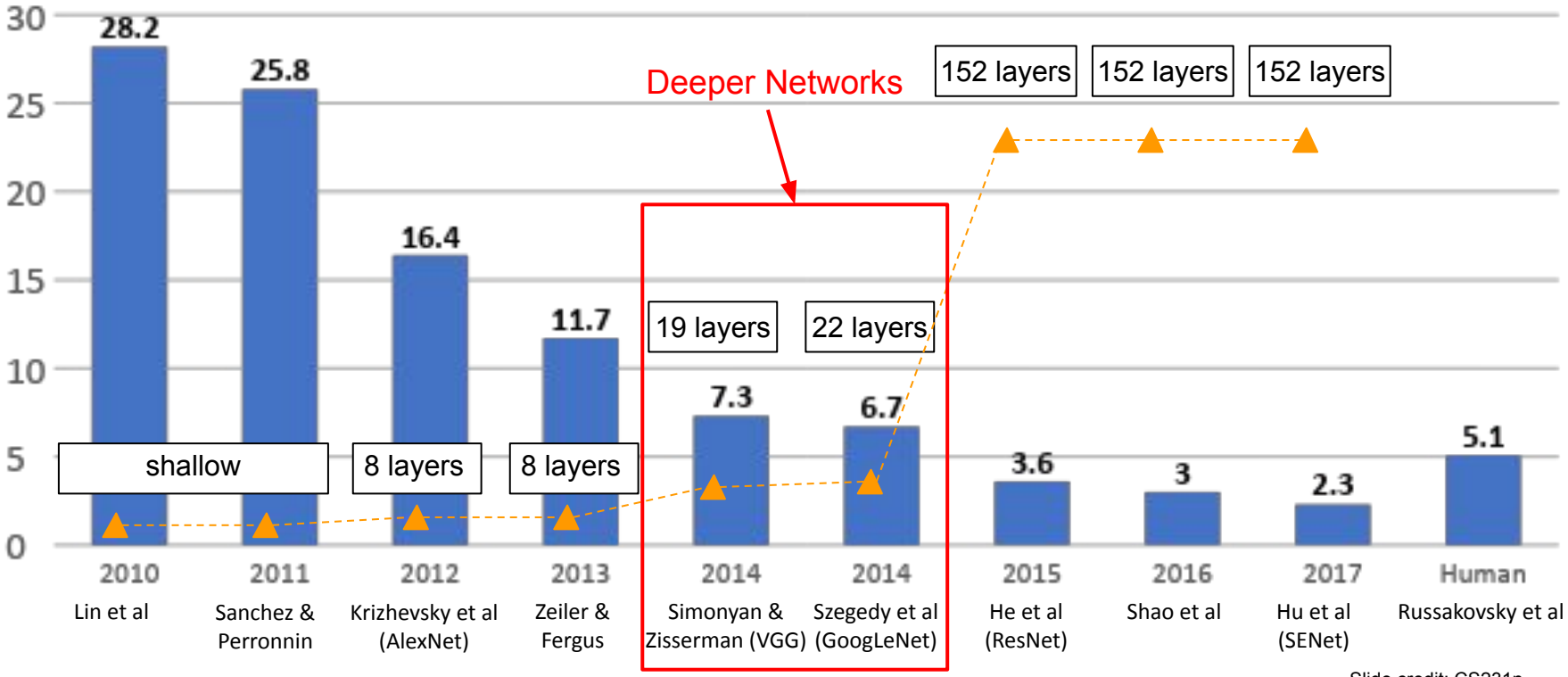
- **Dataset**:
  - 128,175 images, each graded by 3-7 ophthalmologists.
  - 54 total graders, each paid to grade between 20 to 62508 images.
- **Data preprocessing**:
  - Circular mask of each image was detected and rescaled to be 299 pixels wide
- **Model**:
  - Inception-v3 CNN, with ImageNet pre-training
  - Multiple BCE losses corresponding to different binary prediction problems, which were then used for final determination of referable diabetic retinopathy

Graders provided finer-grained labels which were then consolidated into (easier) binary prediction problems



Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners




Slide credit: CS231n

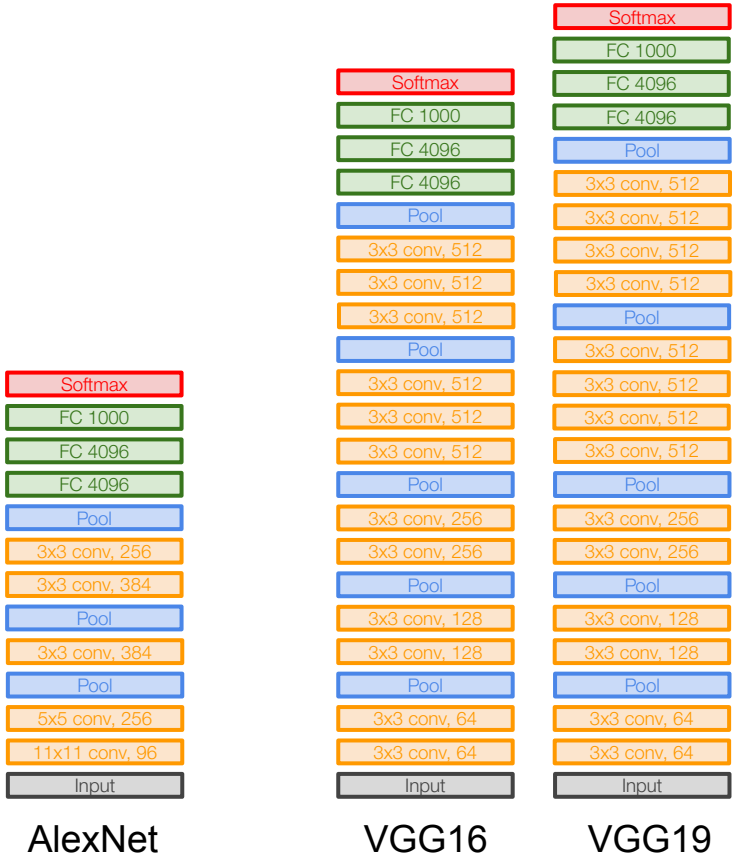
# VGGNet

*[Simonyan and Zisserman, 2014]*

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and  2x2 MAX POOL stride 2

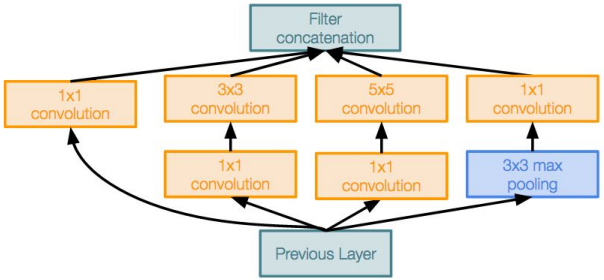



Slide credit: CS231n

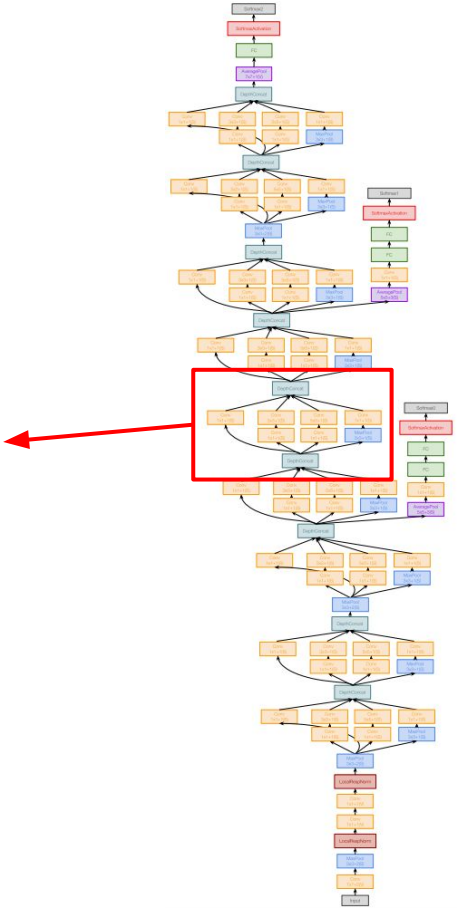
# GoogLeNet

*[Szegedy et al., 2014]*

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other




Inception module



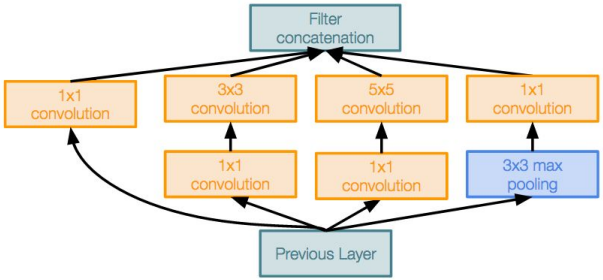
Slide credit: CS231n

# GoogLeNet

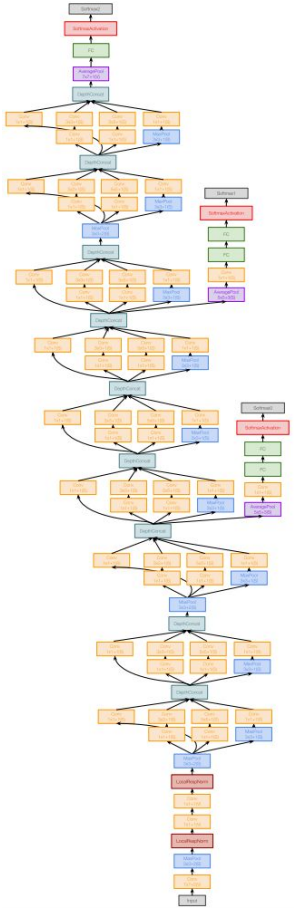
*[Szegedy et al., 2014]*

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers using a global averaging layer
- 12x less params than AlexNet




Inception module



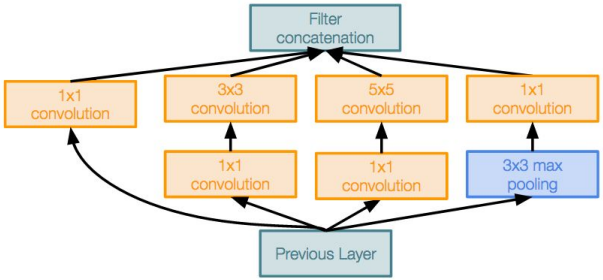
Slide credit: CS231n

# GoogLeNet

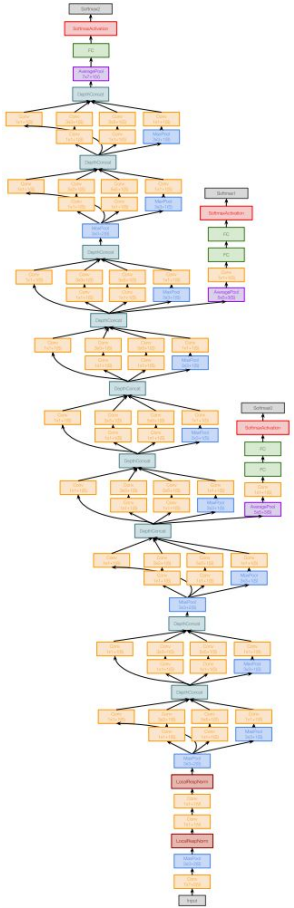
*[Szegedy et al., 2014]*

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers using a global averaging layer
- 12x less params than AlexNet




Inception module



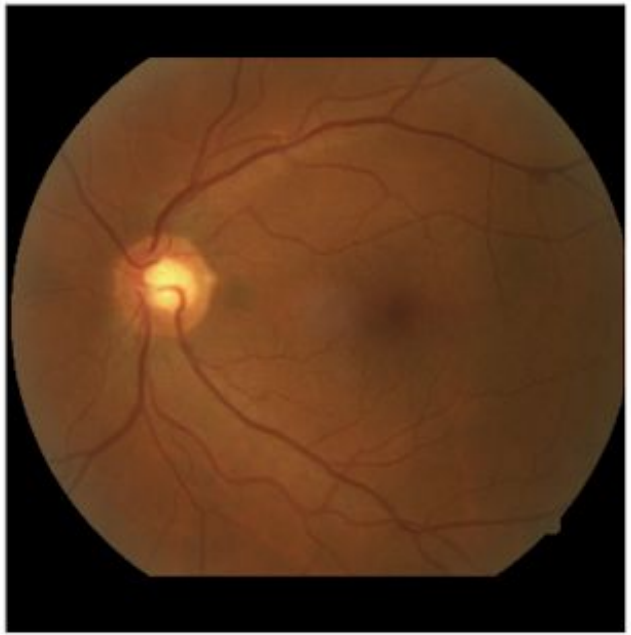
Also called “Inception Network”

Slide credit: CS231n

# Gulshan et al. 2016

- **Dataset**:
  - 128,175 images, each graded by 3-7 ophthalmologists.
  - 54 total graders, each paid to grade between 20 to 62508 images.
- **Data preprocessing**:
  - Circular mask of each image was detected and rescaled to be 299 pixels wide
- **Model**:
  - Inception-v3 CNN, with ImageNet pre-training
  - Multiple BCE losses corresponding to different binary prediction problems, which were then used for final determination of referable diabetic retinopathy

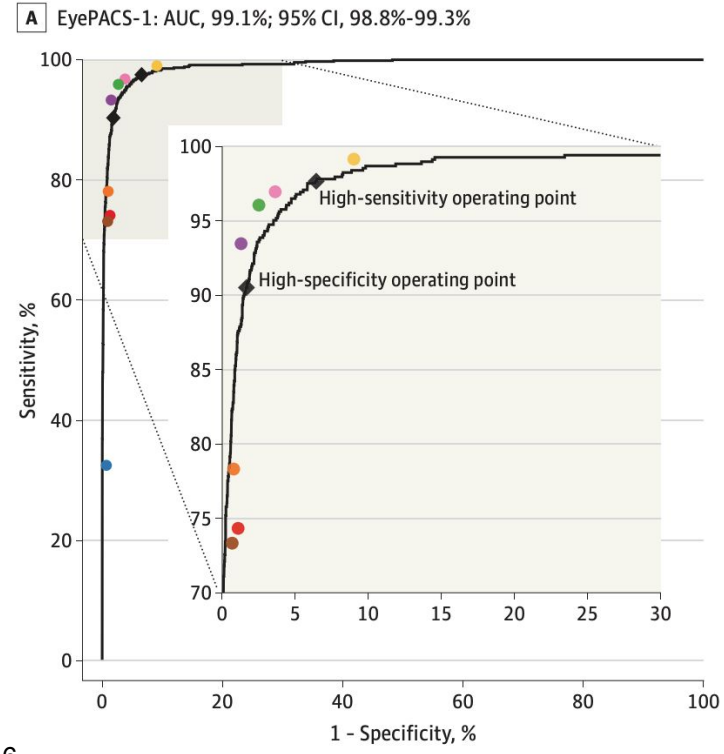
Graders provided finer-grained labels which were then consolidated into (easier) binary prediction problems



Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

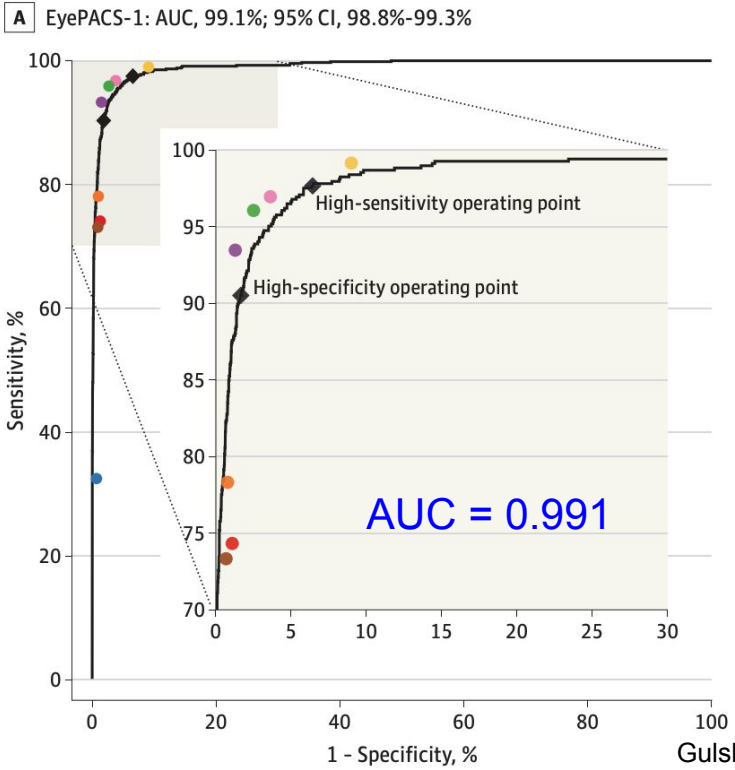
# Gulshan et al. 2016

- **Results**:
  - Evaluated using ROC curves, AUC, sensitivity and specificity analysis




Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

# Gulshan et al. 2016

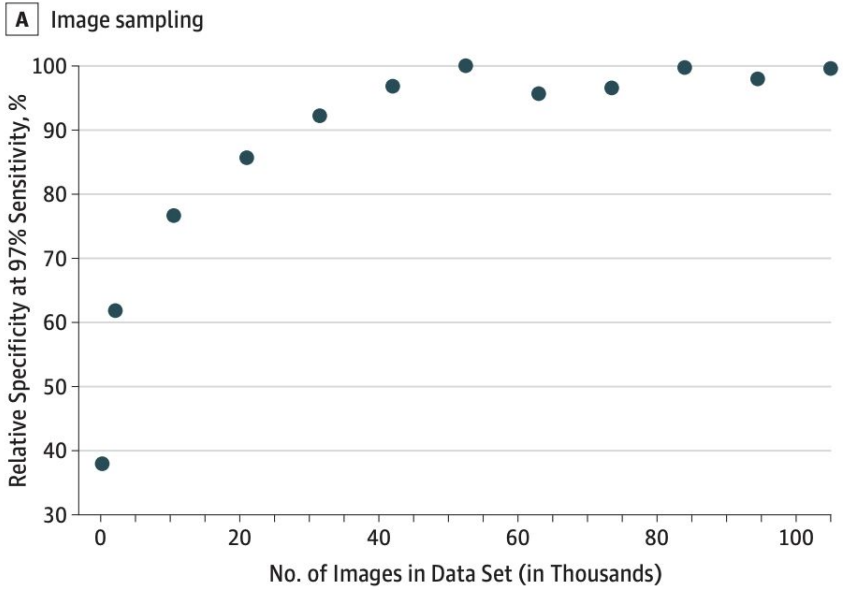


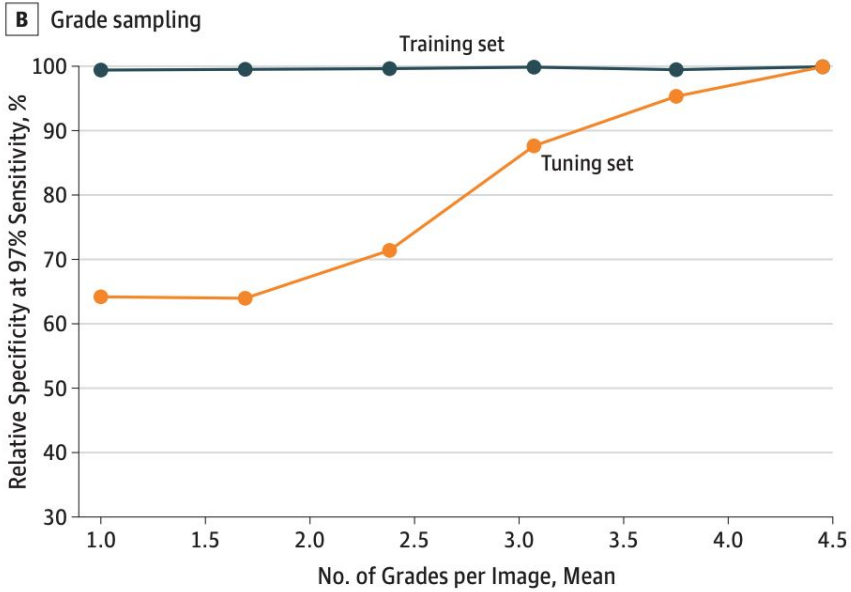

Looked at different operating points

- High-specificity point approximated ophthalmologist specificity for comparison. Should also use high-specificity to make decisions about high-risk actions.
- High-sensitivity point should be used for screening applications.

Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

# Gulshan et al. 2016

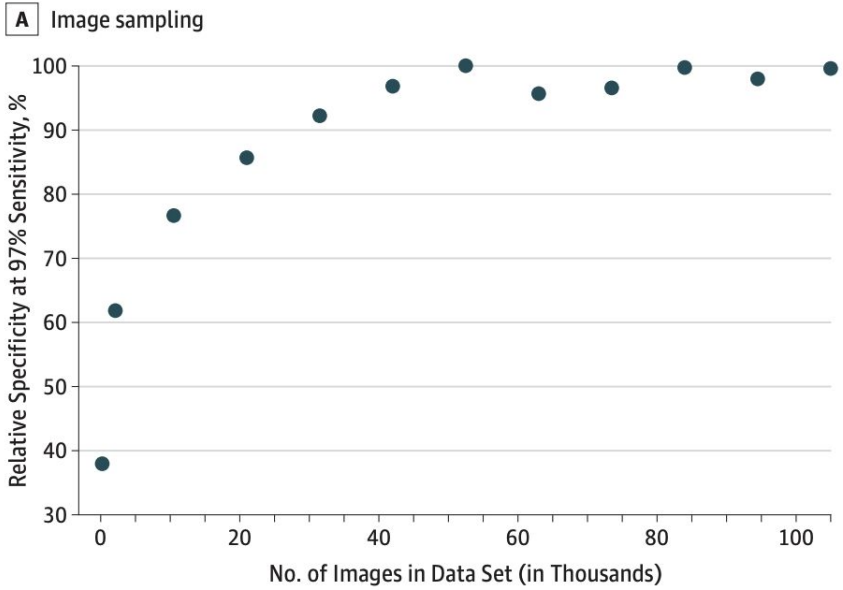


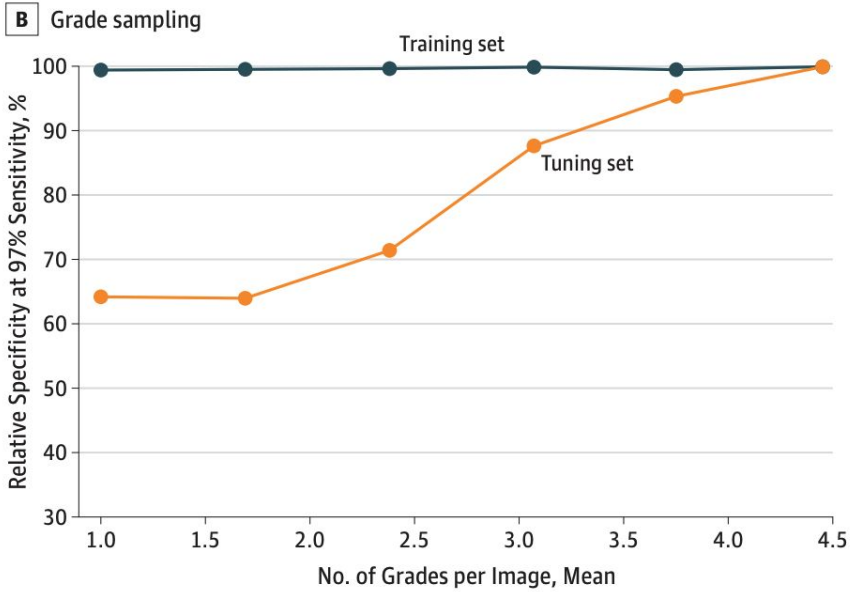




Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

# Gulshan et al. 2016

Q: What could explain the difference in trends for reducing # grades / image on training set vs. tuning set, on tuning set performance?




Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

# Considering multiple possible sources of data

E.g., some with noisier / less accurate labels than others, from different hospital sites, etc.

- Expected diversity of data during deployment should be reflected in both training and test sets
  - Need to see these during training to learn how to handle them
  - Need to see these during testing to accurately evaluate the model

# Considering multiple possible sources of data

E.g., some with noisier / less accurate labels than others, from different hospital sites, etc.

- Expected diversity of data during deployment should be reflected in both training and test sets
  - Need to see these during training to learn how to handle them
  - Need to see these during testing to accurately evaluate the model

- Want test set labels to be as accurate as possible

# Considering multiple possible sources of data

E.g., some with noisier / less accurate labels than others, from different hospital sites, etc.

- Expected diversity of data during deployment should be reflected in both training and test sets
  - Need to see these during training to learn how to handle them
  - Need to see these during testing to accurately evaluate the model

- Want test set labels to be as accurate as possible

- Noisy labels is often still useful during training -- can provide useful signal in aggregate. Much larger amount, but noisy, data is *sometimes* better than small but clean data.
  - “Weakly supervised learning” is a major area of research focused on learning with large amounts of noisy or imprecise labels

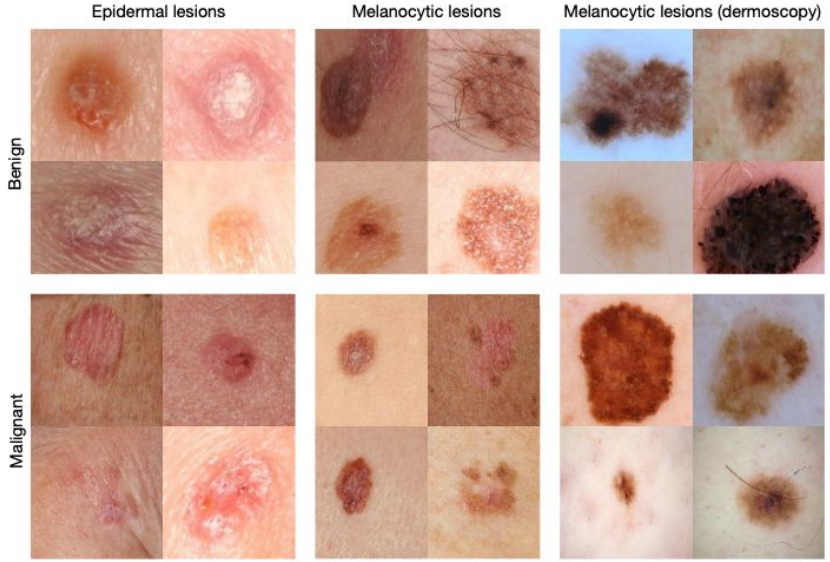
# Preview: advanced approaches for handling limited labeled data

- Semi-supervised learning
- Self-supervised learning
- Weakly supervised learning
- Domain adaptation

Will talk more about these in later lectures...

# Esteva et al. 2017

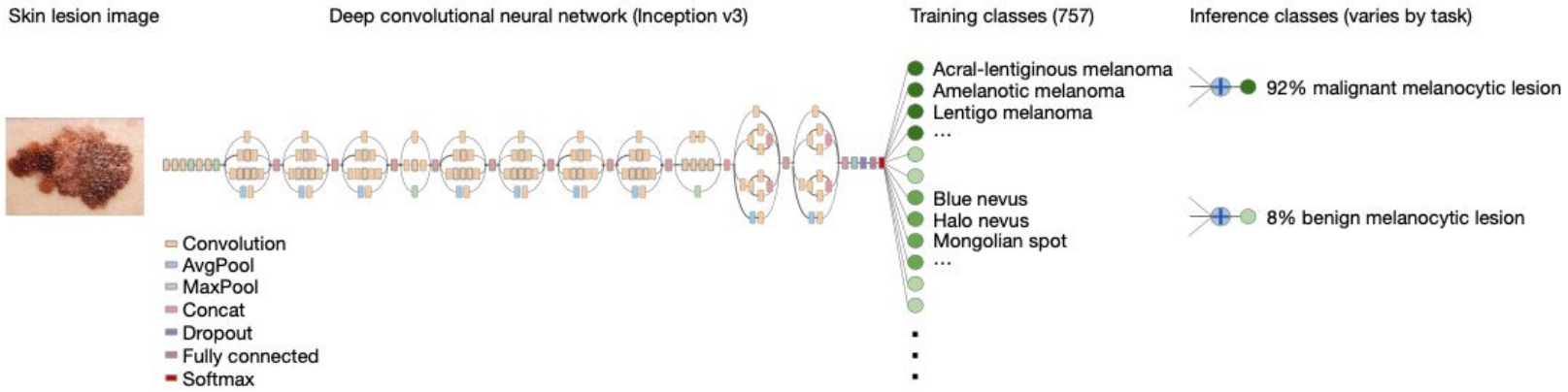
- Two binary classification tasks: malignant vs. benign lesions of epidermal or melanocytic origin
- Inception-v3 (GoogLeNet) CNN with ImageNet pre-training
- Fine-tuned on dataset of 129,450 lesions (from several sources) comprising 2,032 diseases
- Evaluated model vs. 21 or more dermatologists in various settings




Esteva*, Kuprel*, et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, 2017.

# Esteva et al. 2017

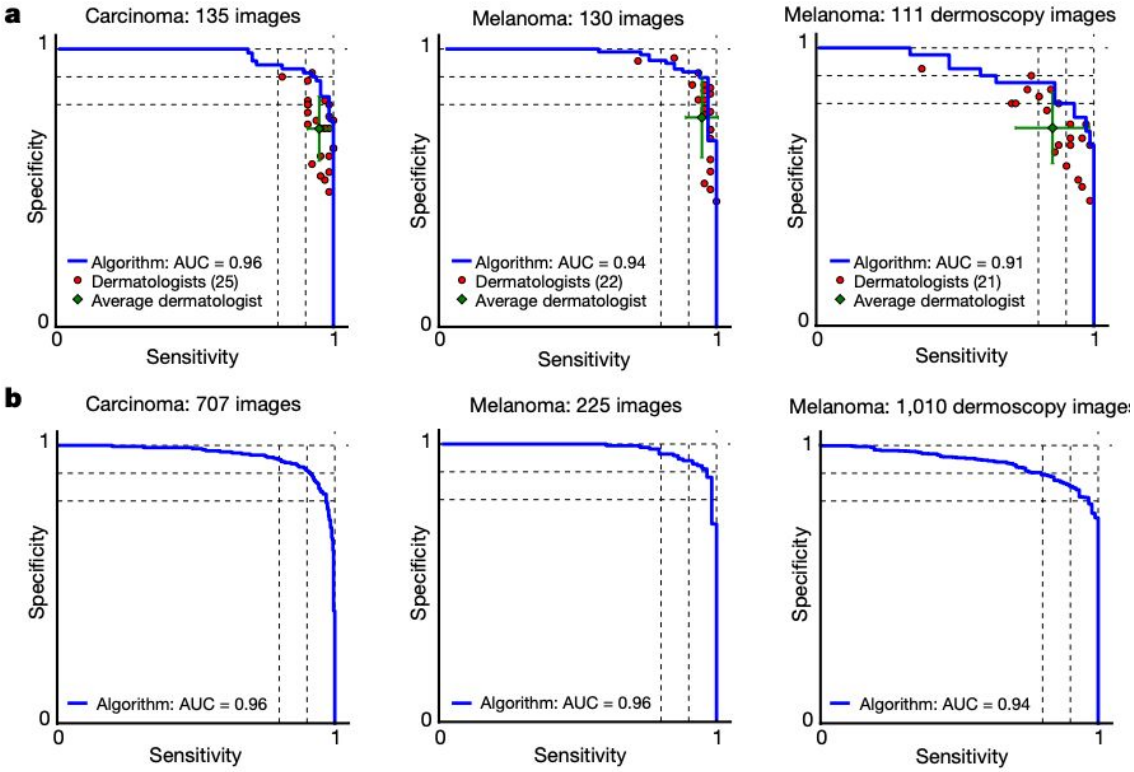
- Train on finer-grained classification (757 classes) but perform binary classification at inference time by summing probabilities of fine-grained sub-classes
- The stronger fine-grained supervision during the training stage improves inference performance!




Esteva*, Kuprel*, et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, 2017.

# Esteva et al. 2017

- Evaluation of algorithm vs. dermatologists




Esteva*, Kuprel*, et al. Dermatologist-level classification of skin cancer with deep neural networks. Nature, 2017.

# Lakhani and Sundaram 2017

- Binary classification of pulmonary tuberculosis from x-rays
- Four de-identified datasets
- 1007 chest x-rays (68% train, 17.1% validation, 14.9% test)
- Tried training CNNs from scratch as well as fine-tuning from ImageNet

**AUC Test Dataset**

| Parameter | Untrained | Pretrained | Untrained with Augmentation* | Pretrained with Augmentation* |
|---|---|---|---|---|
| AlexNet | 0.90 (0.84, 0.95) | 0.98 (0.95, 1.00) | 0.95 (0.90, 0.98) | 0.98 (0.94, 0.99) |
| GoogLeNet | 0.88 (0.81, 0.92) | 0.97 (0.93, 0.99) | 0.94 (0.89, 0.97) | 0.98 (0.94, 1.00) |
| Ensemble | | | | 0.99 (0.96, 1.00) |

Note.—Data in parentheses are 95% confidence interval.

* Additional augmentation of 90, 180, 270 rotations, and Contrast Limited Adaptive Histogram Equalization processing.

Lakhani and Sundaram. Deep learning at chest radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks. Radiology, 2017.

# Lakhani and Sundaram 2017

- Binary classification of pulmonary tuberculosis from x-rays
- Four de-identified datasets
- 1007 chest x-rays (68% train, 17.1% validation, 14.9% test)
- Tried training CNNs from scratch as well as fine-tuning from ImageNet

**AUC Test Dataset**

| Parameter | Untrained | Pretrained | Untrained with Augmentation* | Pretrained with Augmentation* |
|---|---|---|---|---|
| AlexNet | 0.90 (0.84, 0.95) | 0.98 (0.95, 1.00) | 0.95 (0.90, 0.98) | 0.98 (0.94, 0.99) |
| GoogLeNet | 0.88 (0.81, 0.92) | 0.97 (0.93, 0.99) | 0.94 (0.89, 0.97) | 0.98 (0.94, 1.00) |
| Ensemble | | | | 0.99 (0.96, 1.00) |

Note.—Data in parentheses are 95% confidence interval.

* Additional augmentation of 90, 180, 270 rotations, and Contrast Limited Adaptive Histogram Equalization processing.

All training images were resized to 256x256 and underwent base data augmentation of random 227x227 cropping and mirror images. Additional data augmentation experiments in results table.

Lakhani and Sundaram. Deep learning at chest radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks. Radiology, 2017.

# Lakhani and Sundaram 2017

- Binary classification of pulmonary tuberculosis from x-rays
- Four de-identified datasets
- 1007 chest x-rays (68% train, 17.1% validation, 14.9% test)
- Tried training CNNs from scratch as well as fine-tuning from ImageNet

**AUC Test Dataset**

| Parameter | Untrained | Pretrained | Untrained with Augmentation* | Pretrained with Augmentation* |
|---|---|---|---|---|
| AlexNet | 0.90 (0.84, 0.95) | 0.98 (0.95, 1.00) | 0.95 (0.90, 0.98) | 0.98 (0.94, 0.99) |
| GoogLeNet | 0.88 (0.81, 0.92) | 0.97 (0.93, 0.99) | 0.94 (0.89, 0.97) | 0.98 (0.94, 1.00) |
| Ensemble | | | | 0.99 (0.96, 1.00) |

Note.—Data in parentheses are 95% confidence interval.

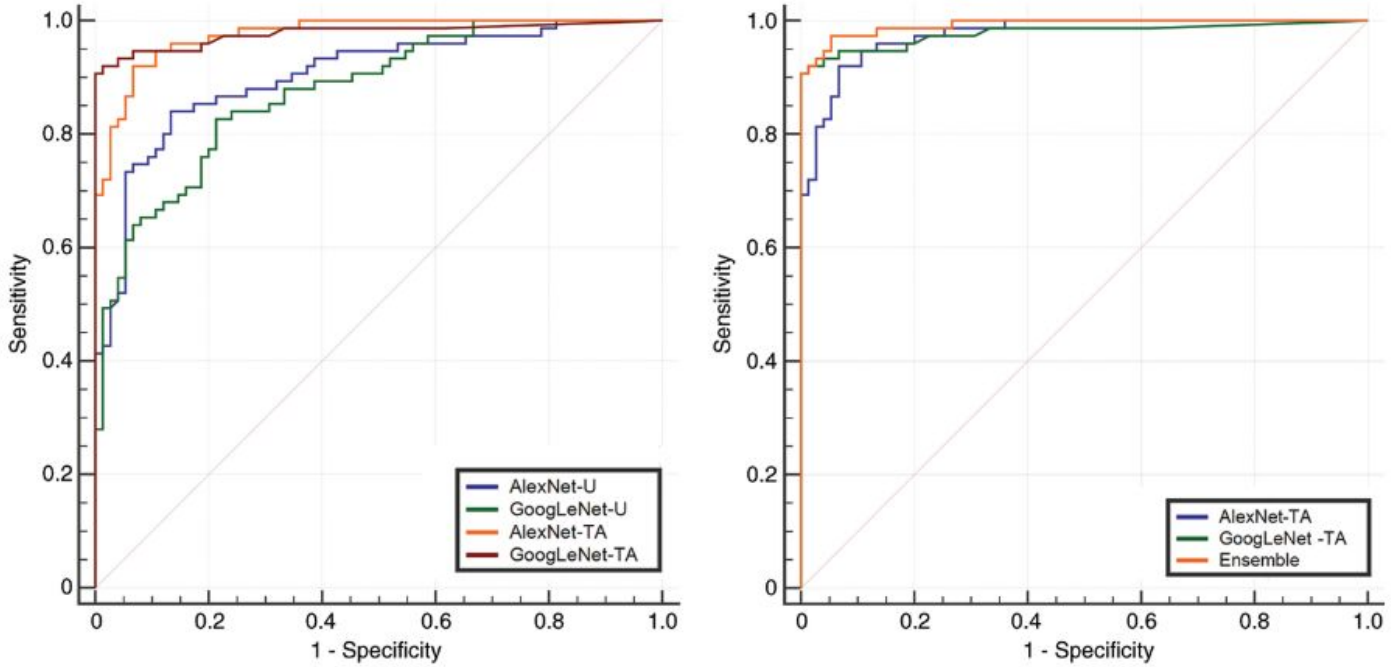
* Additional augmentation of 90, 180, 270 rotations, and Contrast Limited Adaptive Histogram Equalization processing.

All training images were resized to 256x256 and underwent base data augmentation of random 227x227 cropping and mirror images. Additional data augmentation experiments in results table.

Often resize to match input size of pre-trained networks. Also fine approach to making high-res dataset easier to work with!

Lakhani and Sundaram. Deep learning at chest radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks. Radiology, 2017.

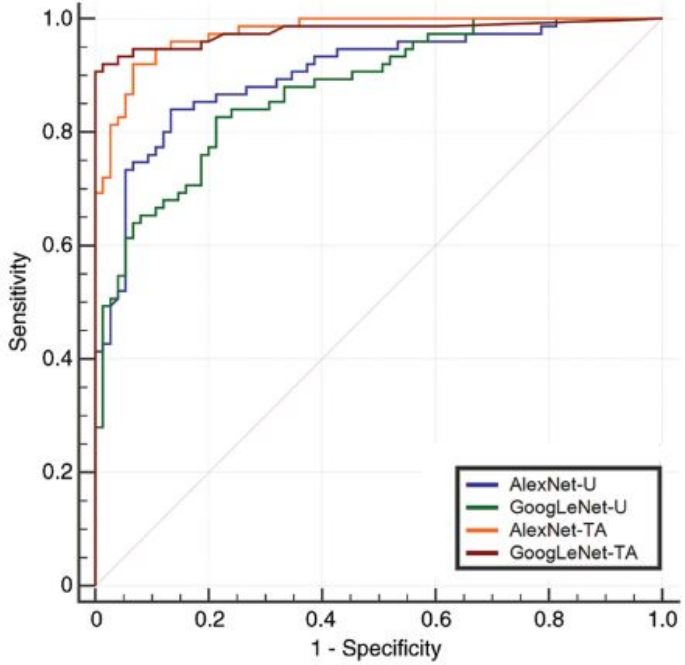
# Lakhani and Sundaram 2017

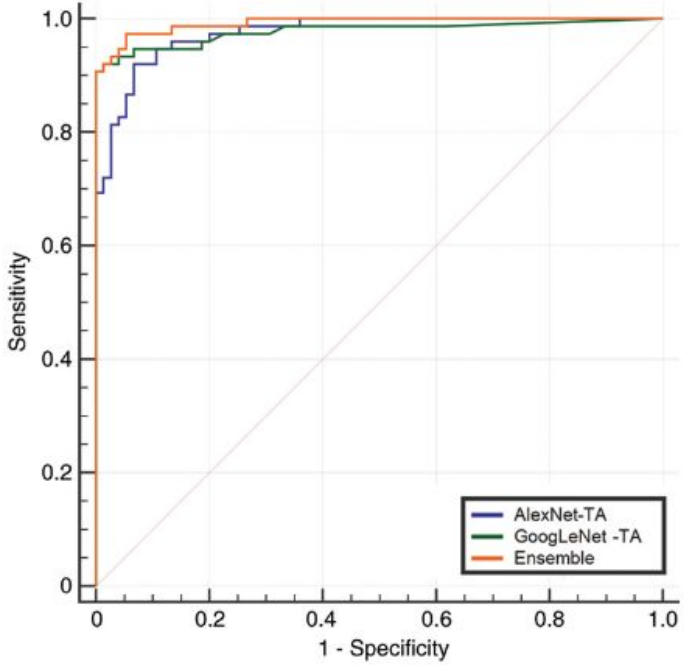



Lakhani and Sundaram. Deep learning at chest radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks. Radiology, 2017.

# Lakhani and Sundaram 2017

Performed further analysis at optimal threshold determined by the Youden Index.

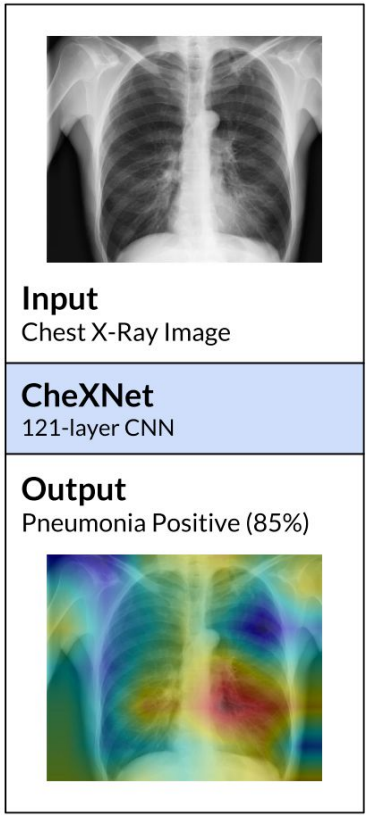



Lakhani and Sundaram. Deep learning at chest radiography: Automated Classification of Pulmonary Tuberculosis by Using Convolutional Neural Networks. Radiology, 2017.

# Rajpurkar et al. 2017

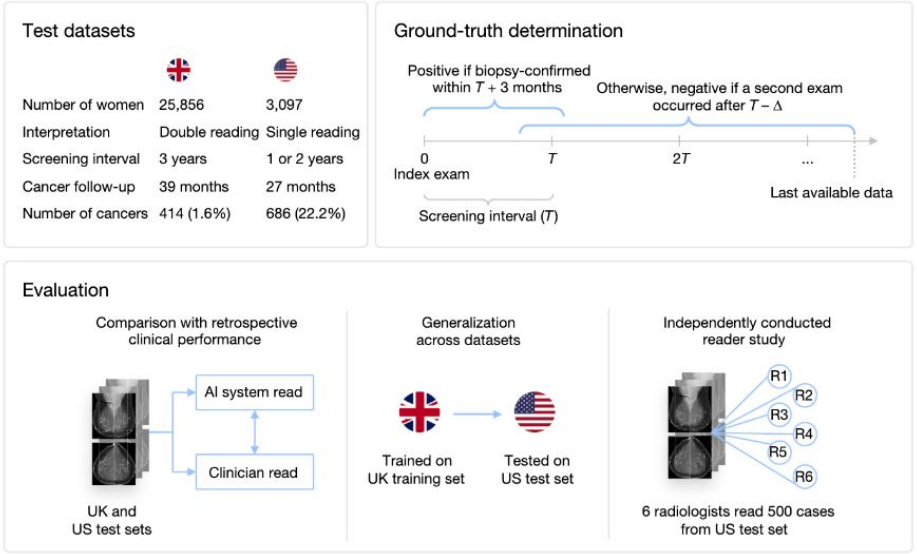
- Binary classification of pneumonia presence in chest X-rays
- Used ChestX-ray14 dataset with over 100,000 frontal X-ray images with 14 diseases
- 121-layer DenseNet CNN
- Compared algorithm performance with 4 radiologists
- Also applied algorithm to other diseases to surpass previous state-of-the-art on ChestX-ray14




Rajpurkar et al. CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning. 2017.

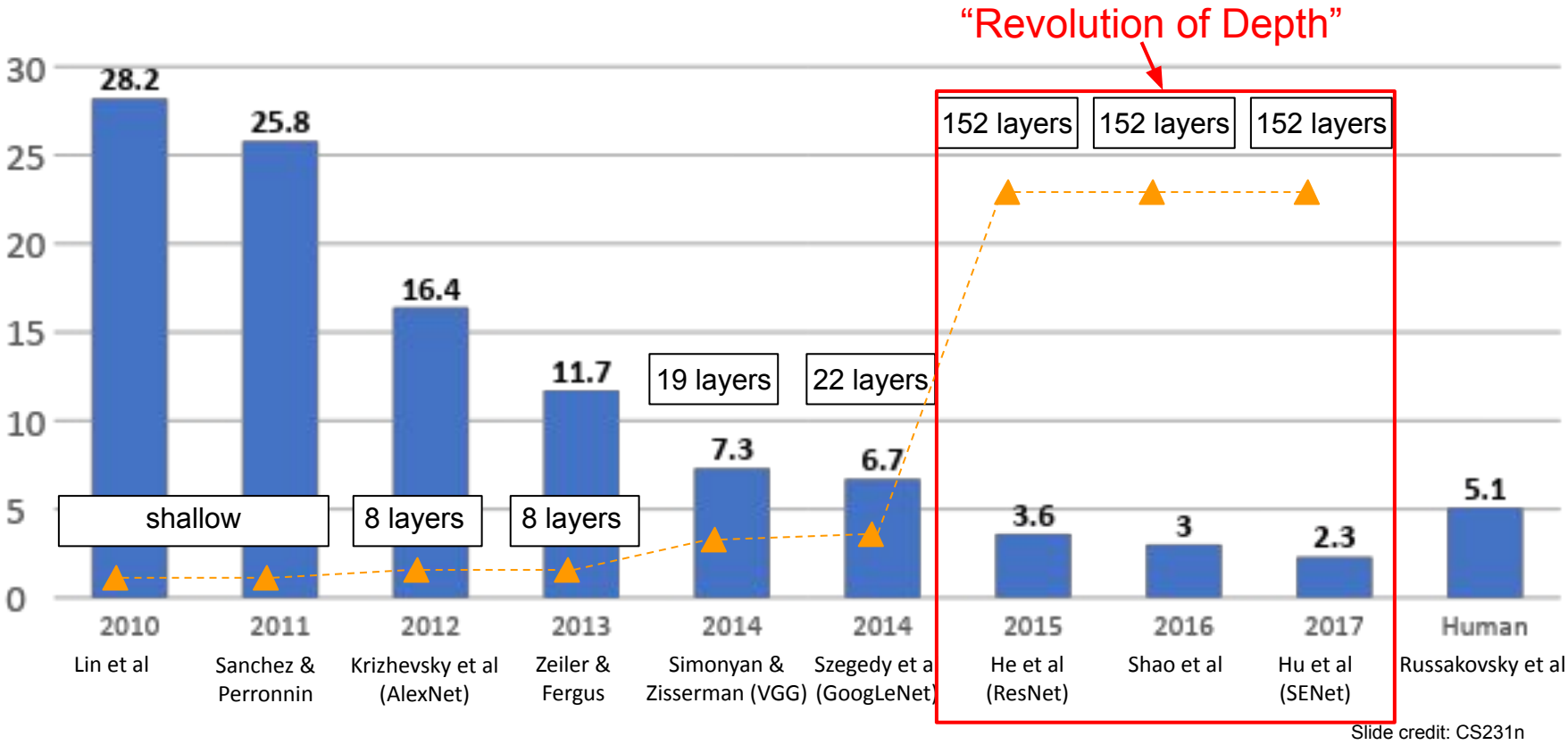
# McKinney et al. 2020

- Binary classification of breast cancer in mammograms
- Used an ensemble of models including ResNets
- International dataset and evaluation, across UK and US




McKinney et al. International evaluation of an AI system for breast cancer screening. Nature, 2020.

# ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



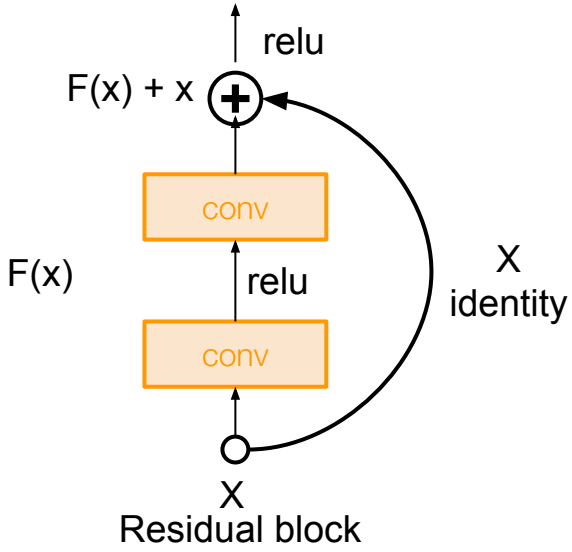

Slide credit: CS231n

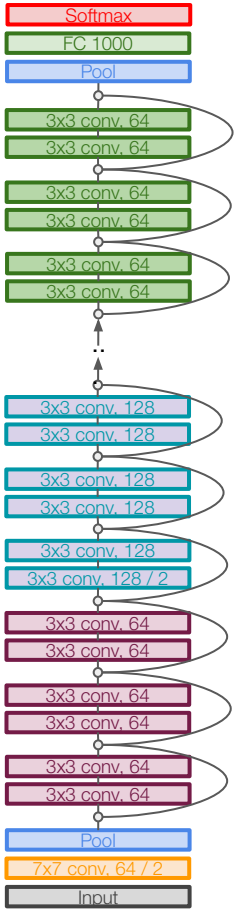
# ResNet

*[He et al., 2015]*

Very deep networks using residual connections

- 152-layer model for ImageNet
- Won all major classification and detection benchmark challenges in 2015



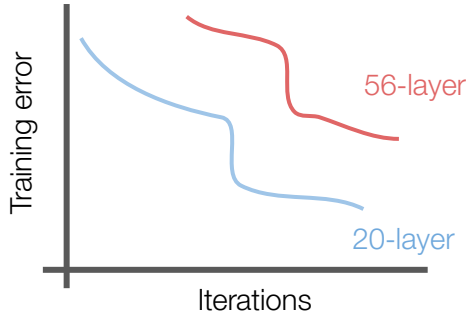



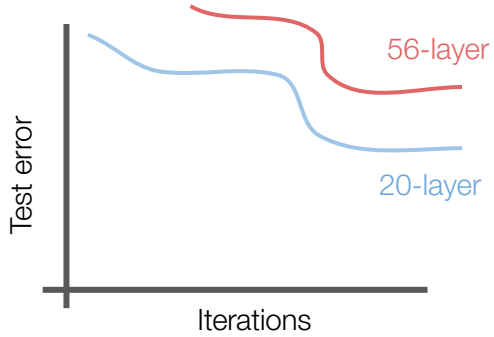

Slide credit: CS231n

# ResNet

*[He et al., 2015]*

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



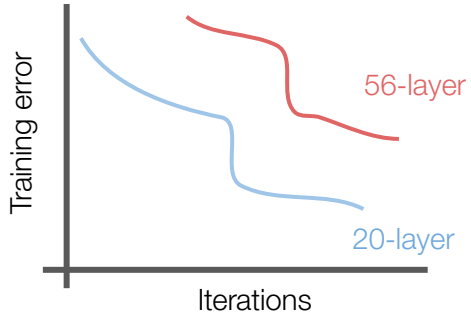

Q: What’s strange about these training and test curves?
[Hint: look at the order of the curves]

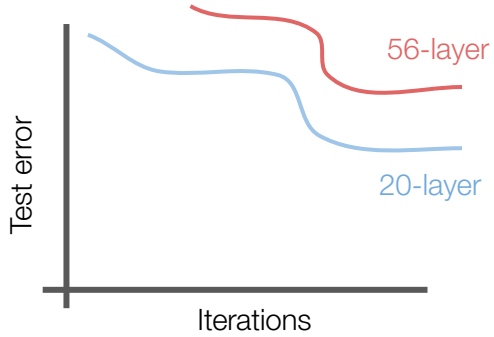
Slide credit: CS231n

# ResNet

*[He et al., 2015]*

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?




56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Slide credit: CS231n

# ResNet

*[He et al., 2015]*

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

Slide credit: CS231n

# ResNet

*[He et al., 2015]*

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

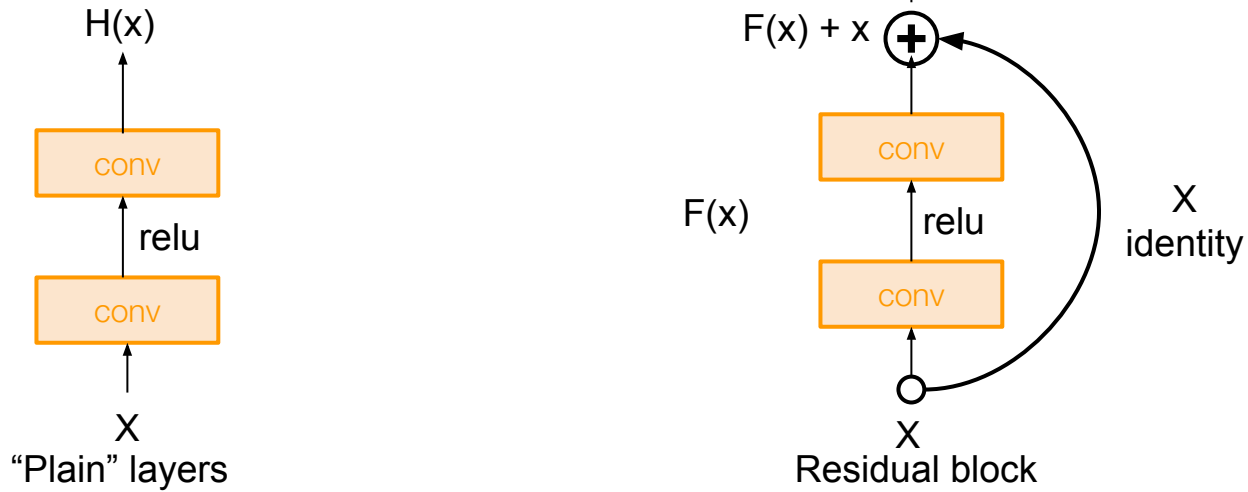
A solution by construction is copying the learned layers over from the shallower model and setting all additional layers to the **identity** function.

Slide credit: CS231n

# ResNet

*[He et al., 2015]*

Solution: Structure each network layer to fit a “residual function” with respect to the identity function, then add the two functions together



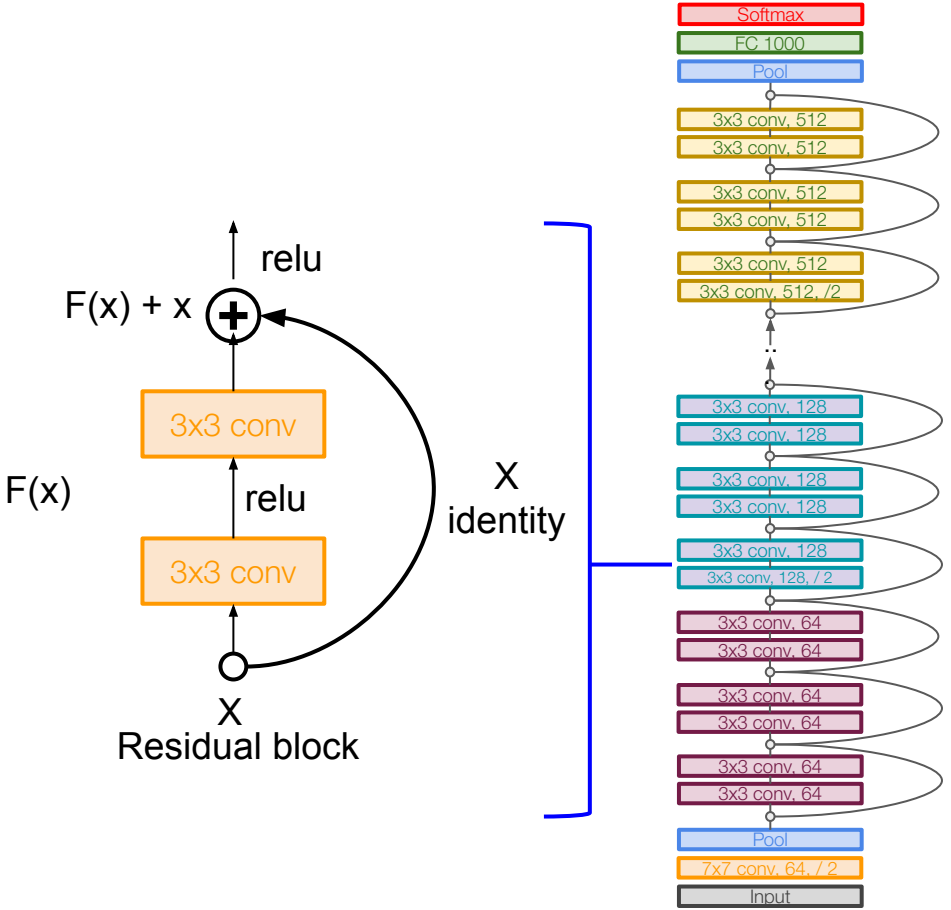

Slide credit: CS231n

# ResNet

*[He et al., 2015]*

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers



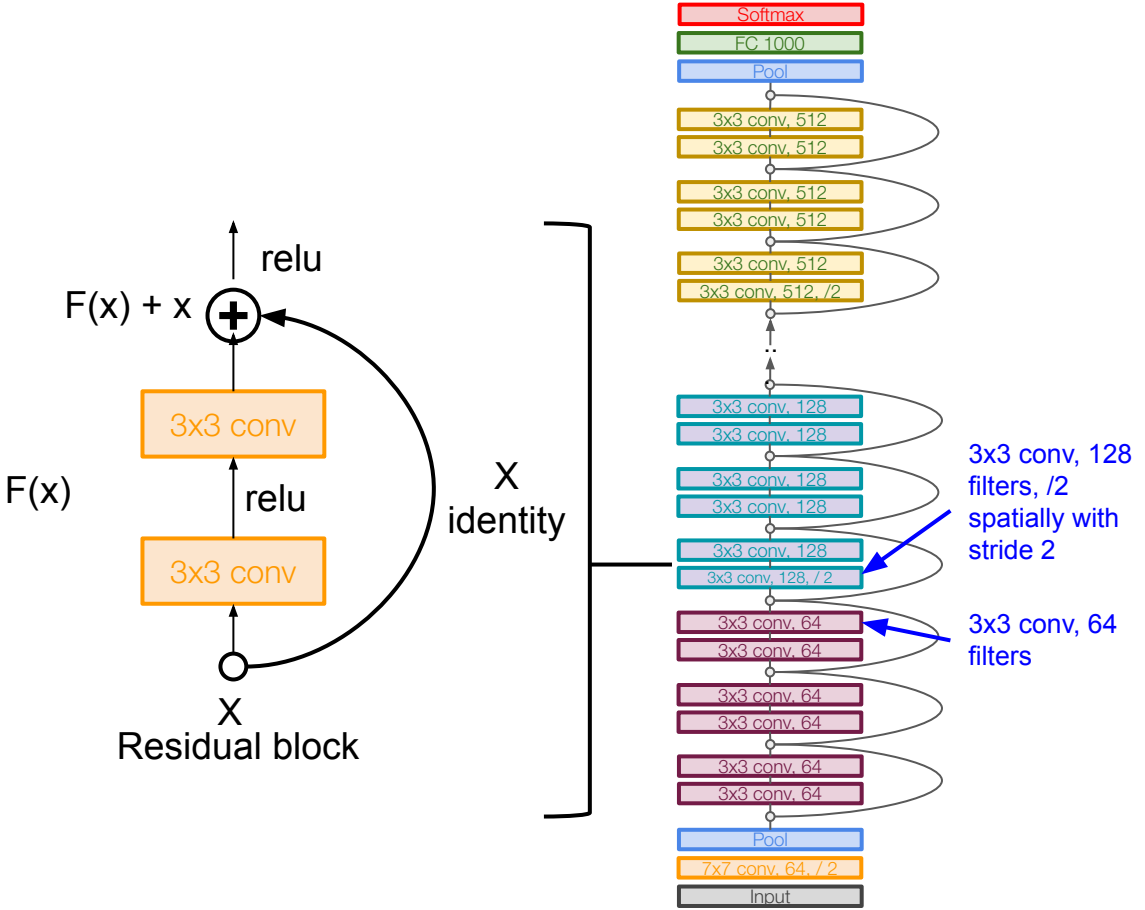

Slide credit: CS231n

# ResNet

*[He et al., 2015]*

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)



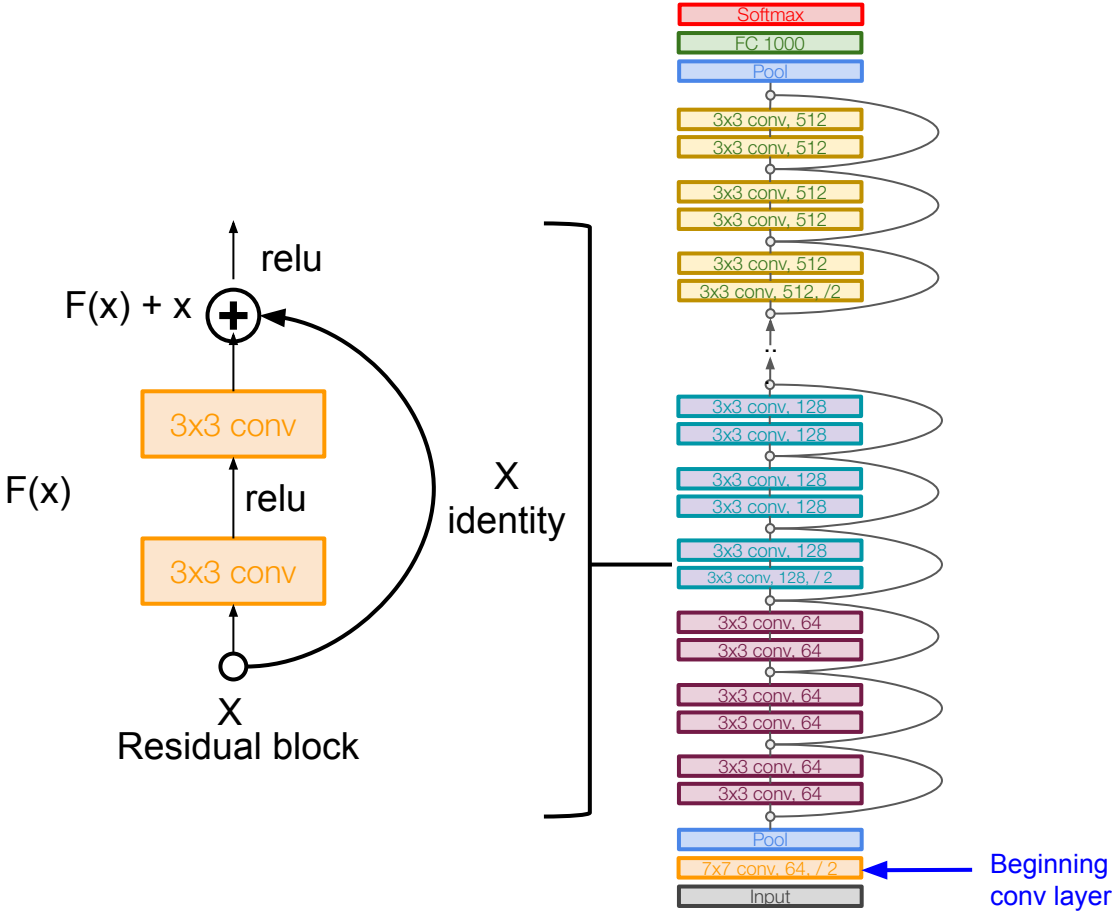

Slide credit: CS231n

# ResNet

*[He et al., 2015]*

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



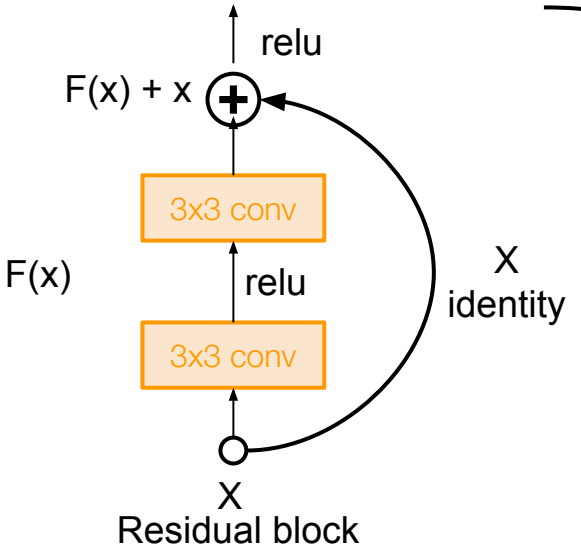

Slide credit: CS231n

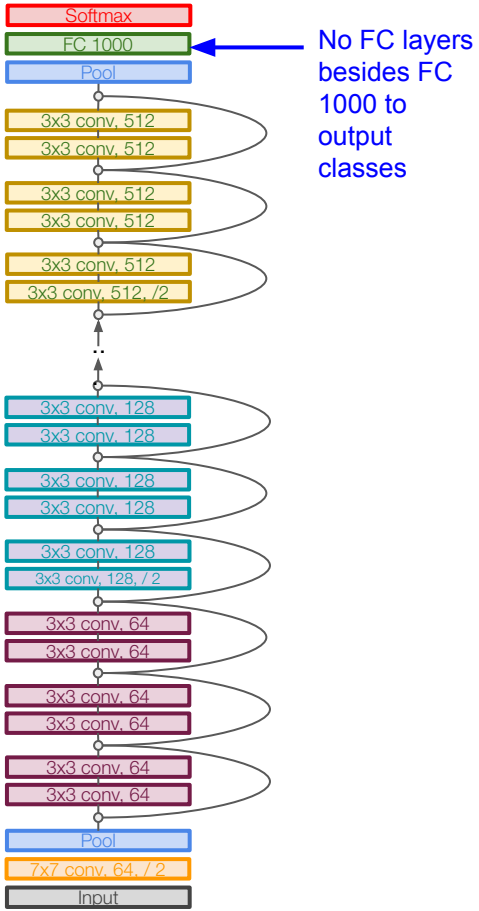
# ResNet

*[He et al., 2015]*

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)



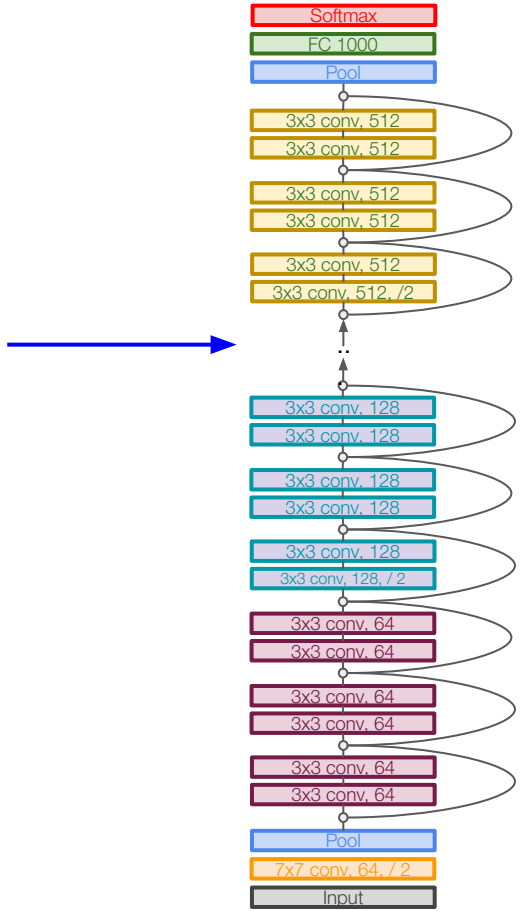




Slide credit: CS231n

# ResNet

*[He et al., 2015]*

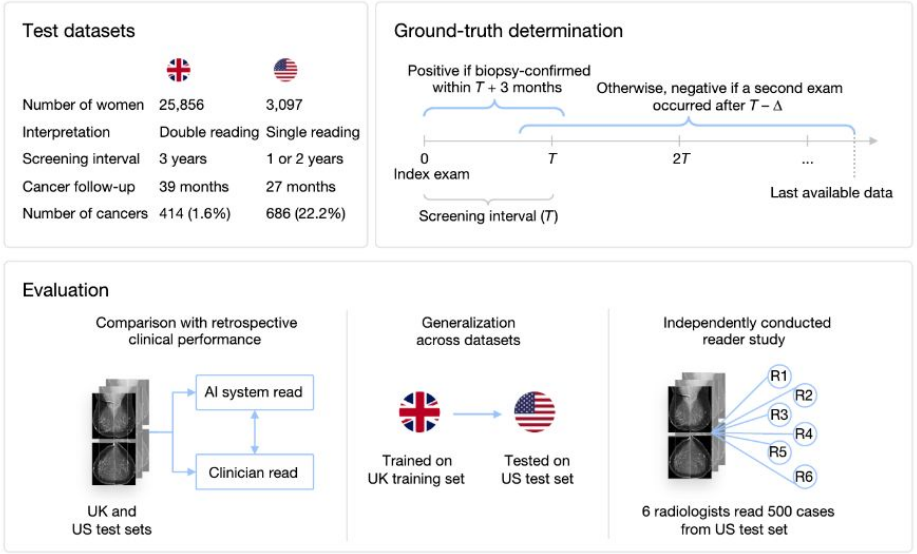
Total depths of 34, 50, 101, or 152 layers for ImageNet




Slide credit: CS231n

# McKinney et al. 2020

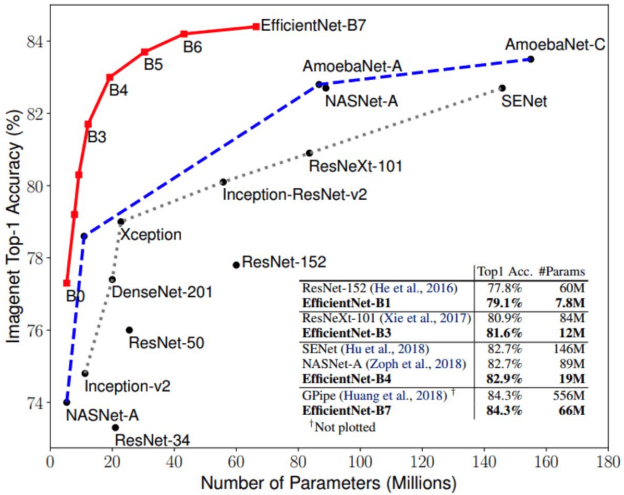
- Binary classification of breast cancer in mammograms
- Used an ensemble of models including ResNets
- International dataset and evaluation, across UK and US




McKinney et al. International evaluation of an AI system for breast cancer screening. Nature, 2020.

# More recent CNN architectures

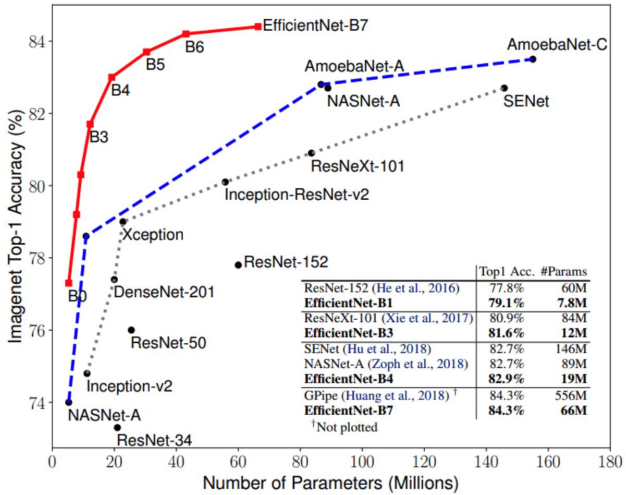
- MobileNet (Sandler et al. 2018) - architecture with separable convolutions for light-weight CNNs
- NASNet (Zoph et al. 2016) and AmoebaNet (Real et al. 2019) - architectures discovered through “neural architecture search” via reinforcement learning or evolutionary algorithms
- EfficientNet (Tan et al. 2020) - family of architectures designed using “compound scaling” that simultaneously scale width, depth, and resolution of neural networks with a fixed ratio



# More recent CNN architectures

Worth exploring for class projects!

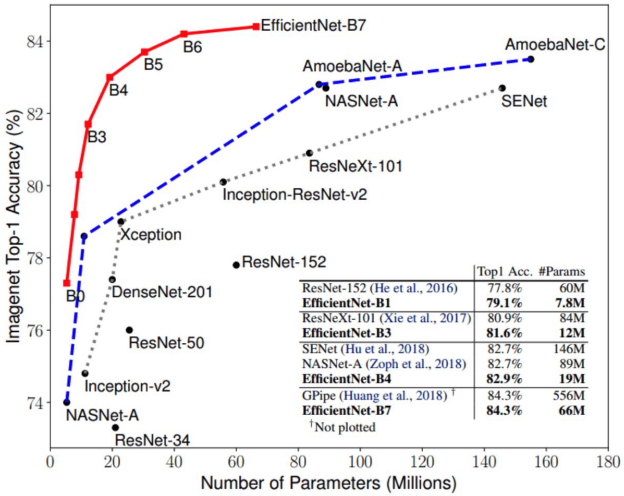
- MobileNet (Sandler et al. 2018) - architecture with separable convolutions for light-weight CNNs
- NASNet (Zoph et al. 2016) and AmoebaNet (Real et al. 2019) - architectures discovered through “neural architecture search” via reinforcement learning or evolutionary algorithms
- EfficientNet (Tan et al. 2020) - family of architectures designed using “compound scaling” that simultaneously scale width, depth, and resolution of neural networks with a fixed ratio



# More recent CNN architectures

Worth exploring for class projects!

- MobileNet (Sandler et al. 2018) - architecture with separable convolutions for light-weight CNNs
- NASNet (Zoph et al. 2016) and AmoebaNet (Real et al. 2019) - architectures discovered through "neural architecture search" via reinforcement learning or evolutionary algorithms
- EfficientNet (Tan et al. 2020) - family of architectures designed using "compound scaling" that simultaneously scale width, depth, and resolution of neural networks with a fixed ratio




Preview: Transformers, a new class of deep learning architecture, was originally designed for NLP/sequence data but has recently also been applied for computer vision tasks. Stay tuned!

# Summary

Today we saw:

- Convolutional neural networks for image classification
- Data considerations for image classification models
- Evaluating image classification models
- Case studies of CNNs for medical image classification
  - More on transfer learning and how much data needs for deep learning
  - More on recent CNN architectures

Next time: Medical Images: Advanced Vision Models (Detection and Segmentation)