Lecture 3:
Medical Images: Classification

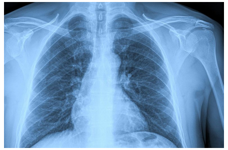
Administrative

- A0 due Tue 9/22 11:59pm
- A1 will also be released Tue, due in 2 weeks (Tue 10/6)
  - 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
- Tensorflow Review Session this Fri 1pm, helpful for A1
- Start thinking about your class project! We will release a piazza post soon with some ideas to help guide your thinking.
  - Project guidelines already posted on class website, will go over more on Wed
  - First due date: Project proposal, due Fri 10/9

Last time: Deep learning framework

Traditional machine learning

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

(e.g., )

(e.g., color and texture histograms)

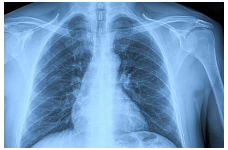
(e.g., support vector machines and random forests)

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

Deep learning

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

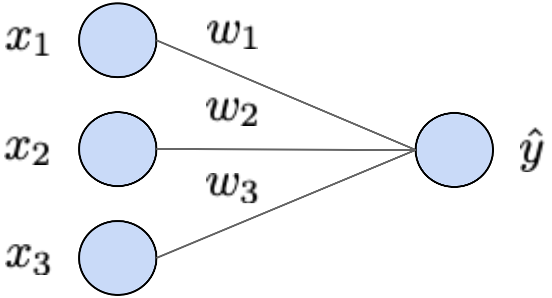
Input → **Deep Learning Model** → **Output**

(e.g., )

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

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

A simple example



Output: $\hat{y} = w_1x_1 + w_2x_2 + w_3x_3 + b$

$= w^Tx + b$

Neural network parameters:

$$W = \{[w_1, w_2, w_3], b\}$$

Loss function:

Per-example: $L^i(W) = (\hat{y}^i - y^i)^2$

Over M examples: $L = \frac{1}{M}\sum_i L^i(W)$

Gradient of loss w.r.t. weights:

Partial derivative of loss w.r.t. kth weight:

$$\frac{\partial L^i}{\partial w_k} = \frac{\partial L^i}{\partial \hat{y}^i}\frac{\partial \hat{y}^i}{\partial w_k} = 2(\hat{y}^i - y^i)x_k^i$$

$$\frac{\partial L}{\partial w_k} = \frac{1}{M}\sum_i \frac{\partial L^i}{\partial w_k} = \frac{1}{M}\sum_i 2(\hat{y}^i - y^i)x_k^i$$

Full gradient expression:

$$\nabla L_W = \left[\frac{\partial L}{w_0}, \ldots, \frac{\partial L}{w_3}\right] = \boxed{\frac{1}{M}\sum_i 2(\hat{y}^i - y^i)x^i}$$

Gradient descent algorithm

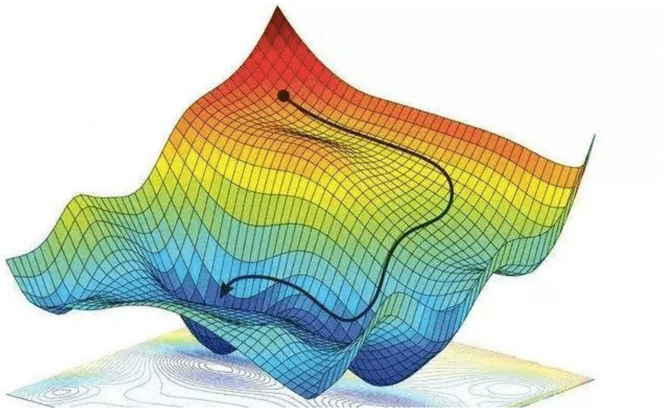
Let the gradient of the loss function with respect to the model parameters w be:

$$\nabla L_w = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, ..., \frac{\partial L}{\partial w_K}\right]$$

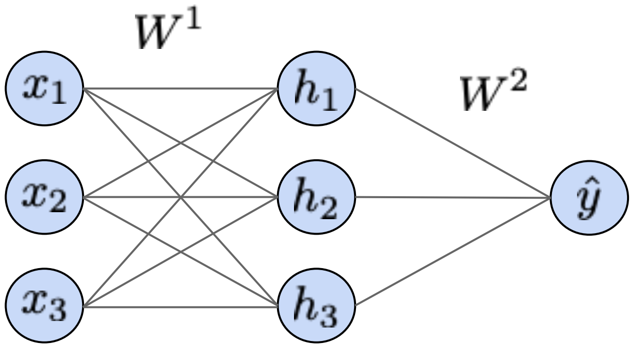
Then we can minimize the loss function by iteratively updating the model parameters (“taking steps”) in the direction of the negative gradient, until convergence:

$$w := w - \alpha \nabla L_w$$

“step size” hyperparameter (design choice) indicating how big of a step in the negative gradient direction we want to take at each update. Too big -> may overshoot minima; too small -> optimization takes too long



Now: a two-layer fully-connected neural network



$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \end{bmatrix} \quad b^1 = \begin{bmatrix} b^1_1 \\ b^1_2 \\ b^1_3 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \end{bmatrix} \quad b^2 = \begin{bmatrix} b^2_1 \end{bmatrix}$$

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

Neural network parameters:

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

Loss function (regression loss, same as before)**:**

Per-example: $L^i(W) = (\hat{y}^i - y^i)^2$

Over M examples: $L = \frac{1}{M} \sum_i L^i(W)$

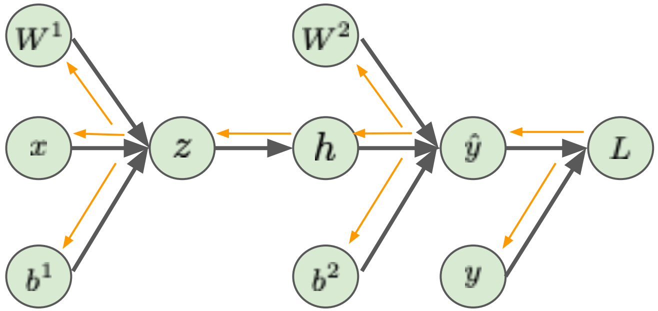
Gradient of loss w.r.t. weights:

Function more complex -> now much harder to derive the expressions! Instead… computational graphs and backpropagation.

Computing gradients with backpropagation

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

Think of computing loss function as staged computation of intermediate variables:



"Forward pass": $z = W^1 x + b^1$

$$h = \sigma(z)$$

$$\hat{y} = W^2 h + b^2$$

$$L = (\hat{y} - y)^2$$

Now, can use a repeated application of the chain rule, going backwards through the computational graph, to obtain the gradient of the loss with respect to each node of the computation graph.

"Backward pass": $\frac{\partial L}{\partial \hat{y}} = 2(\hat{y} - y)$ (not all gradients shown)

$$\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W^2}$$

$$\frac{\partial L}{\partial H} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial H}$$

$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial H} \frac{\partial H}{\partial Z}$$

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial Z} \frac{\partial z}{\partial W^1}$$

Plug in from earlier computations via chain rule

Local gradients to derive

Training our two-layer neural network in code, in Tensorflow 2.0

```python
# Our (X,Y) training set converted to TF tensors
X_tf = tf.convert_to_tensor(X, np.float32)
Y_tf = tf.convert_to_tensor(Y, np.float32)
# Create a TF dataset with specified minibatch size
batch_size = 50
dataset = tf.data.Dataset.from_tensor_slices((X_tf, Y_tf))
dataset = dataset.batch(batch_size)

# initialize model parameters to be learned
W1 = tf.Variable(tf.random.uniform((input_dim, hid_dim)))
W2 = tf.Variable(tf.random.uniform((hid_dim, output_dim)))
b1 = tf.Variable(tf.random.uniform((1, hid_dim)))
b2 = tf.Variable(tf.random.uniform((1, output_dim)))

# perform gradient descent
epochs = 5000
optimizer = tf.optimizers.SGD(learning_rate=1e-2)
losses = []

for epoch in range(epochs):
    for batch in dataset:
        X_batch, Y_batch = batch
        with tf.GradientTape() as tape:

            # forward pass
            Z_batch = tf.add(tf.matmul(X_batch, W1), b1)
            H_batch = tf.math.sigmoid(Z_batch)
            Out_batch = tf.add(tf.matmul(H_batch, W2), b2)
            loss = tf.losses.MSE(Y_batch, Out_batch)

            # backward pass and gradient update
            gradients = tape.gradient(loss, [W1, W2, b1, b2])
            optimizer.apply_gradients(zip(gradients, [W1, W2, b1, b2]))
```

Evaluate gradients using automatic differentiation and perform gradient update

Also high level libraries built on top of Tensorflow, that provide even easier-to-use APIs:

In Tensorflow 2.0:

```python
for epoch in range(epochs):
    for batch in dataset:
        X_batch, Y_batch = batch
        with tf.GradientTape() as tape:

            # forward pass
            Z_batch = tf.add(tf.matmul(X_batch, W1), b1)
            H_batch = tf.math.sigmoid(Z_batch)
            Out_batch = tf.add(tf.matmul(H_batch, W2), b2)
            loss = tf.losses.MSE(Y_batch, Out_batch)

            # backward pass and gradient update
            gradients = tape.gradient(loss, [W1, W2, b1, b2])
            optimizer.apply_gradients(zip(gradients, [W1, W2, b1, b2]))
```

In Keras:

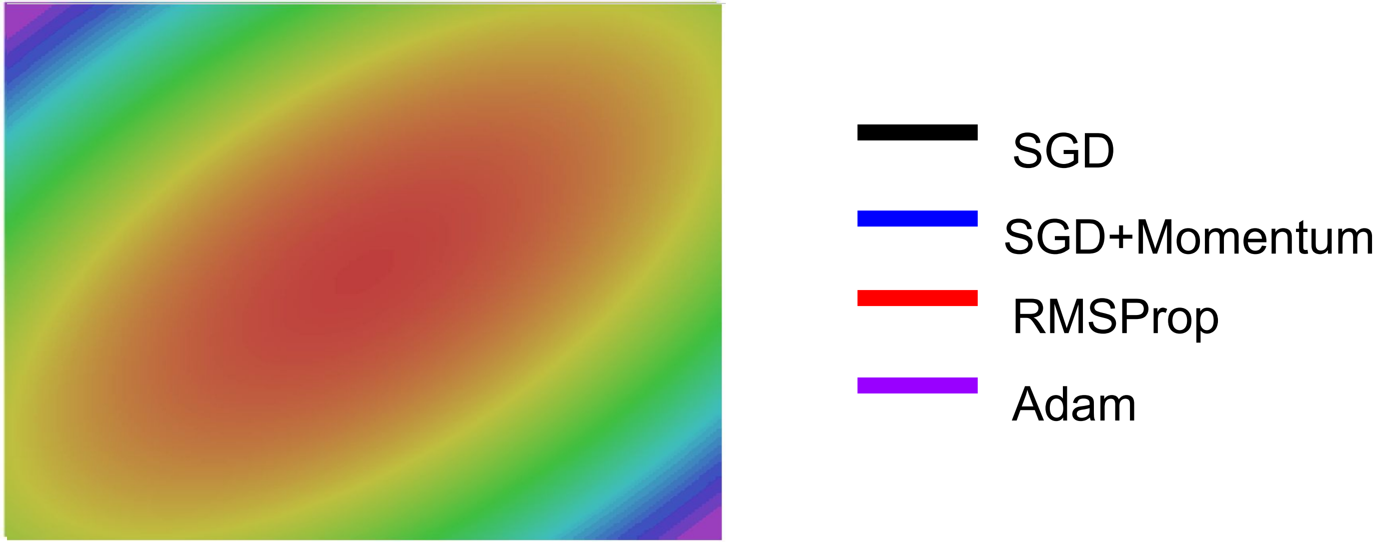
```python
keras_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=3, activation='sigmoid', use_bias=True),
    tf.keras.layers.Dense(units=1, use_bias=True)
])
keras_model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=1e-2),
          loss='mse')
keras_model.fit(dataset, epochs=1000)
```

Specify hyperparameters for the training procedure

Training hyperparameters: control knobs for the art of training neural networks

Optimization methods: SGD, SGD with momentum, RMSProp, Adam

- **Adam** is a good default choice in many cases; it often works ok even with constant learning rate
- **SGD+Momentum** can outperform Adam but may require more tuning of LR and schedule



Note on hyperparameter discussion in Lecture 2

- In this class, we will not dive deeper into theory or derivations for these hyperparameters (this is covered in other classes), and don’t worry if you feel you do not have full understanding on these aspects

- Our goal is to develop your ability to practically use deep learning as effectively as possible to solve diverse problems in healthcare. So we have given you a “menu” of hyperparameters that you can utilize when you try to improve performance on your models

- See https://piazza.com/class/kevndkjwzyx57j?cid=20 for further clarification

Note on course lectures more generally

- Objective is to establish strong conceptual foundation for developing AI models in healthcare

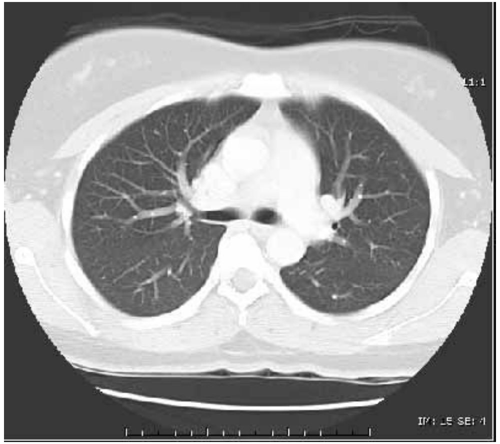
- Assignments are main measure of what you “need to know” from this class

- Lectures teach what you need to know for assignments, but may sometimes go a bit deeper. Goal is to give conceptual grounding such that you can refer back and have the foundation to explore independently in areas that you choose to dive further (e.g. for your class project or other future projects!)

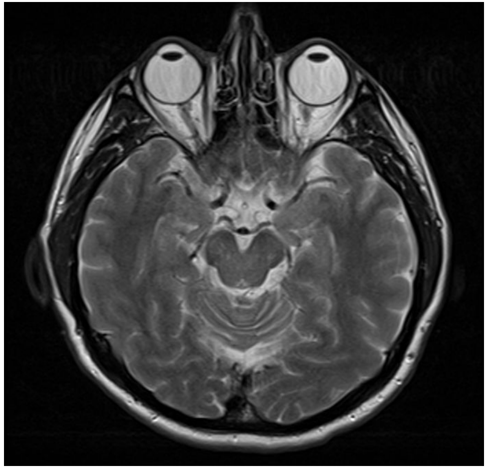
Today: Medical image data

E.g.:

X-rays (invented 1895).



CT (invented 1972).



MRI (invented 1977).

Agenda

Today: Medical Images: Classification

- Deep learning models for image classification
- Data considerations for image classification models
- Evaluating image classification models
- Case studies

Agenda

Today: Medical Images: Classification

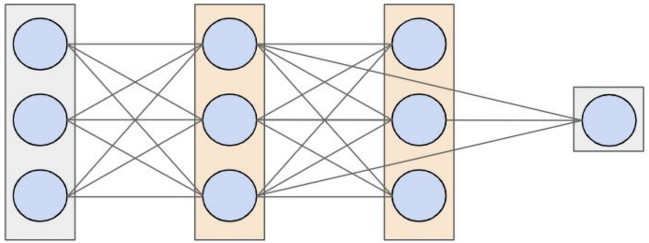
- Deep learning models for image classification
- Data considerations for image classification models
- Evaluating image classification models
- Case studies

Wed: Medical Images: Advanced Vision Models (Detection and Segmentation)

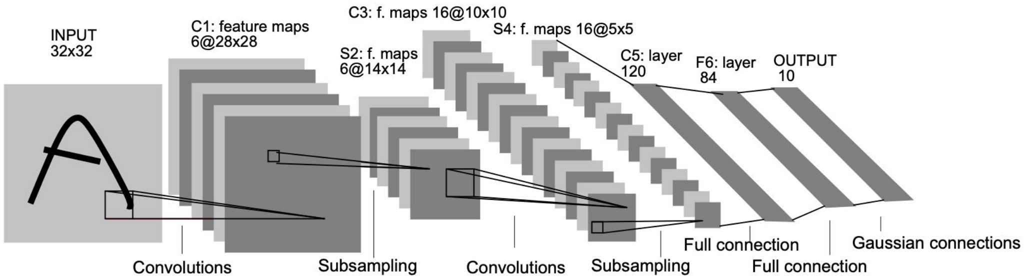
Next Mon: Medical Images: Advanced Vision Models (3D and Video)

Deep Learning Models for Image Classification

Different classes of neural networks

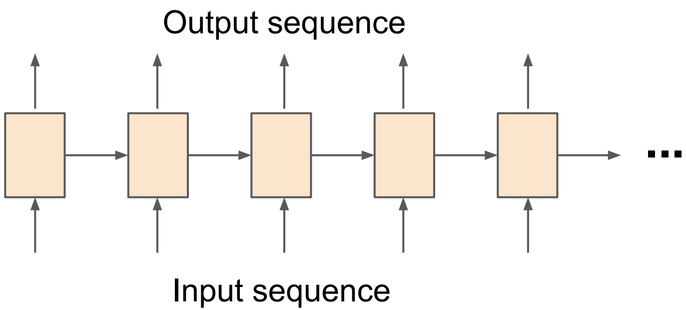


Fully connected neural networks
(linear layers, good for “feature vector” inputs)

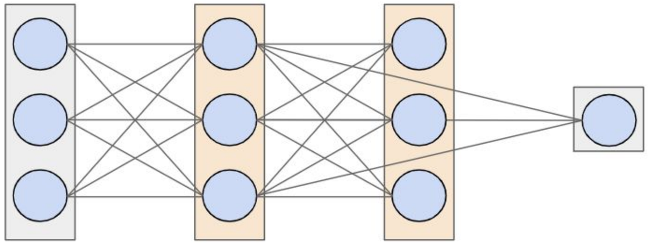


Convolutional neural networks
(convolutional layers, good for image inputs)

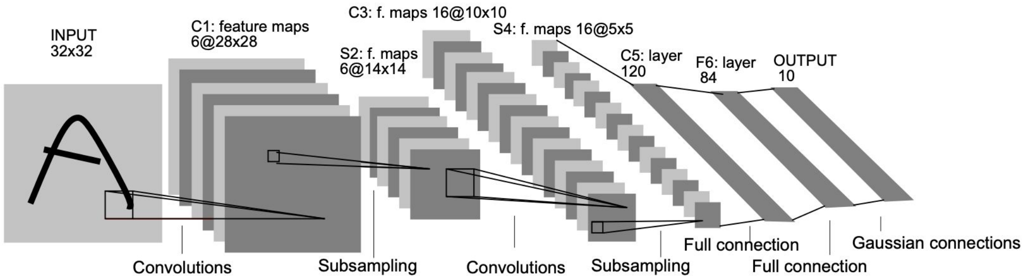
Recurrent neural networks
(linear layers modeling recurrence relation across sequence, good for sequence inputs)



Different classes of neural networks

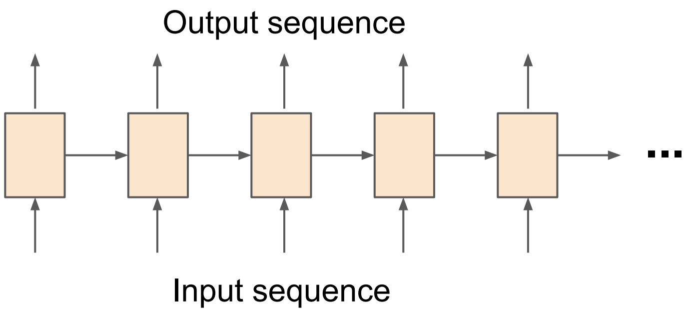


Fully connected neural networks
(linear layers, good for "feature vector" inputs)



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

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

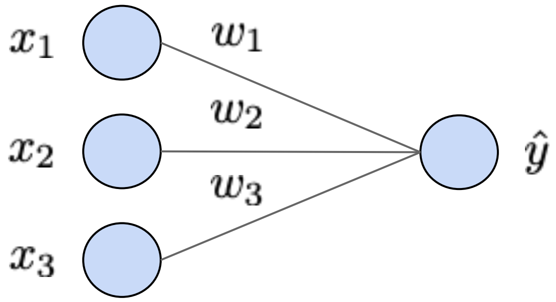


Reminder from last time: fully-connected layers

Our first architecture: a single-layer, fully connected neural network

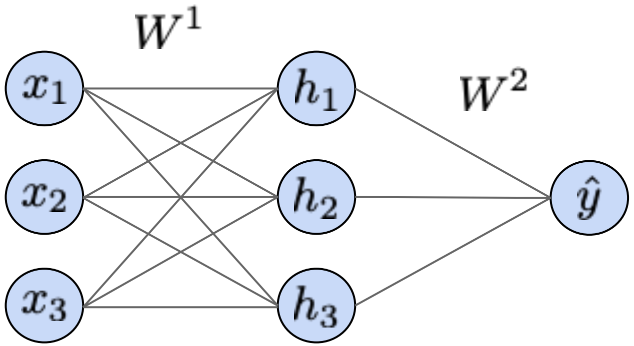
For simplicity, use a 3-dimensional input (N = 3)

all inputs of a layer are connected to all outputs of a layer



Output: $$\begin{aligned}\hat{y} &= w_1 x_1 + w_2 x_2 + w_3 x_3 + b \\ &= w^T x + b\end{aligned}$$

Two-layer fully-connected neural network



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

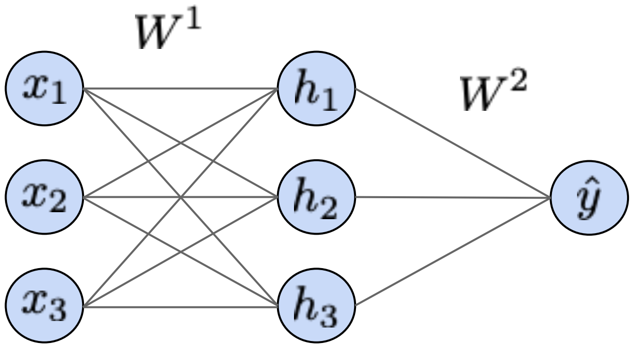
Neural network parameters:

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

$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \end{bmatrix} \quad b^1 = \begin{bmatrix} b^1_1 \\ b^1_2 \\ b^1_3 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \end{bmatrix} \quad b^2 = \begin{bmatrix} b^2_1 \end{bmatrix}$$

Two-layer fully-connected neural network



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

Neural network parameters:

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

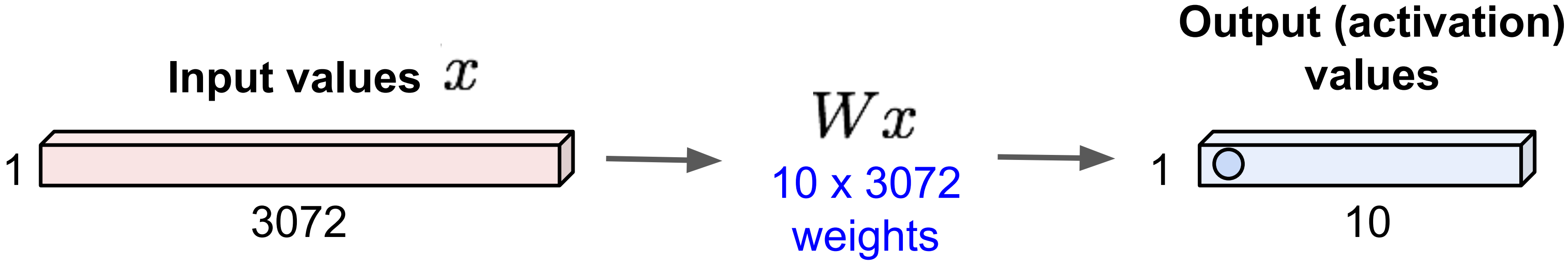
$$W^1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \end{bmatrix} \quad b^1 = \begin{bmatrix} b^1_1 \\ b^1_2 \\ b^1_3 \end{bmatrix}$$

$$W^2 = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \end{bmatrix} \quad b^2 = \begin{bmatrix} b^2_1 \end{bmatrix}$$

Dimensions of the weight matrix for each fully connected layer is [output dim. x input dim.]

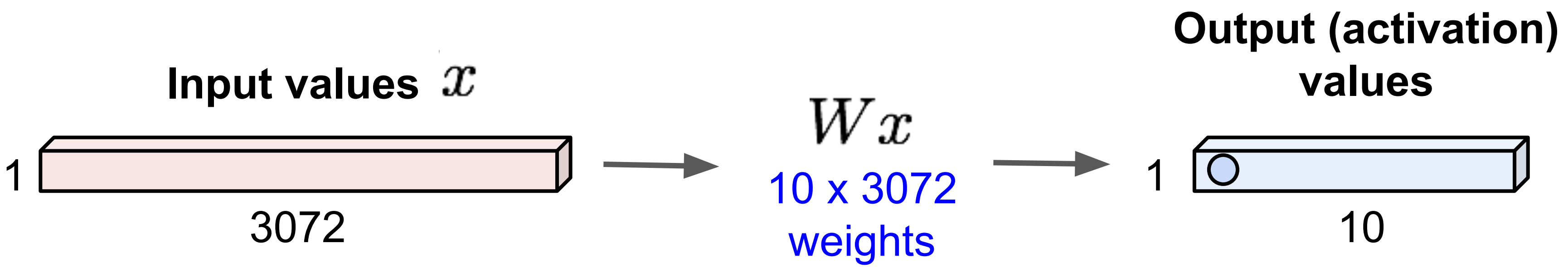
Dimensions of the bias vector is [output dim x 1]

Fully-connected layers: in graphical form



Slide credit: CS231n

Fully-connected layers: in graphical form



In Keras:

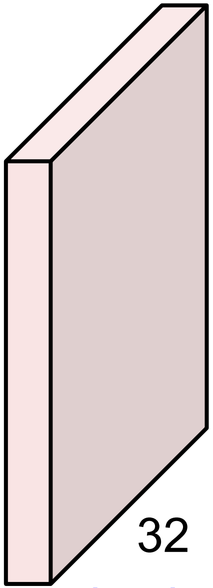
Activation function and bias configurations included!

```
keras_model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=3, activation='sigmoid', use_bias=True),
    tf.keras.layers.Dense(units=1, use_bias=True)
])
keras_model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=1e-2),
          loss='mse')
keras_model.fit(dataset, epochs=1000)
```

Slide credit: CS231n

Convolutional layer

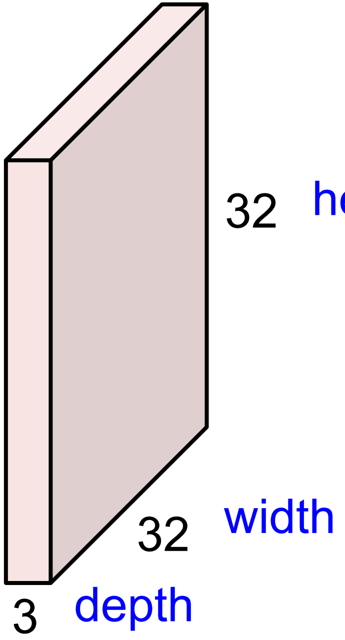
32x32x3 image -> preserve spatial structure



Slide credit: CS231n

Convolutional layer

32x32x3 image -> preserve spatial structure

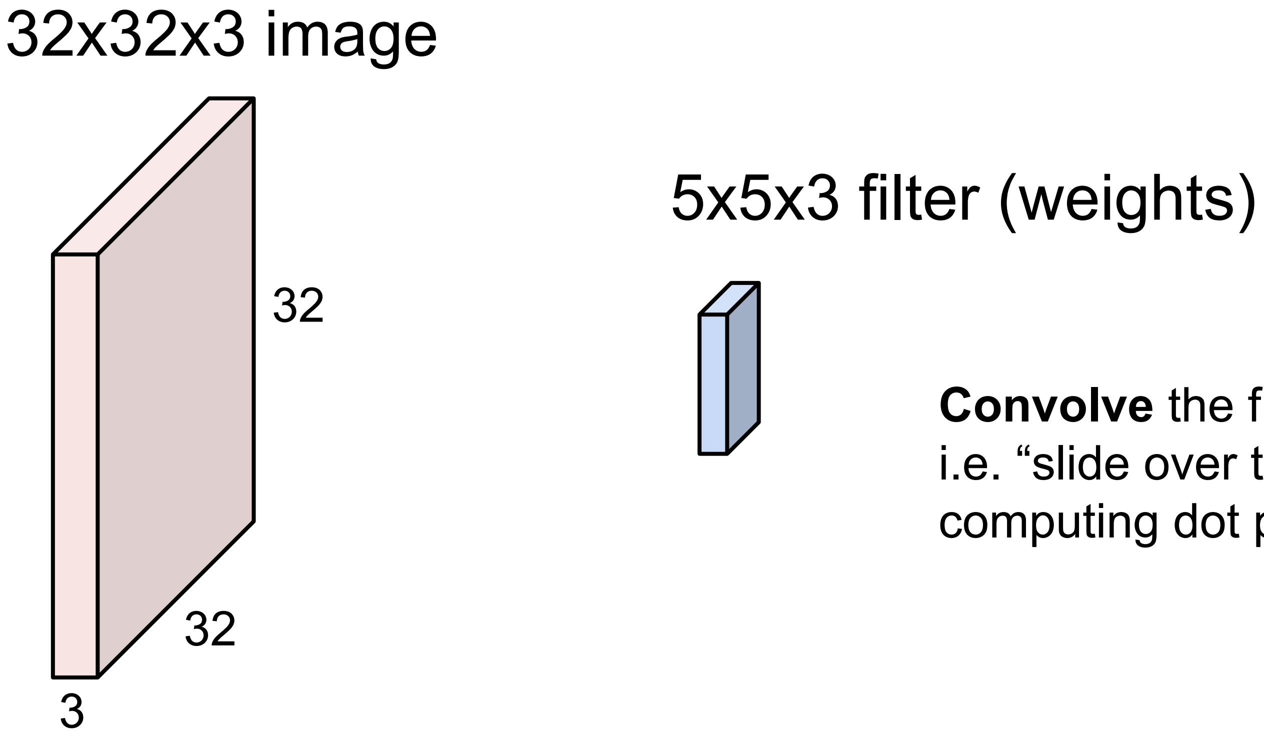


Input now has spatial height and width dimensions!

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

Slide credit: CS231n

Convolutional layer



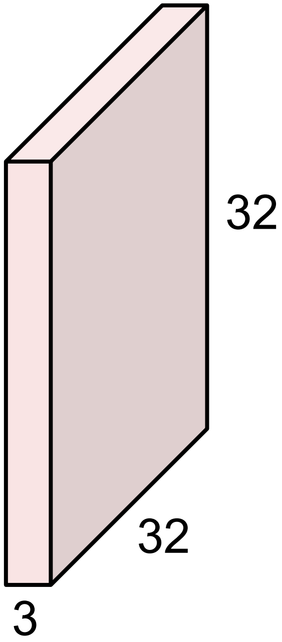
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Slide credit: CS231n

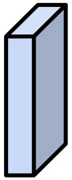
Convolutional layer

Filters always extend the full depth of the input volume

32x32x3 image



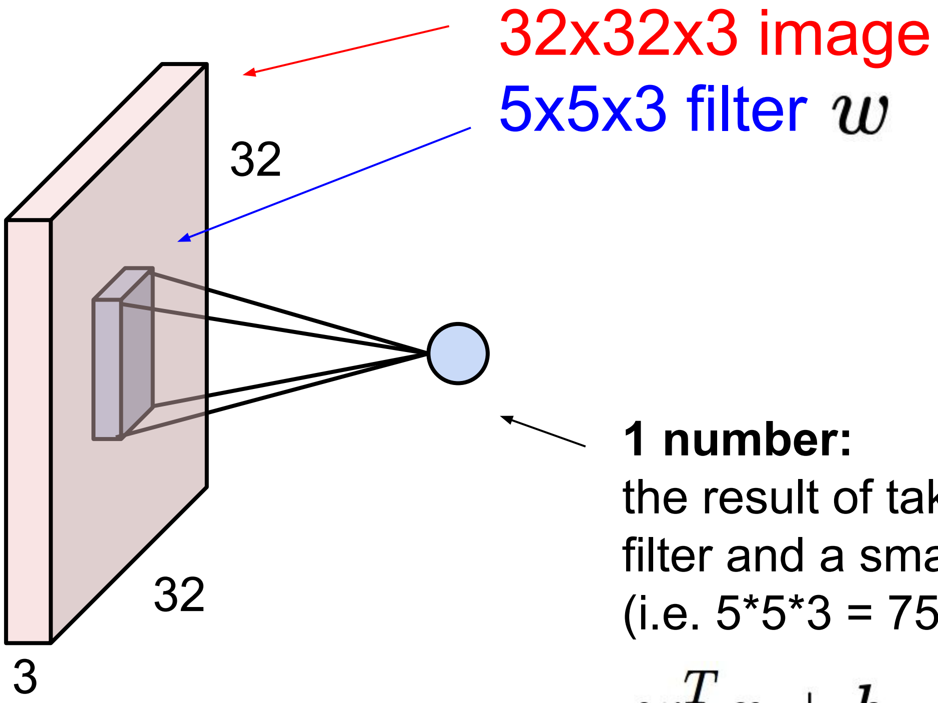
5x5x3 filter (weights)



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

Slide credit: CS231n

Convolutional layer

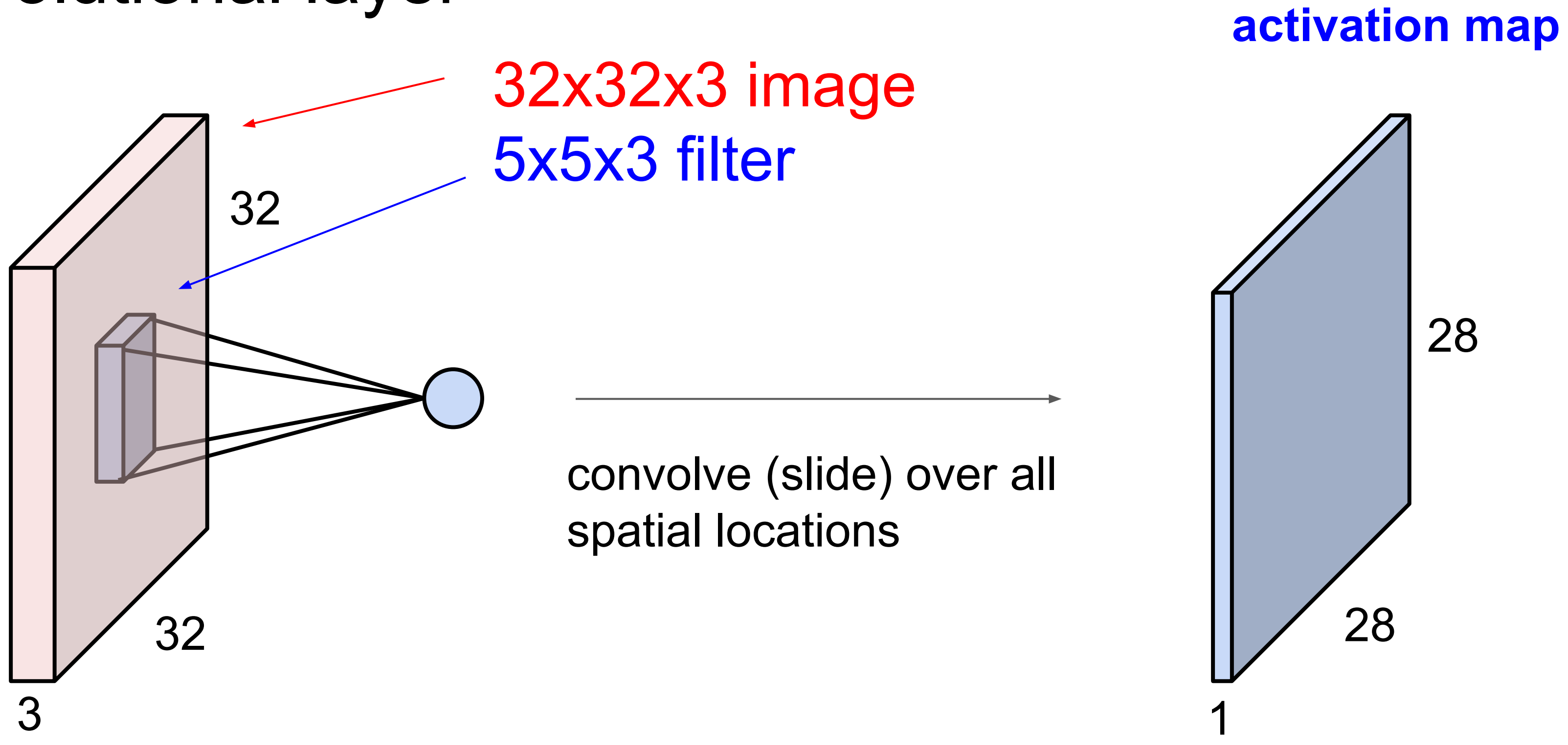


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: CS231n

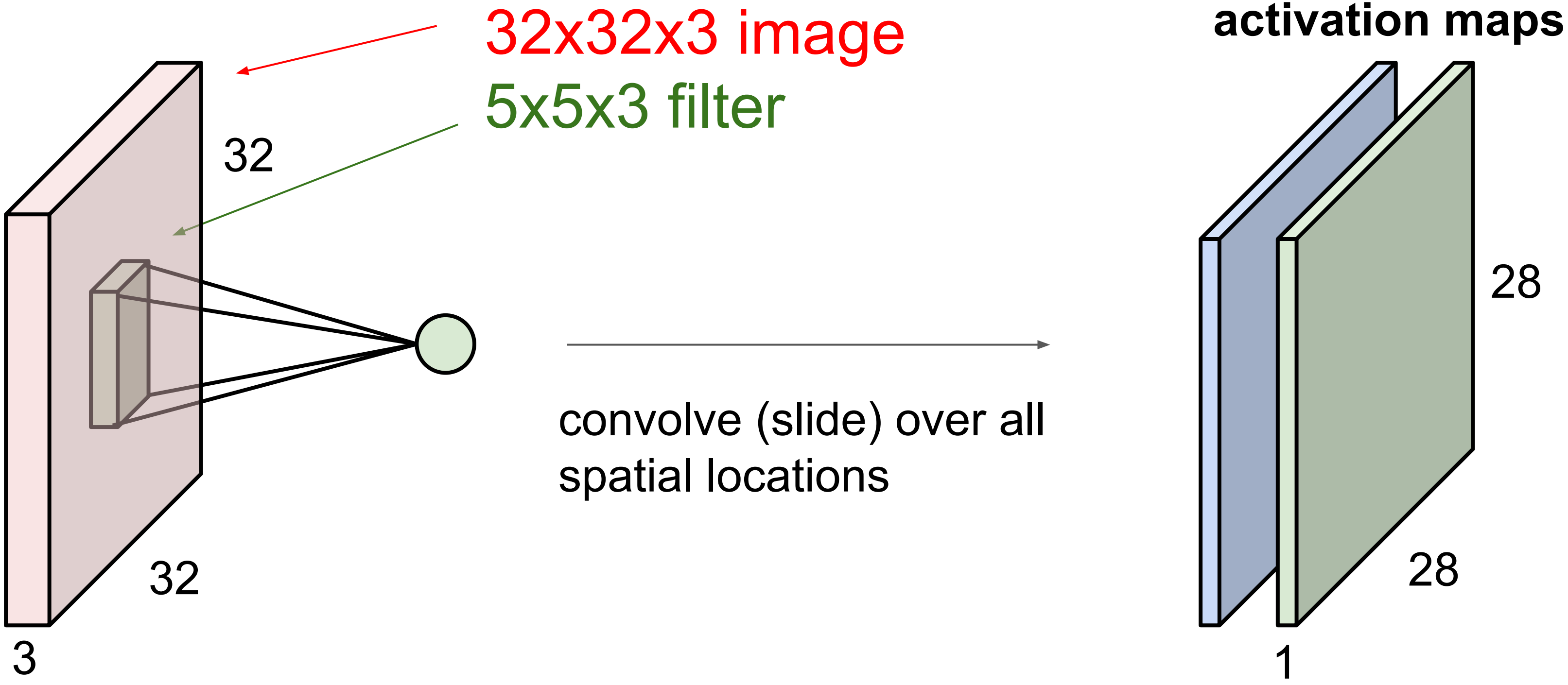
Convolutional layer



Slide credit: CS231n

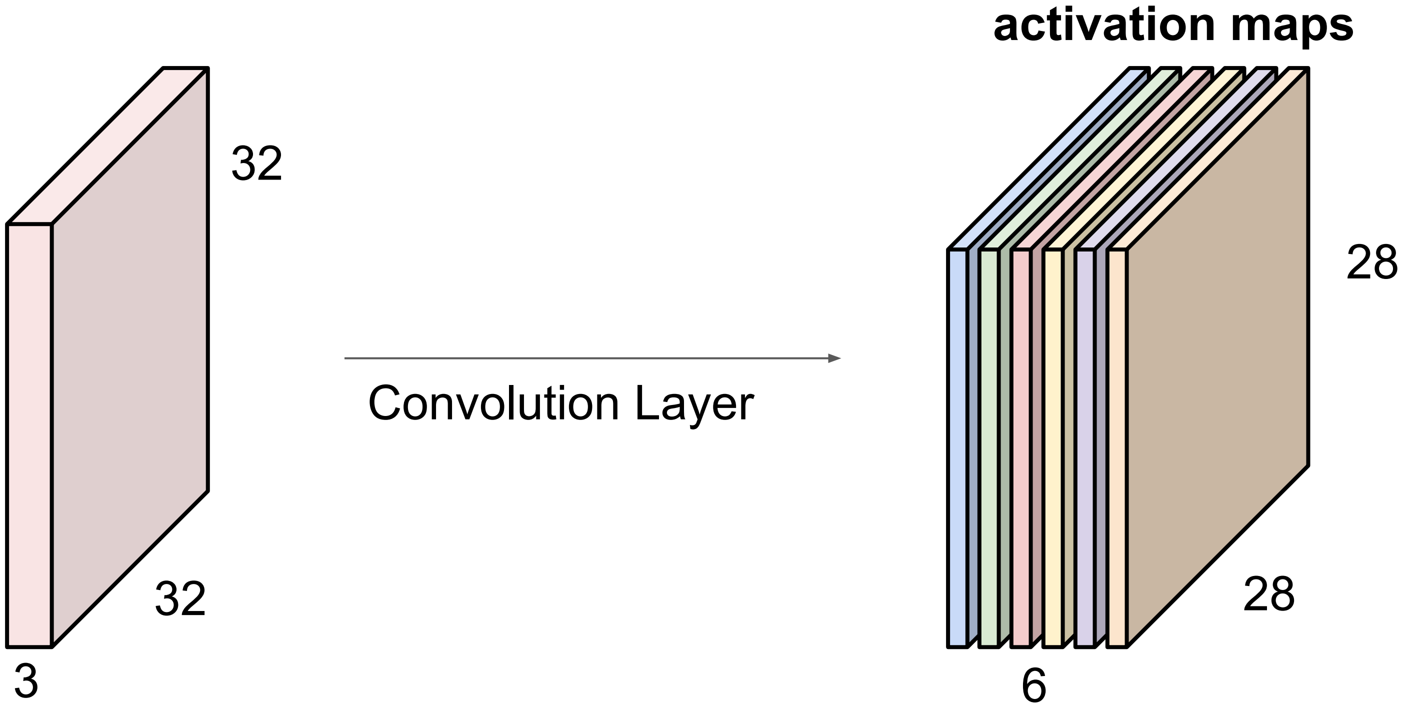
Convolutional layer

consider a second, green filter



Slide credit: CS231n

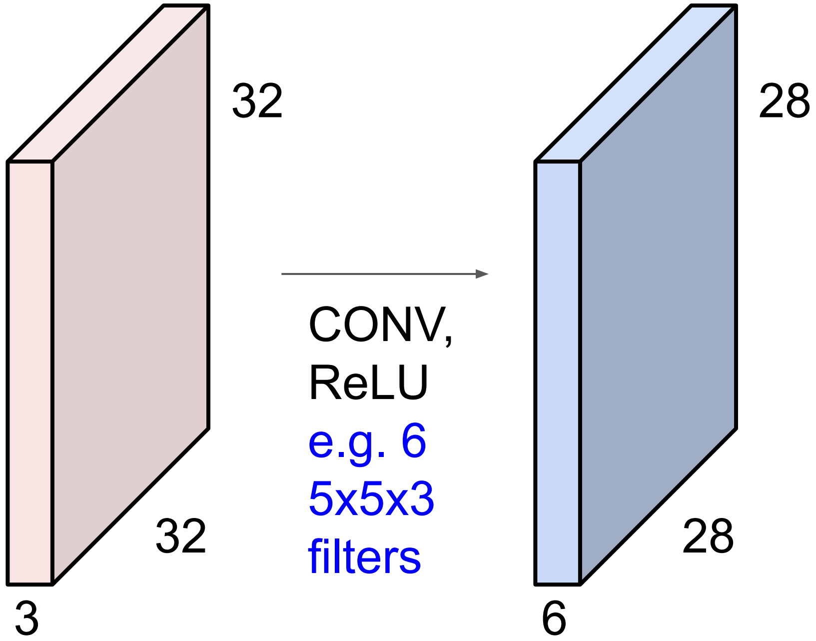
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

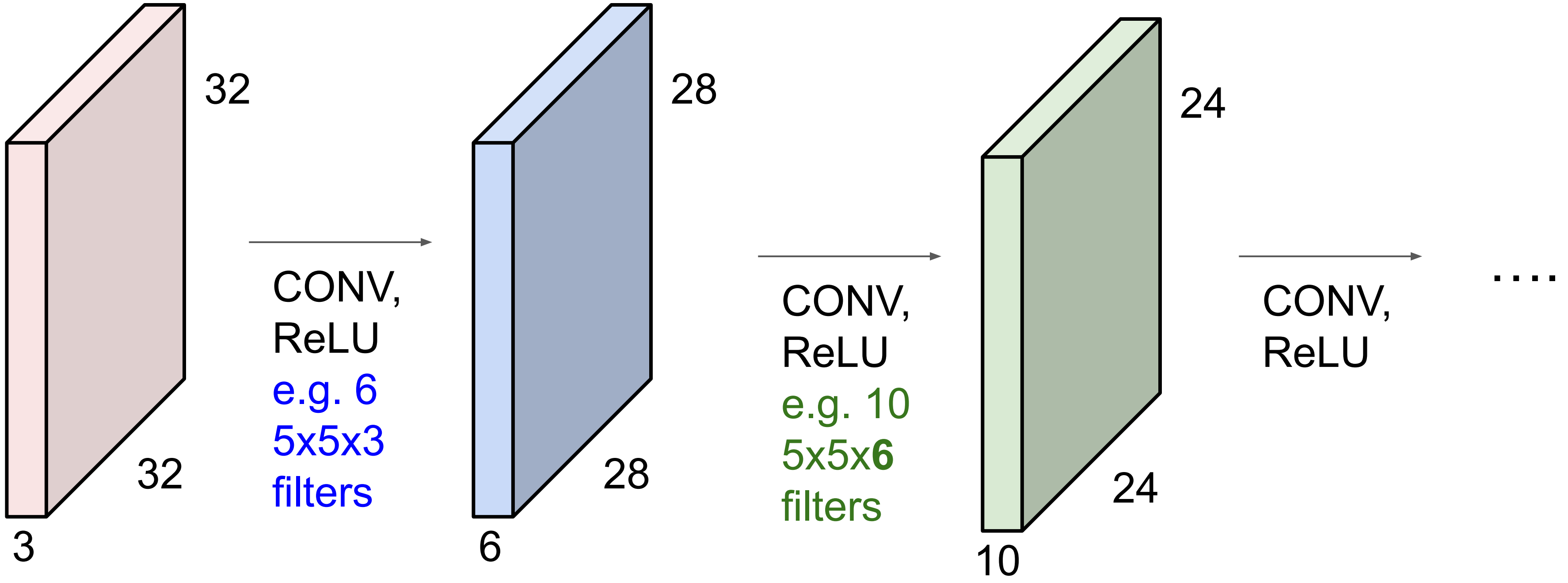
Slide credit: CS231n

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

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.

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

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

Conv2D

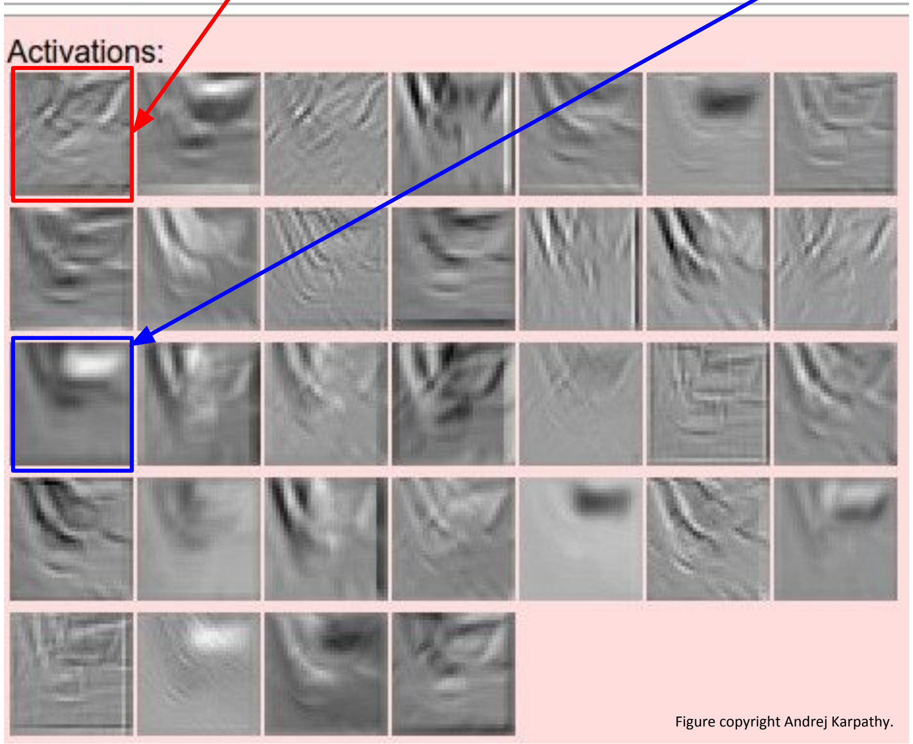
[source]

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

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

one filter =>
one activation map

example 5x5 filters
(32 total)



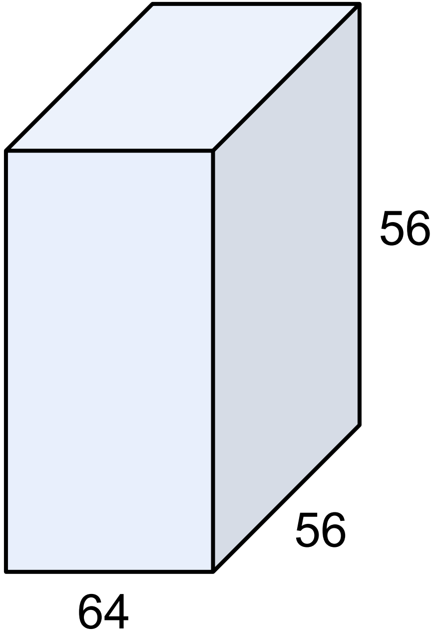
Aside: We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

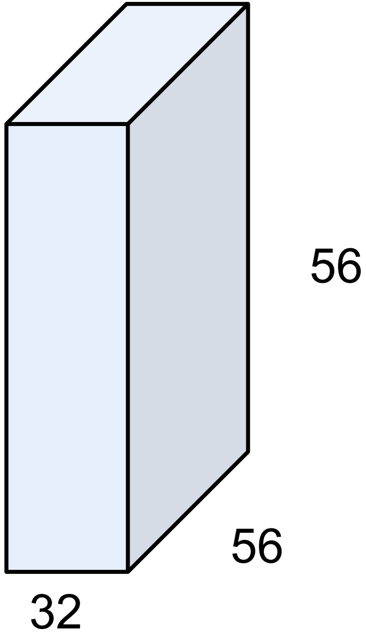
Slide credit: CS231n

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



1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

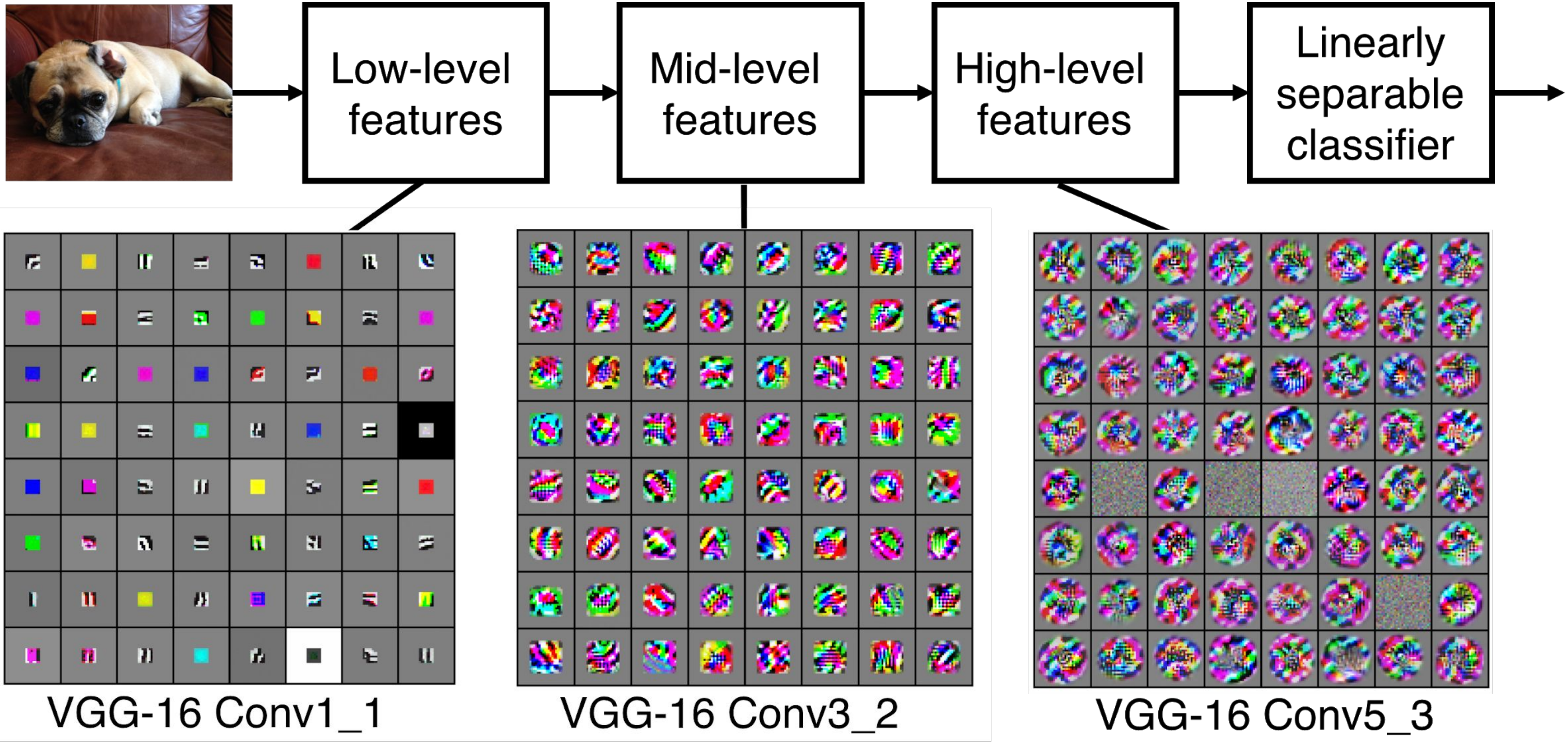


Slide credit: CS231n

Preview: can visualize learned features

[Zeiler and Fergus 2013]

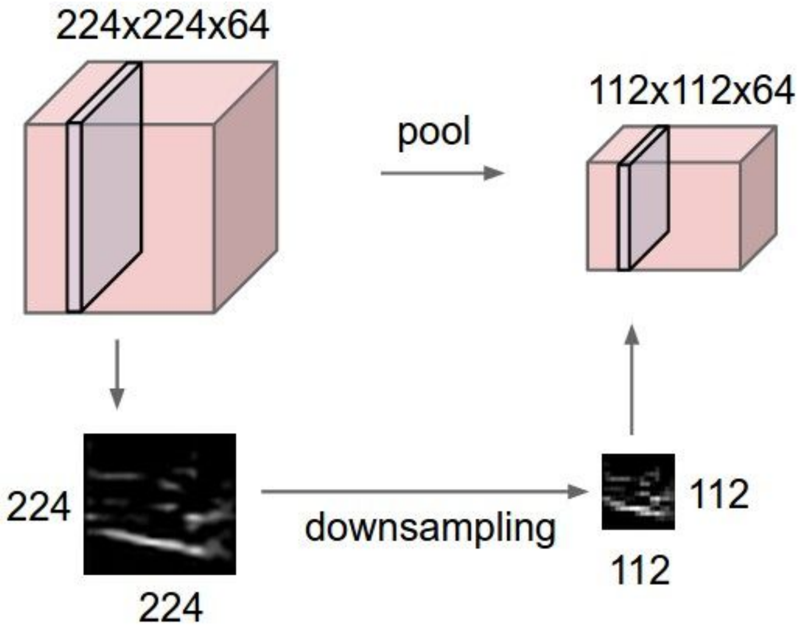
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



Slide credit: CS231n

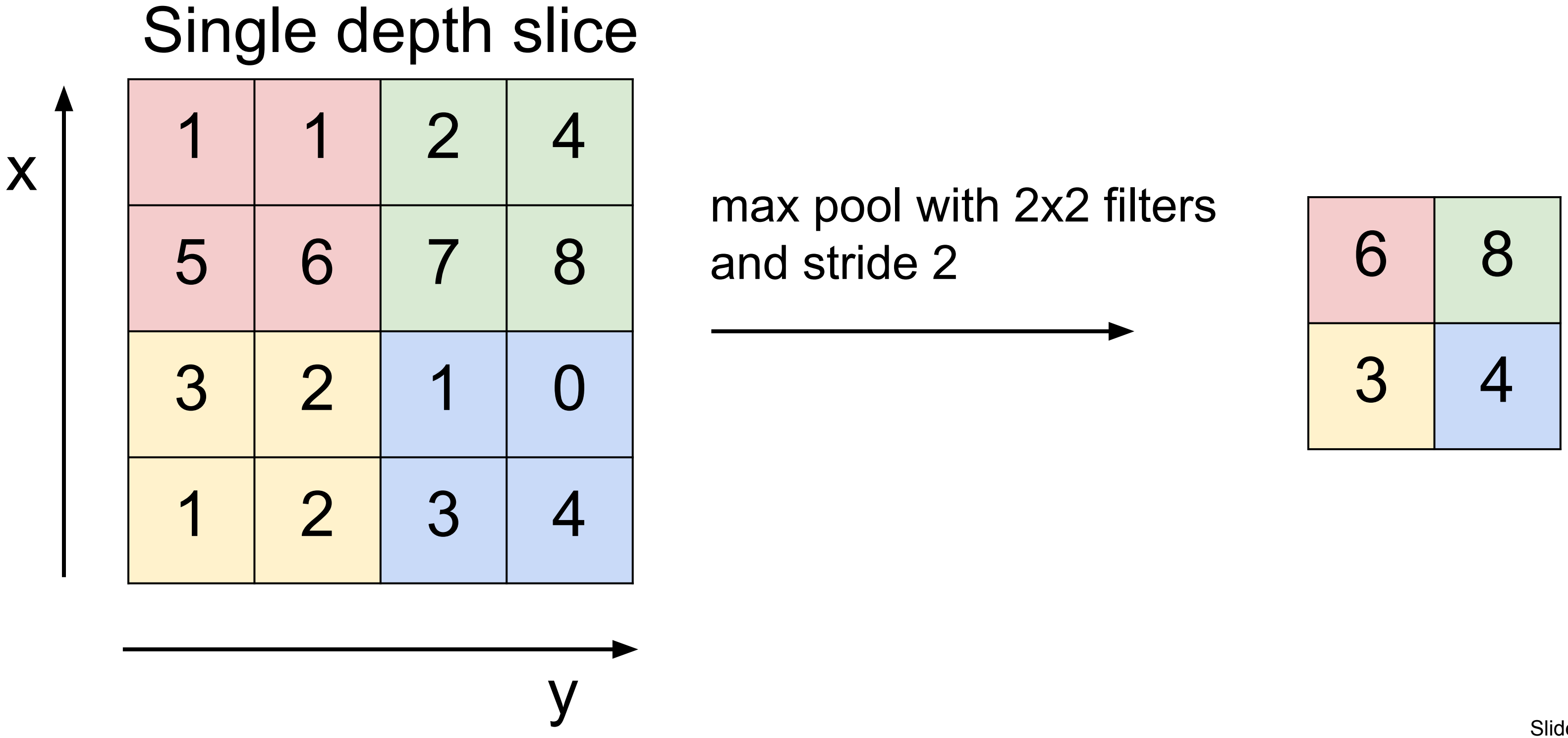
Pooling layer

- 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:

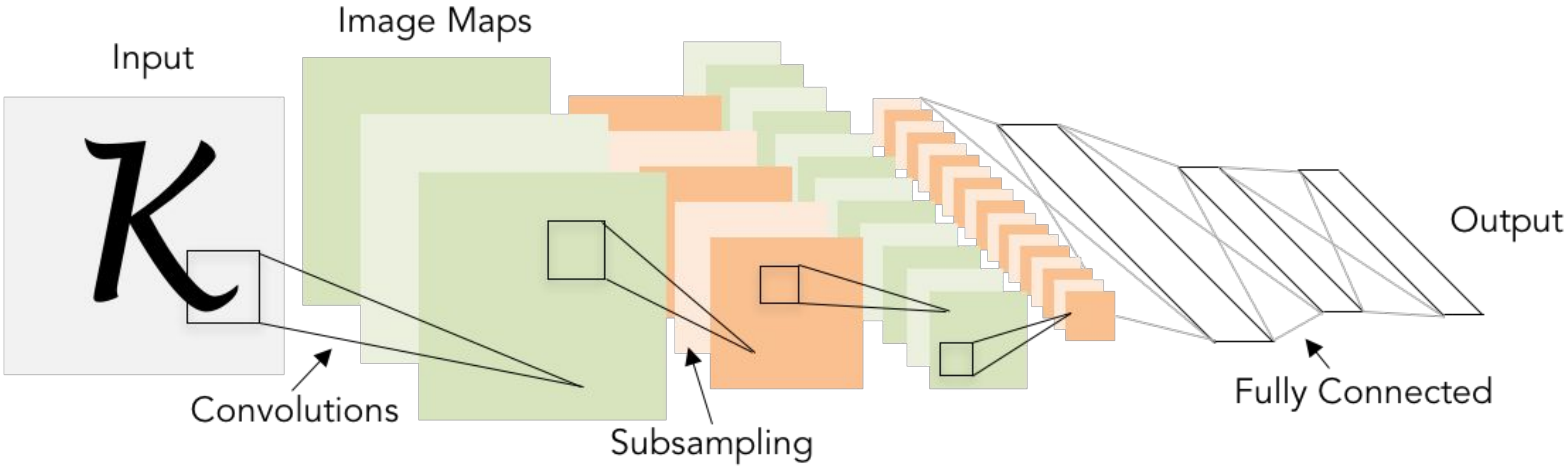
MaxPooling2D [source]

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

Slide credit: CS231n

LeNet-5

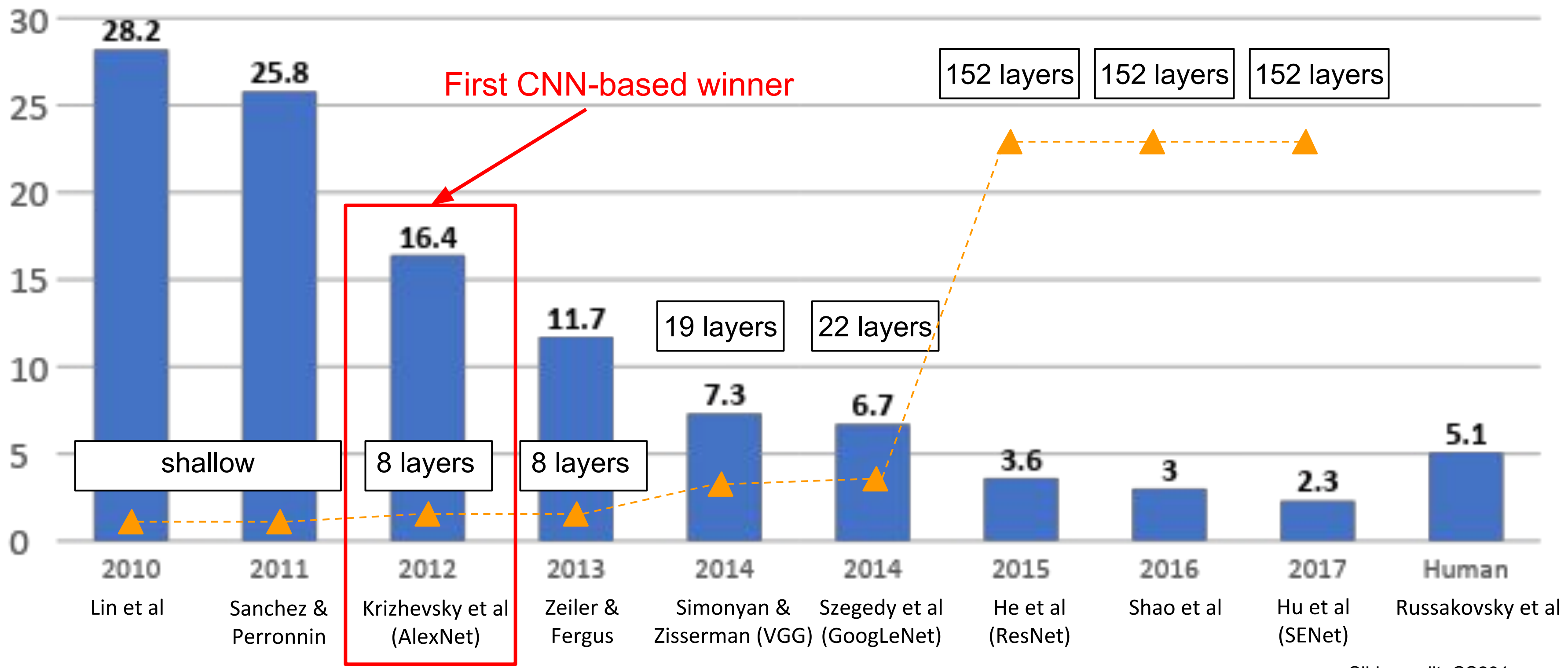
[LeCun et al., 1998]



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]

Slide credit: CS231n

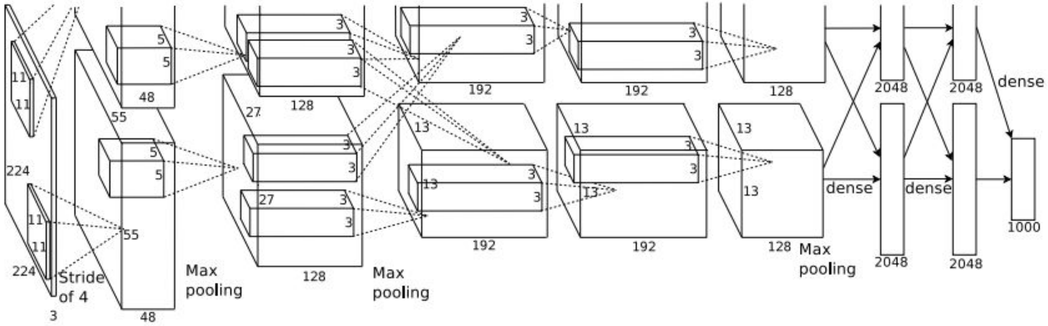
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide credit: CS231n

AlexNet

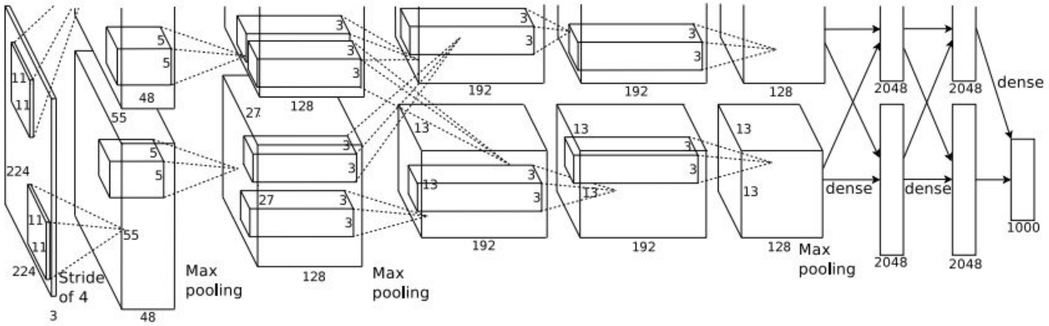
[Krizhevsky et al. 2012]



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)

AlexNet

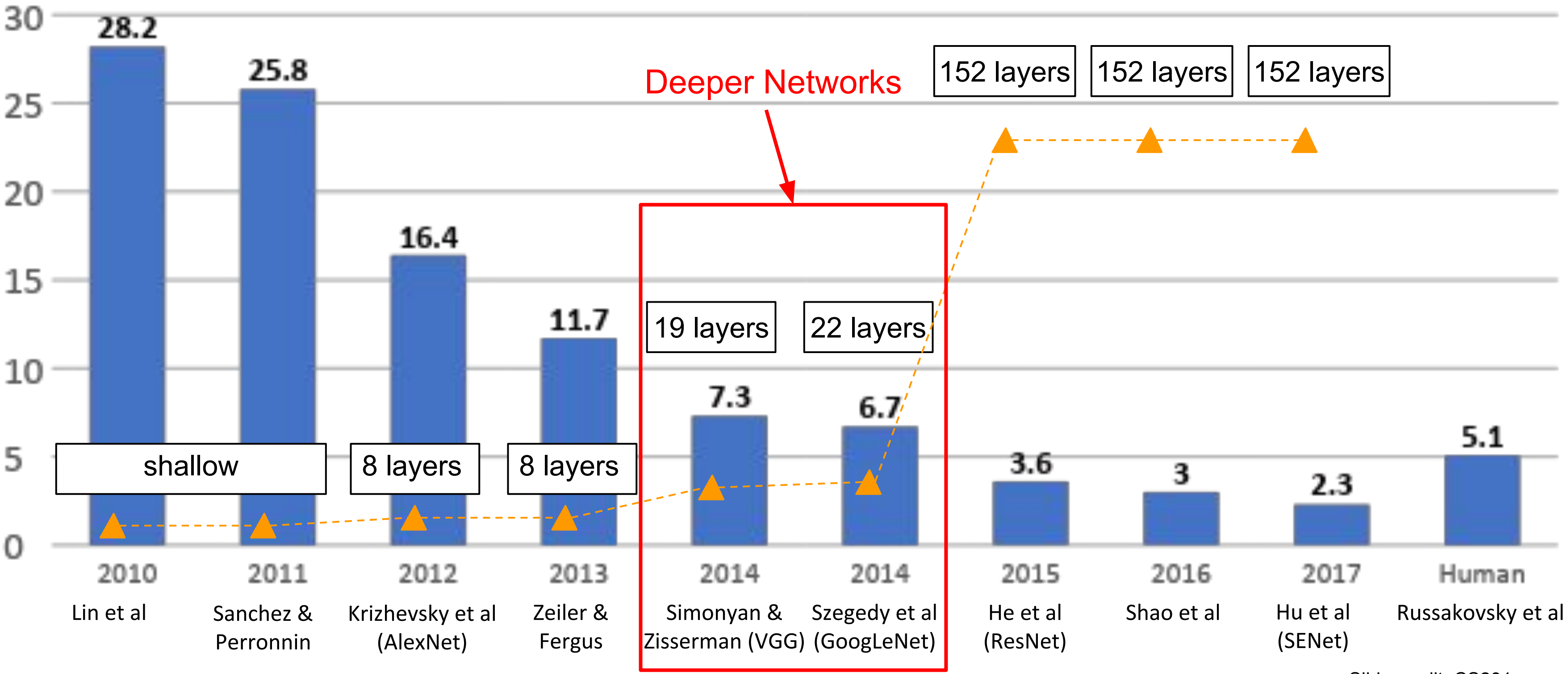
[Krizhevsky et al. 2012]



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/Retrospectives:
- 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
- 7 CNN ensemble: 18.2% -> 15.4%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Slide credit: CS231n

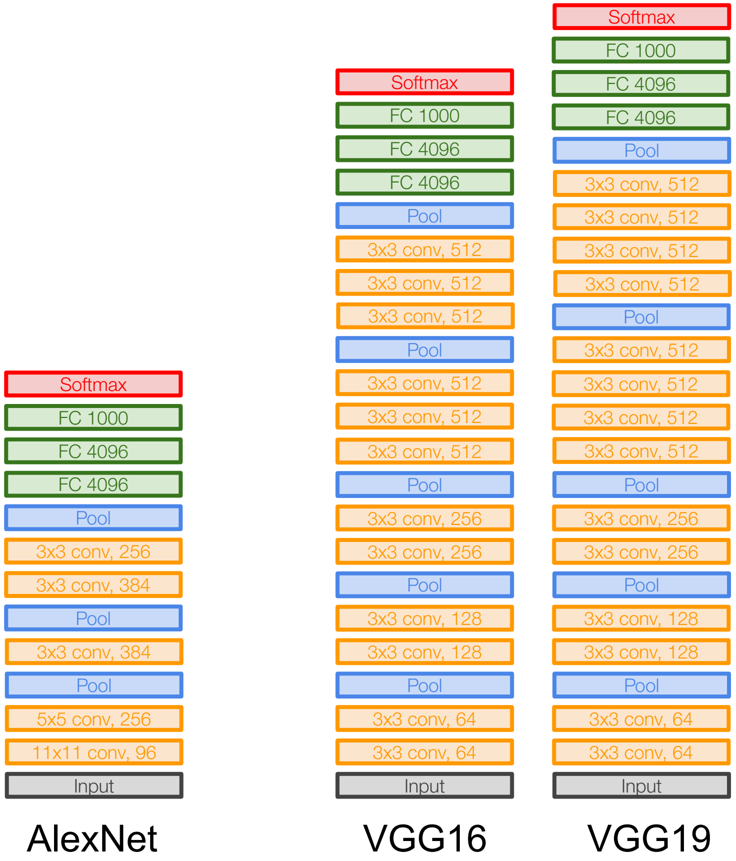
VGGNet

[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

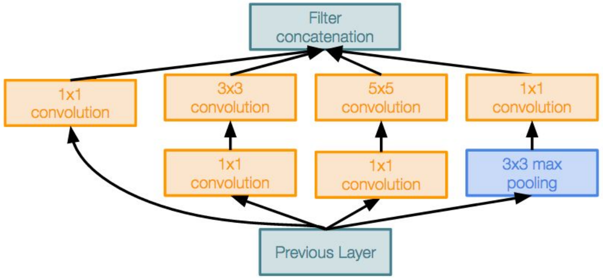


Slide credit: CS231n

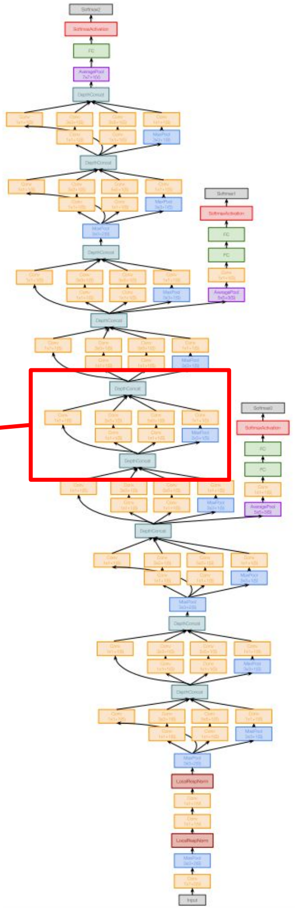
GoogLeNet

[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



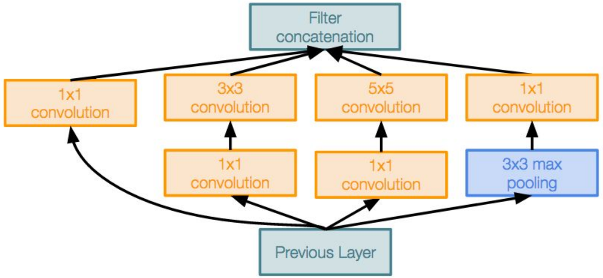
Slide credit: CS231n

GoogLeNet

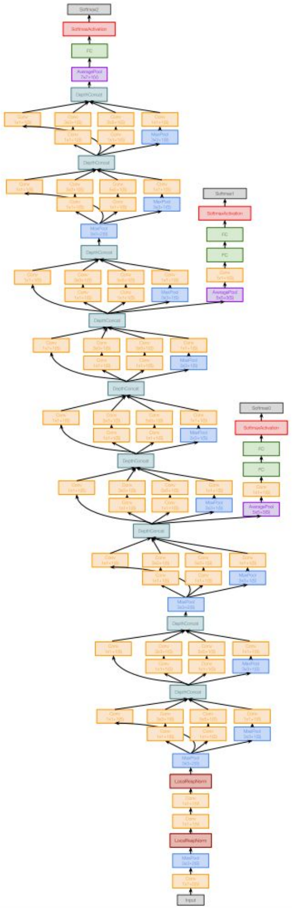
[Szegedy et al., 2014]

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



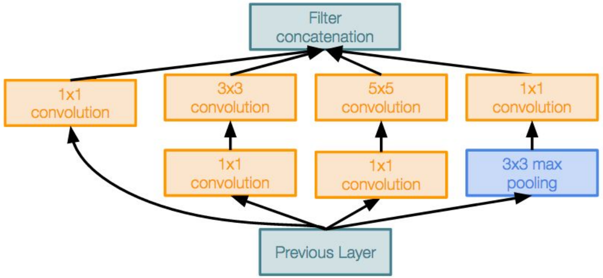
Slide credit: CS231n

GoogLeNet

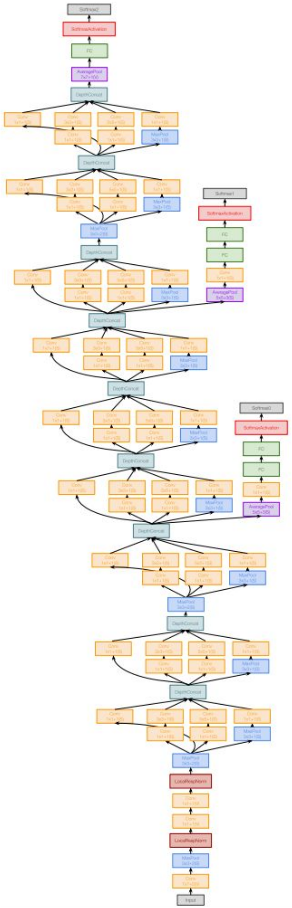
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

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



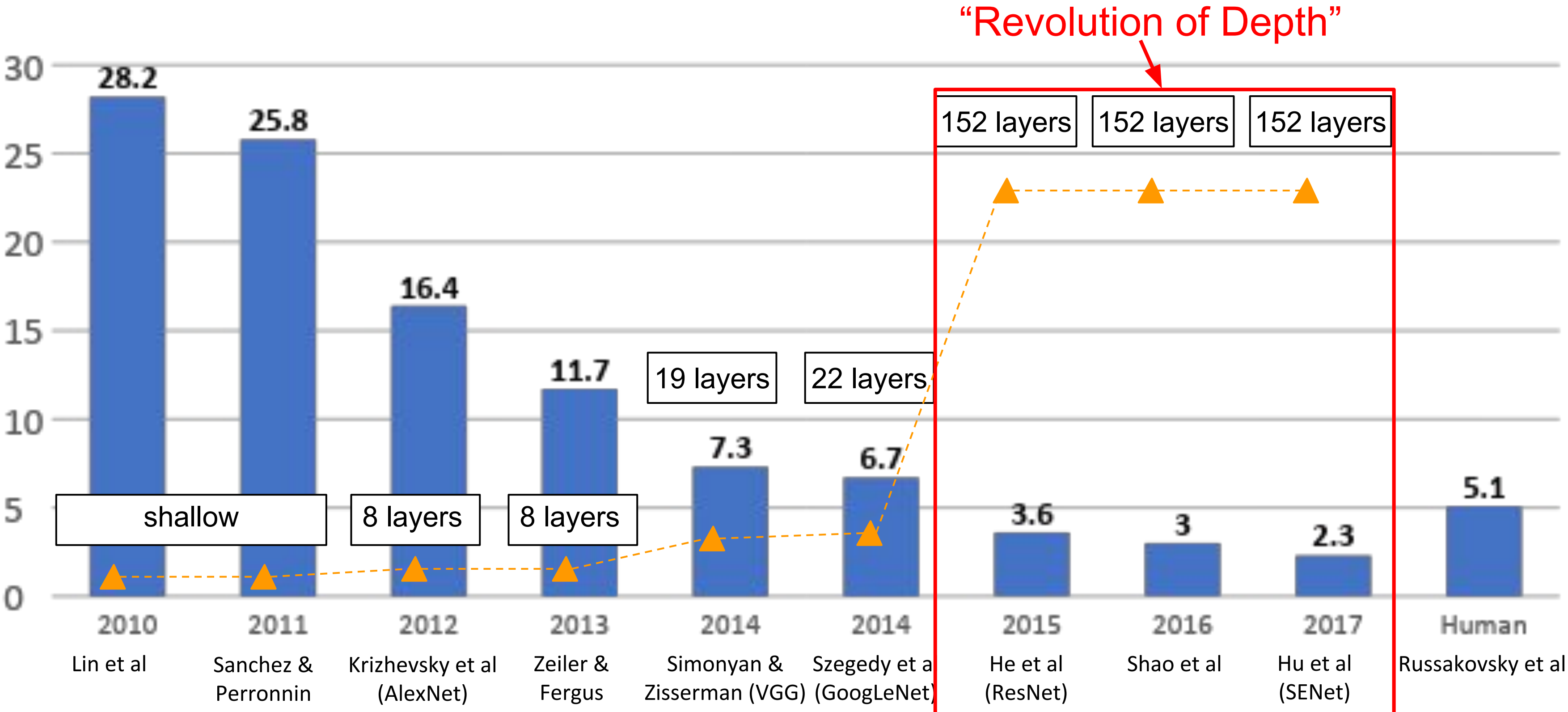
Inception module



Also called “Inception Network”

Slide credit: CS231n

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



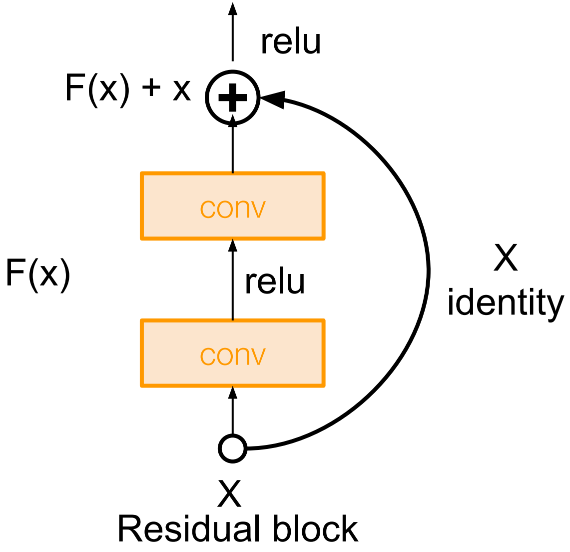
Slide credit: CS231n

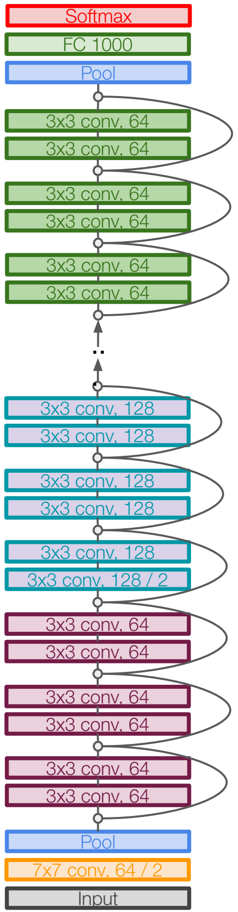
ResNet

[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



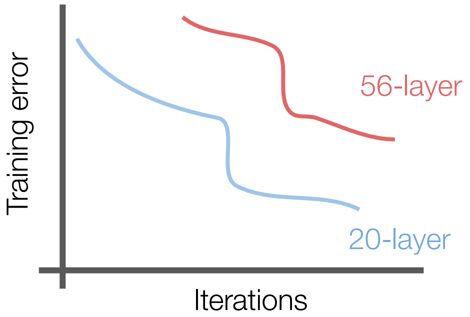


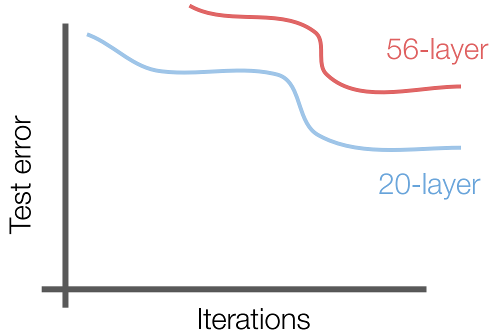
Slide credit: CS231n

ResNet

[He et al., 2015]

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





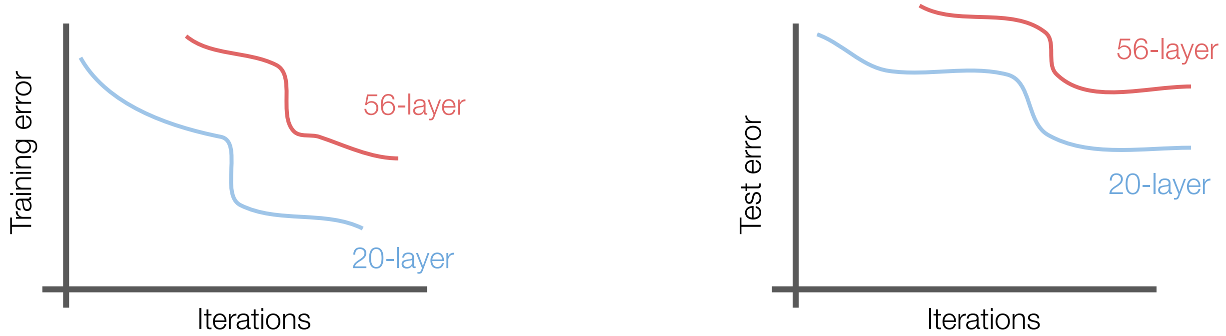
Q: What's strange about these training and test curves?
[Hint: look at the order of the curves]

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.

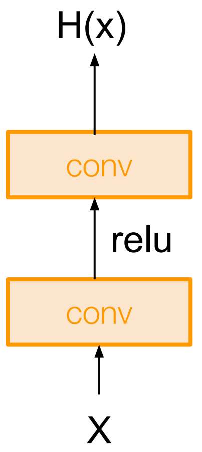
A solution by construction is copying the learned layers over from the shallower model and setting all additional layers to the **identity** function.

Slide credit: CS231n

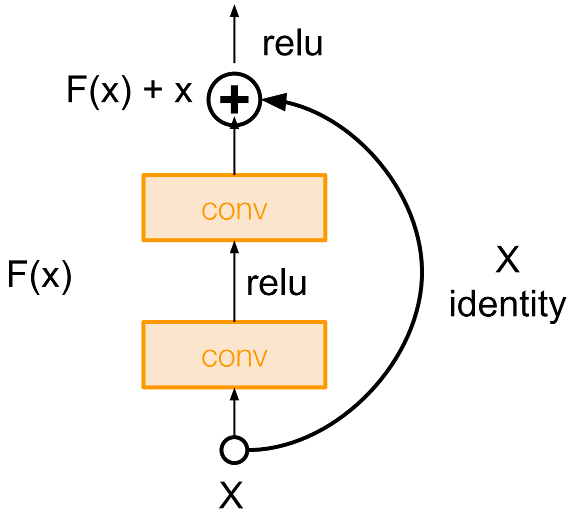
ResNet

[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



“Plain” layers



Residual block

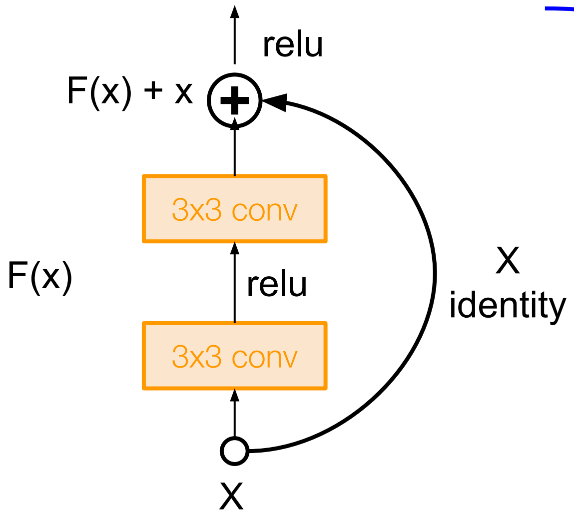
Slide credit: CS231n

ResNet

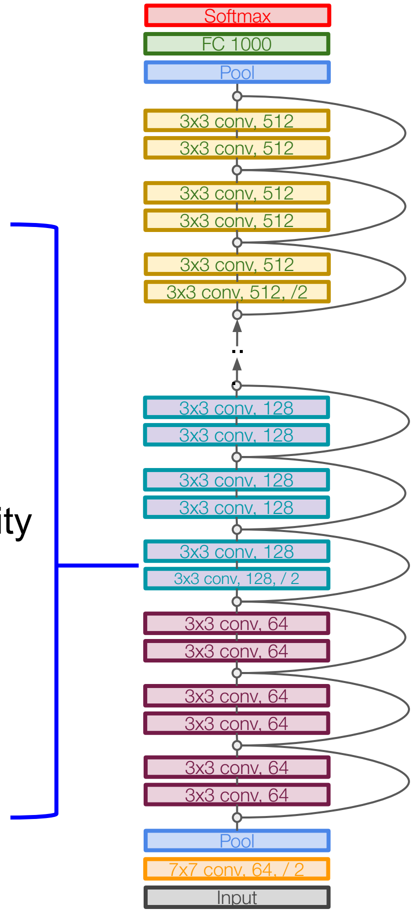
[He et al., 2015]

Full ResNet architecture:

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



Residual block



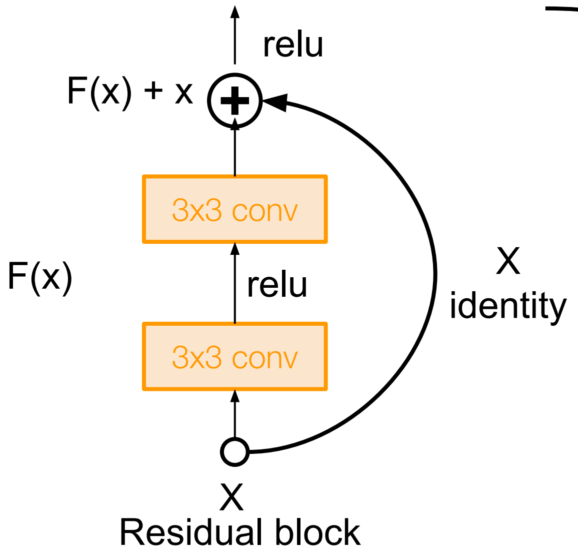
Slide credit: CS231n

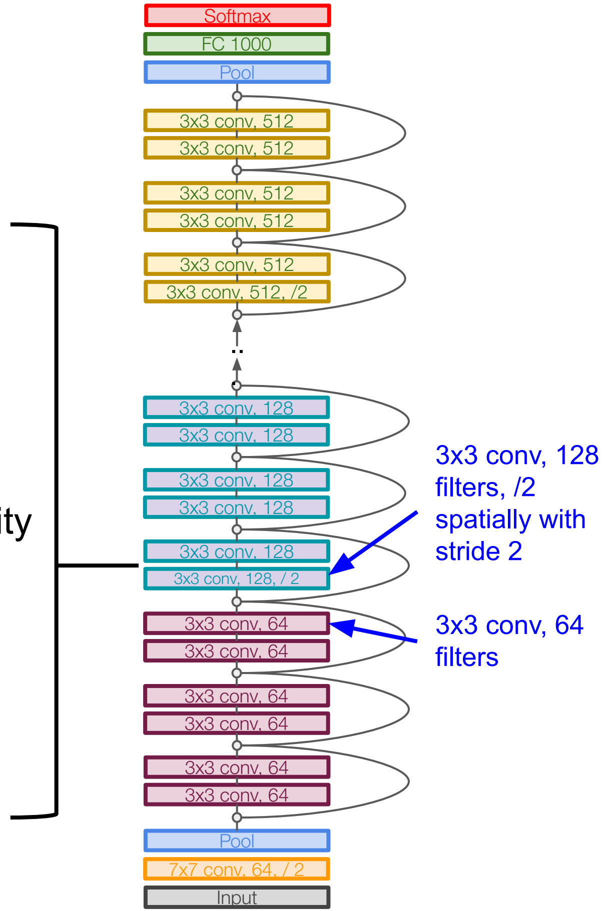
ResNet

[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)



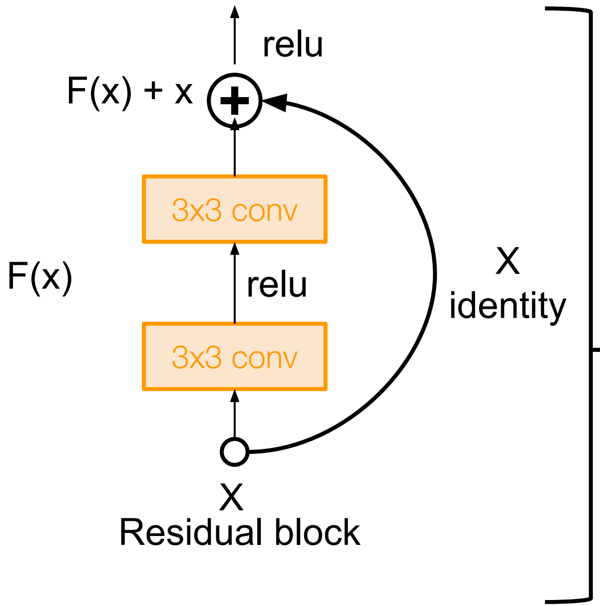


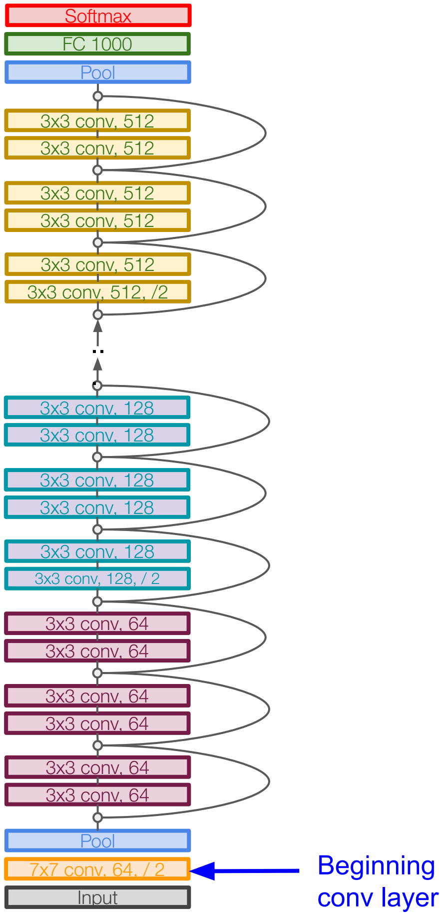
Slide credit: CS231n

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





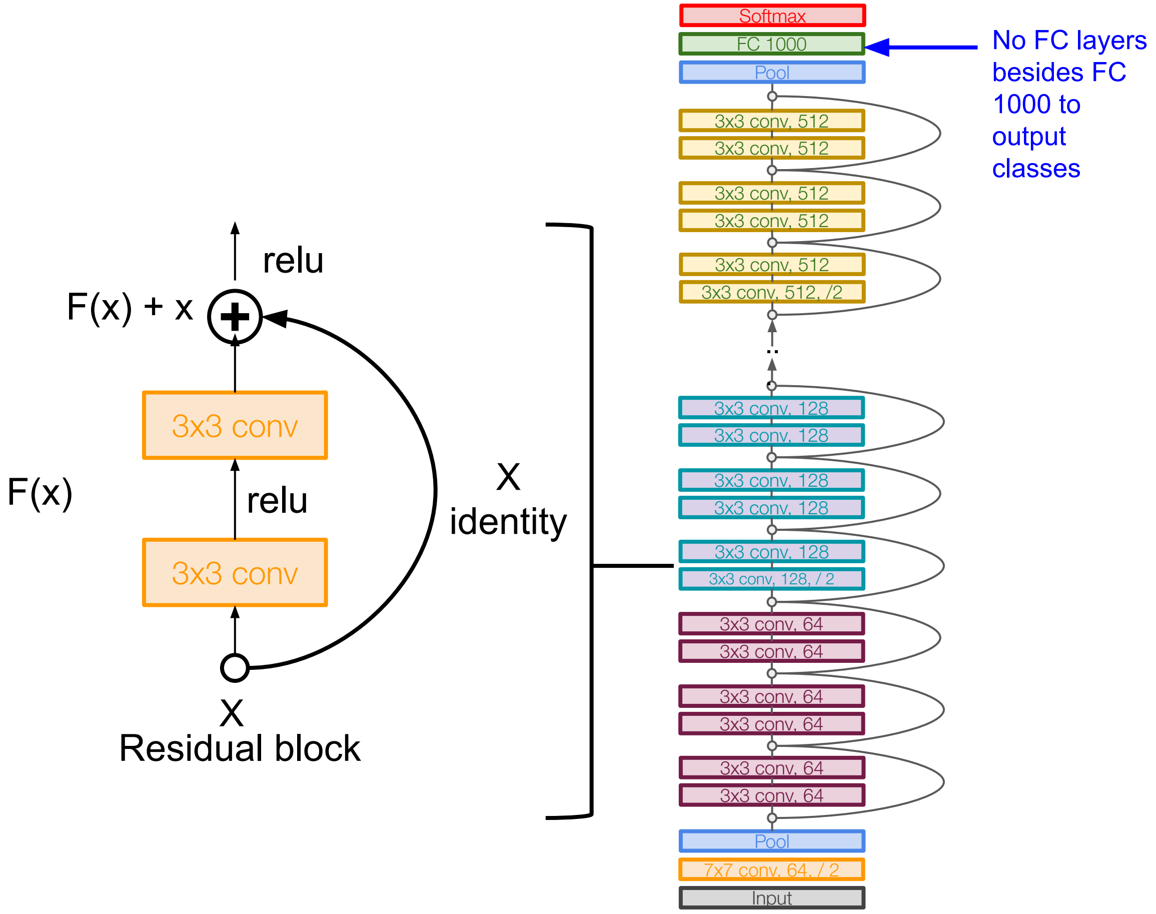
Slide credit: CS231n

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)

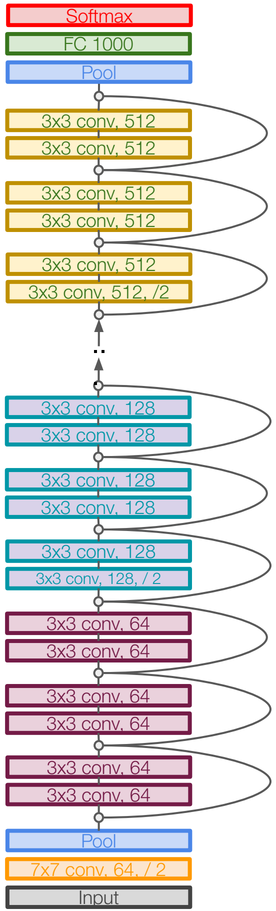


Slide credit: CS231n

ResNet

[He et al., 2015]

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



Slide credit: CS231n

More on loss functions

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

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

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.

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.

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.

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

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.

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.

Binary Cross-Entropy

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.

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

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.

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

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.

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

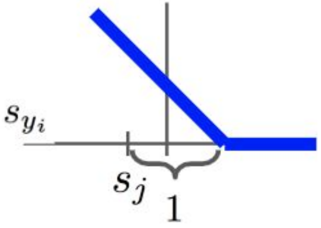
```
keras.losses.categorical_crossentropy(y_true, y_pred, from_logits=False, label_smoothing=0)
```

→ 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

Training (50%)	Validation (30%)	Test (20%)
	Held-out evaluation set for selecting best hyperparameters during training	Do not use until final evaluation

Training, validation, and test sets

Training (50%)	Validation (30%)	Test (20%)

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

“Trainval” (70%)	Test (30%)

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.

Fold 1	Fold 2	Fold 3	Fold 4	Test
Fold 1	Fold 2	Fold 3	Fold 4	Test
Fold 1	Fold 2	Fold 3	Fold 4	Test
Fold 1	Fold 2	Fold 3	Fold 4	Test

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.

Data preprocessing

Min-max scaling:

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

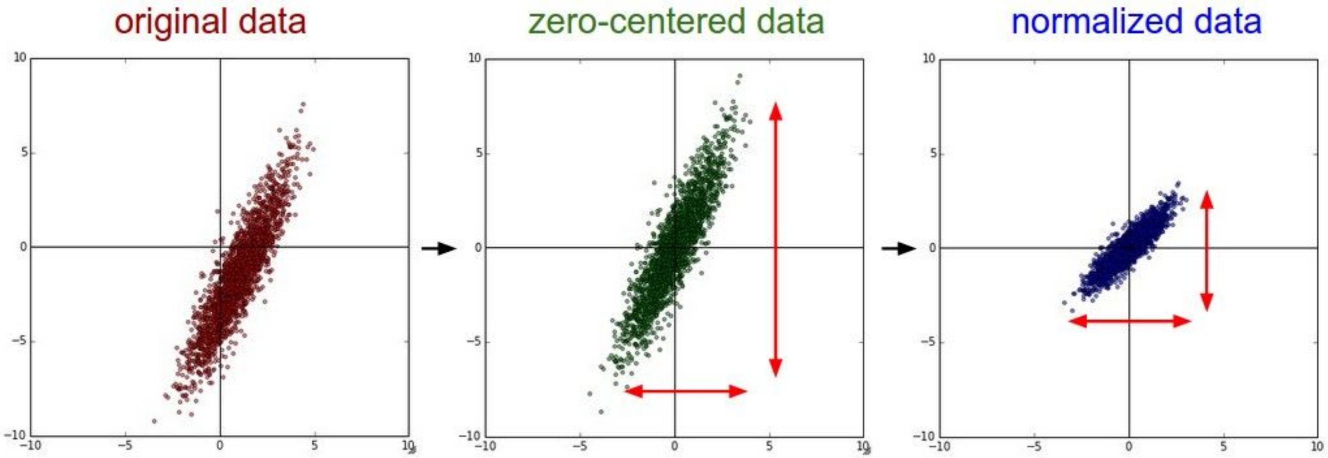
where x_min and x_max are min and max values in the original data

- 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.




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

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

Slide credit: CS231n

Data preprocessing: for images

For images, common to perform simpler normalization:

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

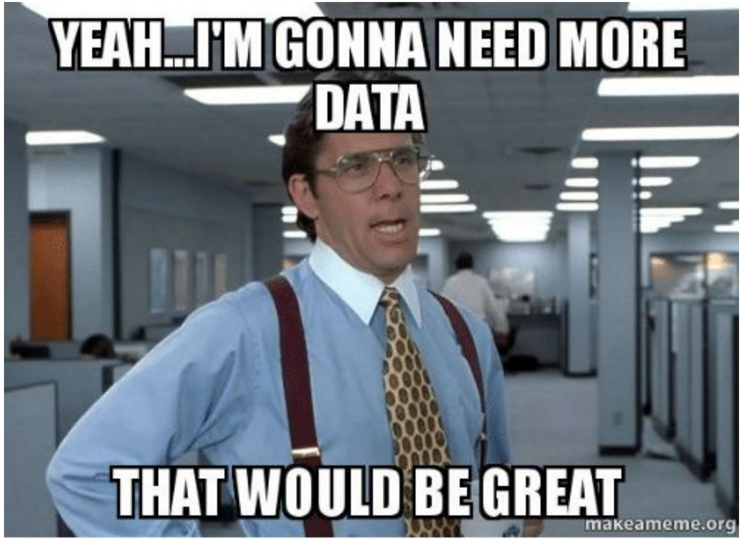
- Subtract the mean image (used in original AlexNet model)
  (mean image = [32,32,3] array)
- Subtract per-channel mean (used in original VGG model)
  (mean along each channel = 3 numbers)
- Subtract per-channel mean and
  Divide by per-channel std (used in original ResNet model)
  (mean along each channel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

Slide credit: CS231n

How much data do you need for deep learning?

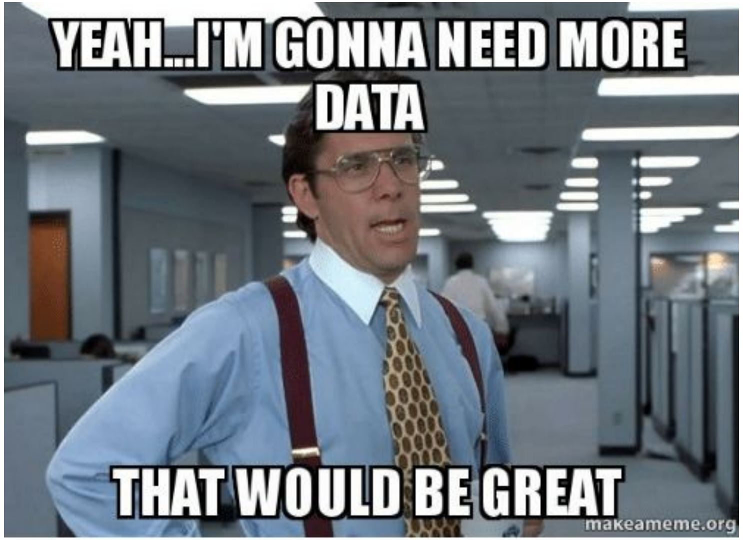
How much data do you need for deep learning?

A: A lot.



How much data do you need for deep learning?

A: A lot.

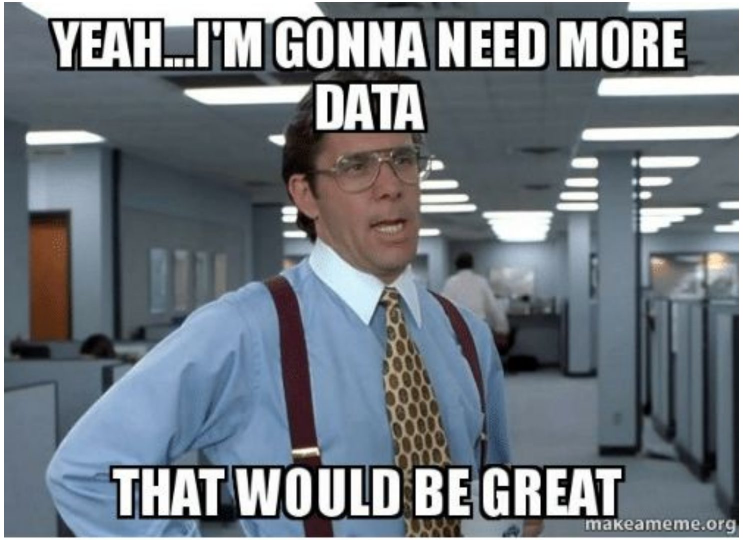


Premise of deep learning uses many parameters (e.g. millions) to fit complex functions -> if the dataset is too small, easiest solution that model ends up learning can be overfitting to memorizing the labels of the training examples

How much data do you need for deep learning?

A: A lot.

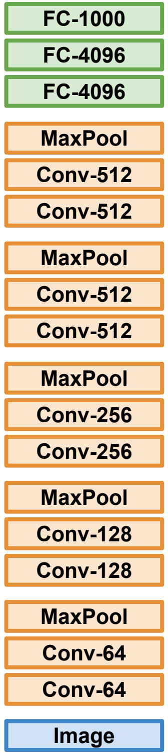
ImageNet dataset consists of 1M images: 1000 classes with 1000 images each



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

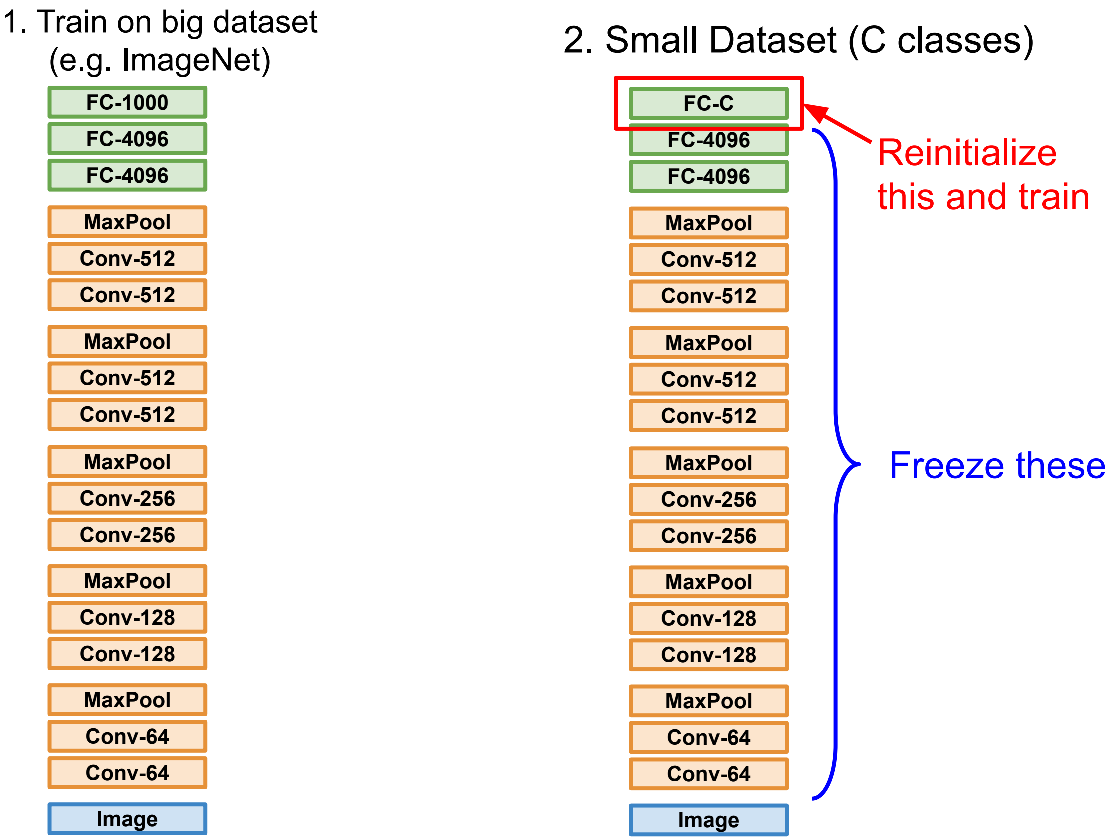
Transfer learning: amplifying training data

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



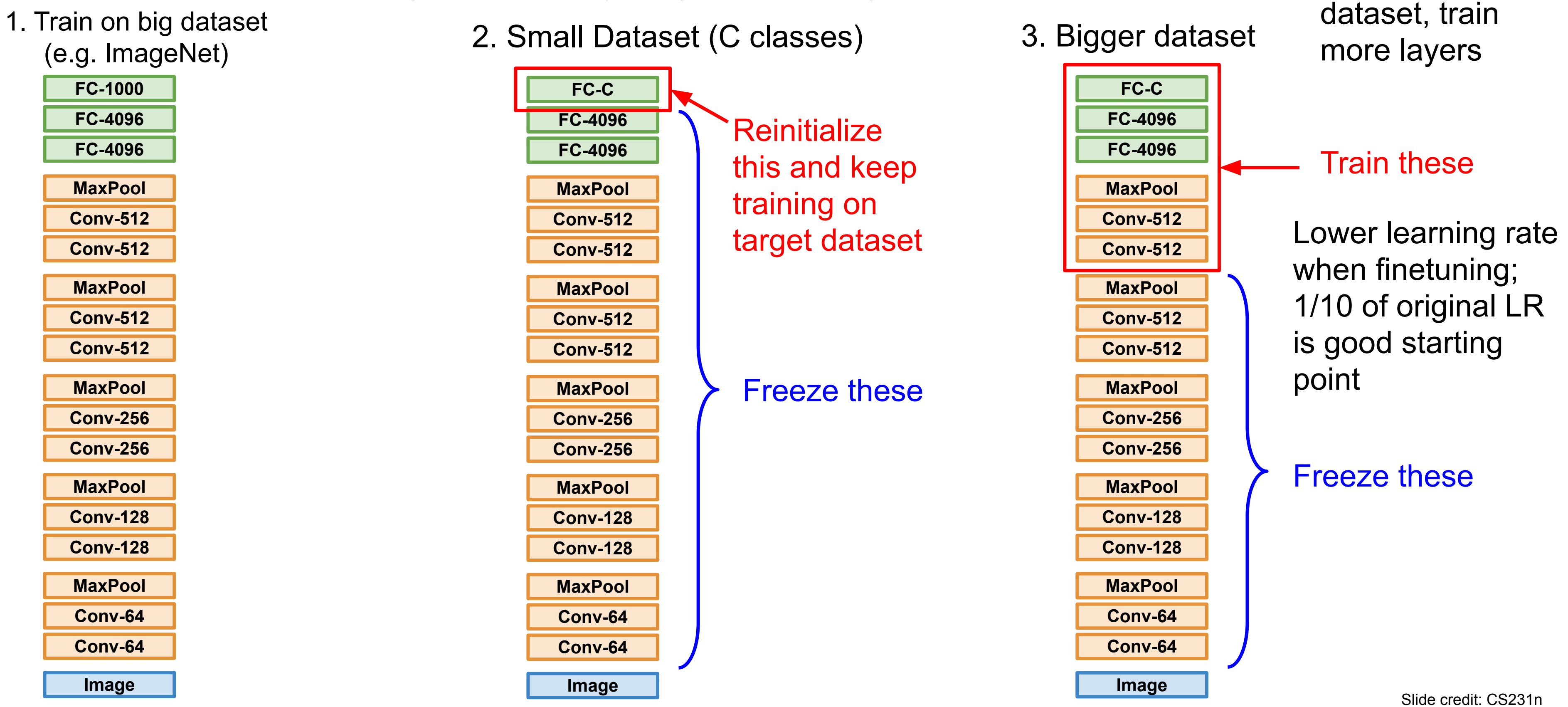
Slide credit: CS231n

Transfer learning: amplifying training data



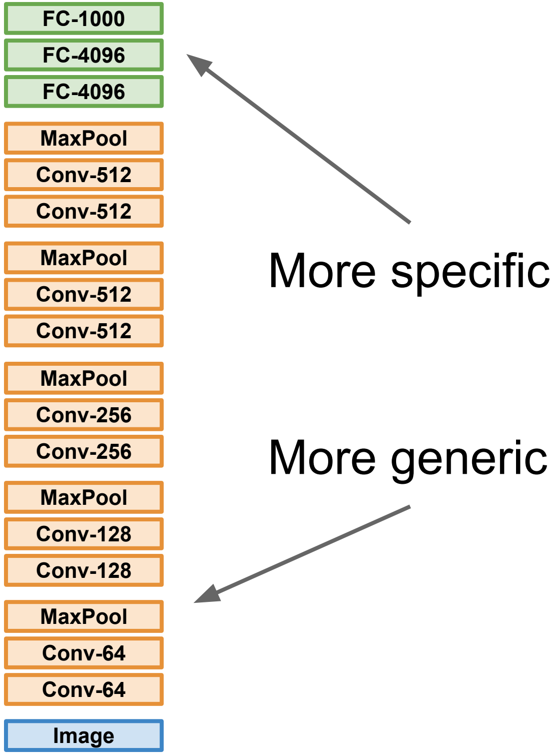
Slide credit: CS231n

Transfer learning: amplifying training data



Slide credit: CS231n

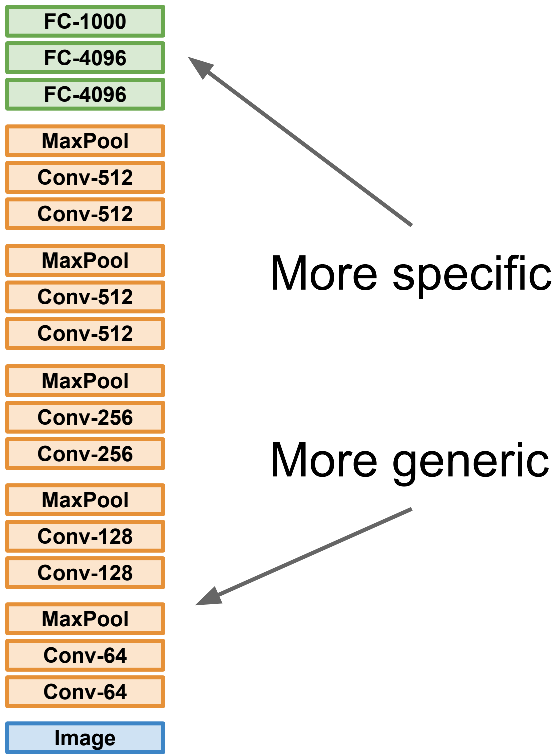
Transfer learning from a large dataset to your dataset...



	very similar dataset	very different dataset
very little data		
quite a lot of data		

Slide credit: CS231n

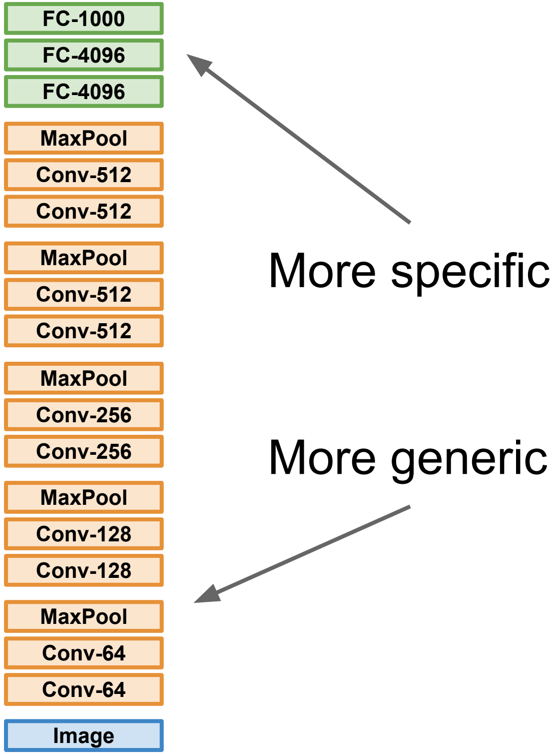
Transfer learning from a large dataset to your dataset...



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

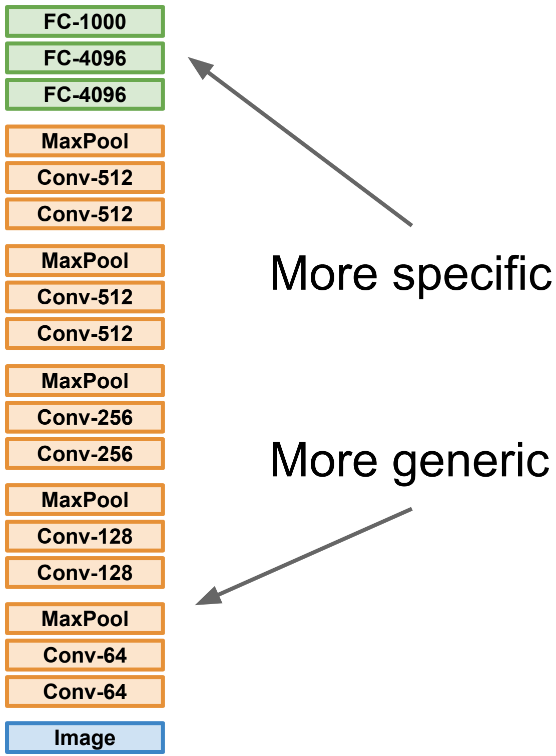
Slide credit: CS231n

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		

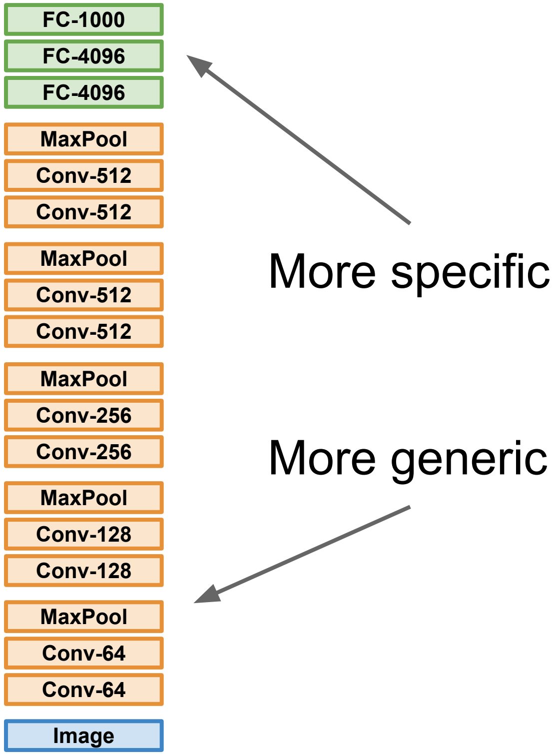
Slide credit: CS231n



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

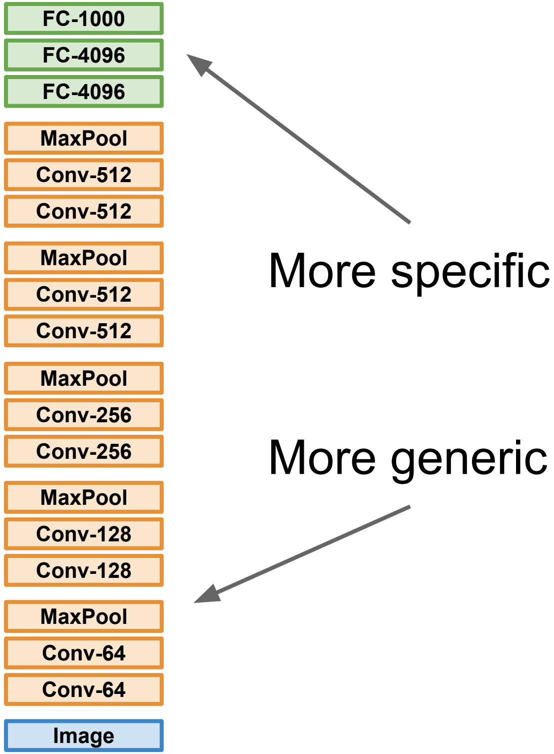


Transfer learning from a large dataset to your dataset...

	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 (take with grain of salt, really depends on problem), in addition to transfer learning:

- 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 (take with grain of salt, really depends on problem), in addition to transfer learning:

- 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 (take with grain of salt, really depends on problem), in addition to transfer learning:

- 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 (take with grain of salt, really depends on problem), in addition to transfer learning:

- 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 (take with grain of salt, really depends on problem), in addition to transfer learning:

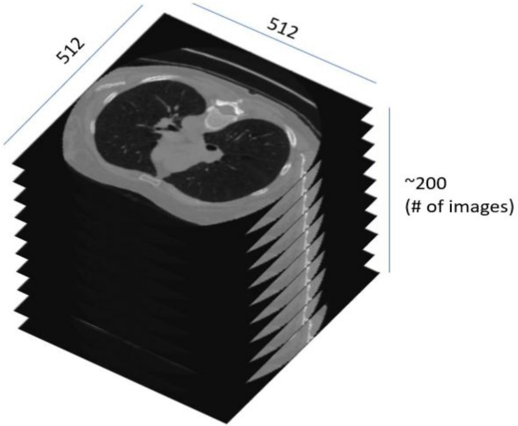
- Low dozens: generally too small to learn a meaningful model, using standard supervised deep learning
- High dozens to low hundreds: may see models with some predictive ability, unlikely to really wow or be "superhuman" though
- High hundreds to thousands: "happy regime" for deep learning

	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

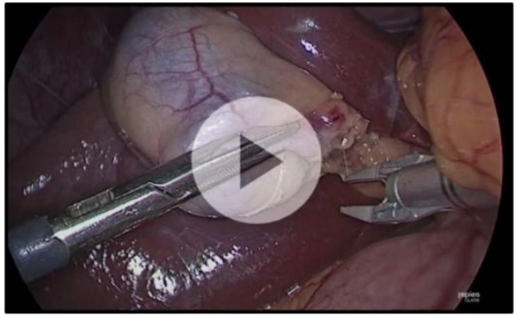
In general, deep learning is data hungry -- the more data the better

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

What counts as a data example?

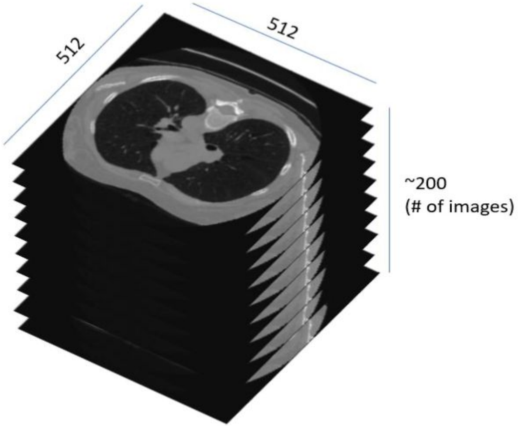


1 3D CT volume with 200 slices ≠ 200 data examples

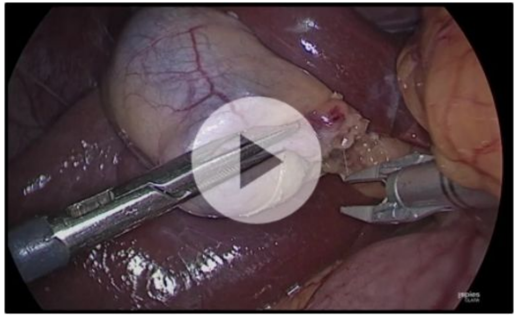


5 surgery videos with thousands of frames each ≠ thousands of data examples

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.

What if there are 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

What if there are 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

What if there are 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. More, but noisy data, often 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
- Weakly supervised learning
- Domain adaptation

Will talk more about these in later lectures...

Evaluating image classification models

Evaluation metrics

Confusion matrix

		Prediction	
		0	1
Ground Truth	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

		Prediction 0	Prediction 1
Ground Truth	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

		Prediction 0	Prediction 1
Ground Truth	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

		Prediction	
		0	1
Ground Truth	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

		Prediction 0	Prediction 1
Ground Truth	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

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

Accuracy: (TP + TN) / total

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

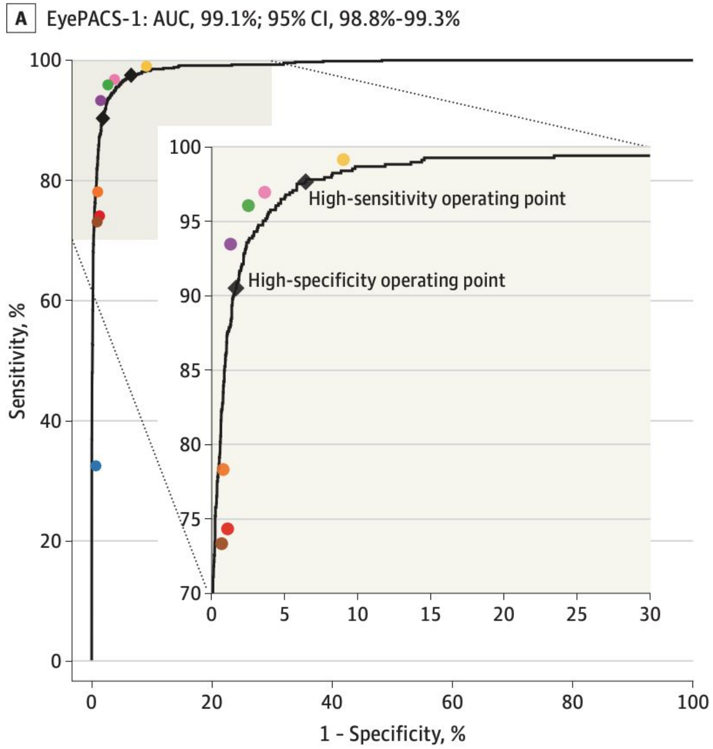
Specificity (true negative rate)**:**
TN / total negatives

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

Negative predictive value:
TN / total predicted negatives

Evaluation metrics

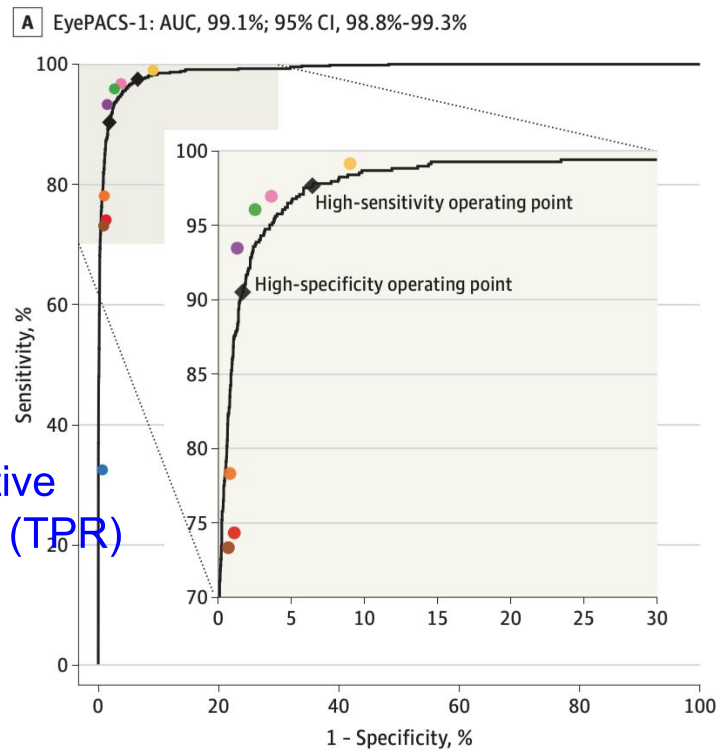
- **Receiver Operating Characteristic (ROC) curve**:
  - Plots sensitivity and specificity (specifically, 1 - specificity) as prediction threshold is varied
  - Gives trade-off between sensitivity and specificity
  - Also report summary statistic AUC (area under the curve)



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)

True Positive Rate (TPR)



False Positive Rate (FPR)

Evaluation metrics

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

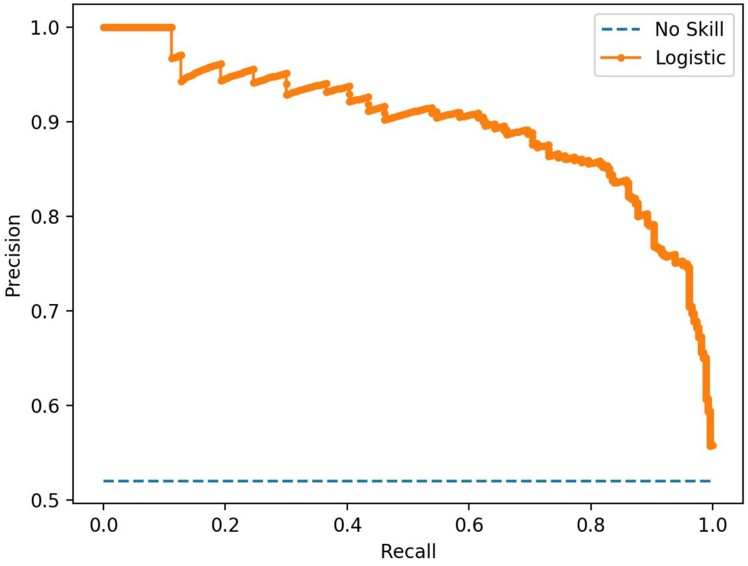


Figure credit: https://3qeqpr26caki16dnhd19sv6by6v-wpengine.netdna-ssl.com/wp-content/uploads/2018/08/Precision-Recall-Plot-for-a-No-Skill-Classifier-and-a-Logistic-Regression-Model4.png

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

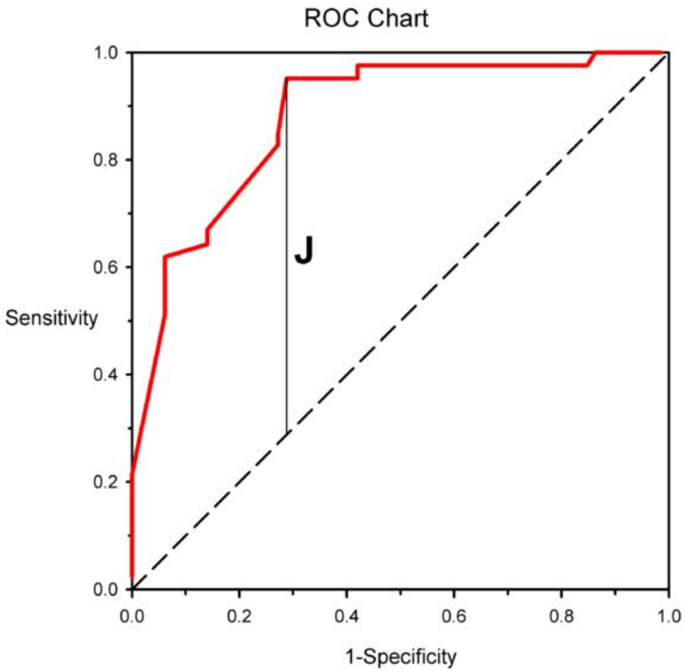


Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

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

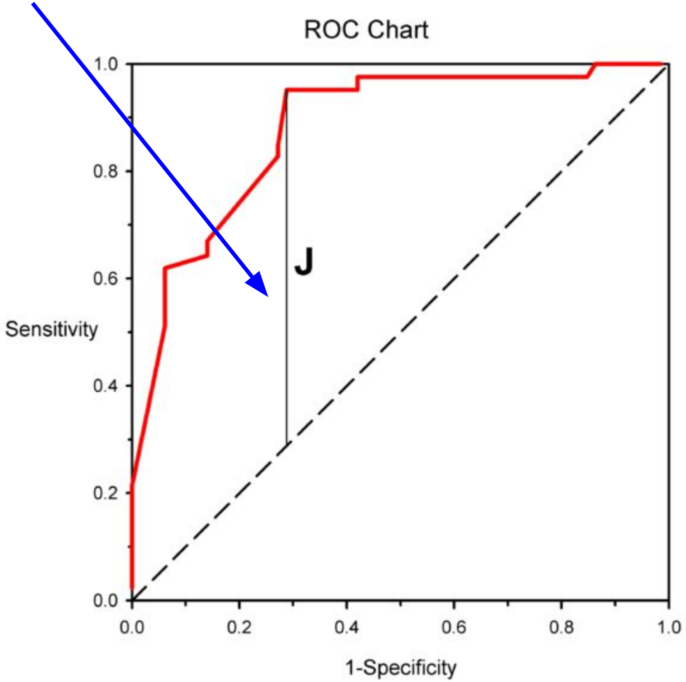


Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

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

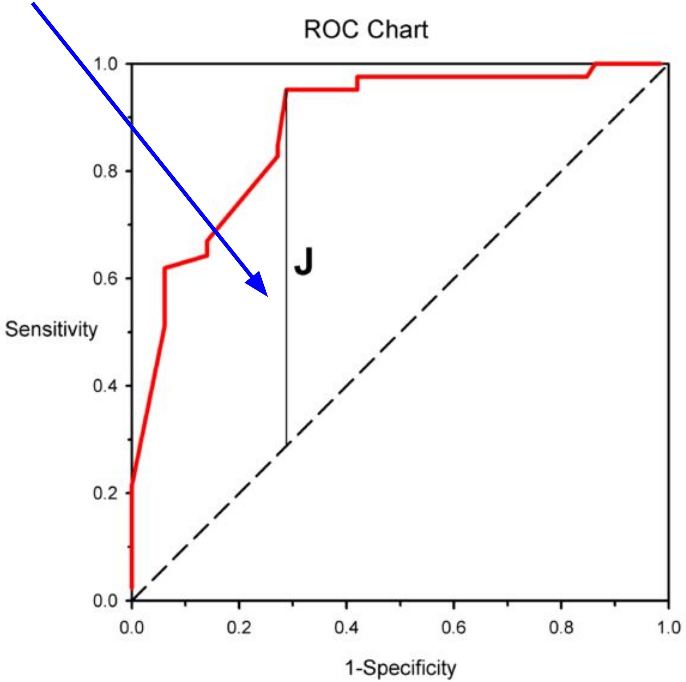


Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

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

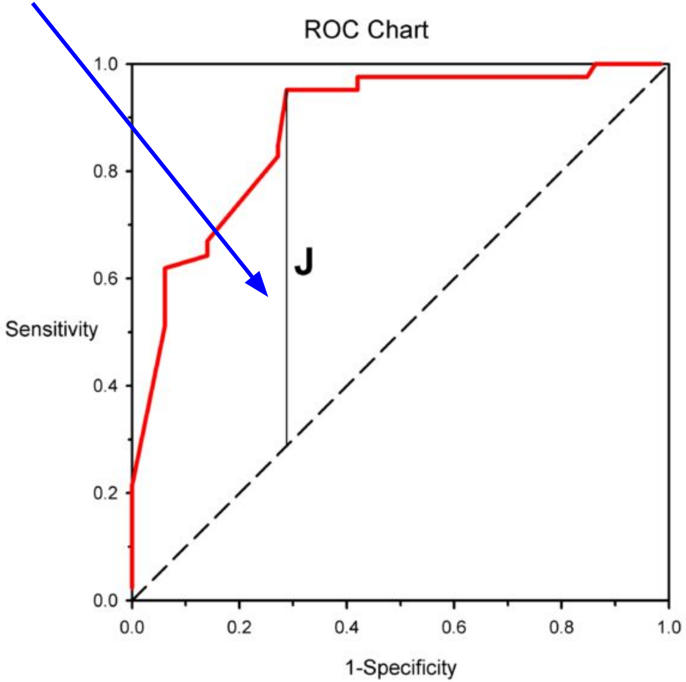


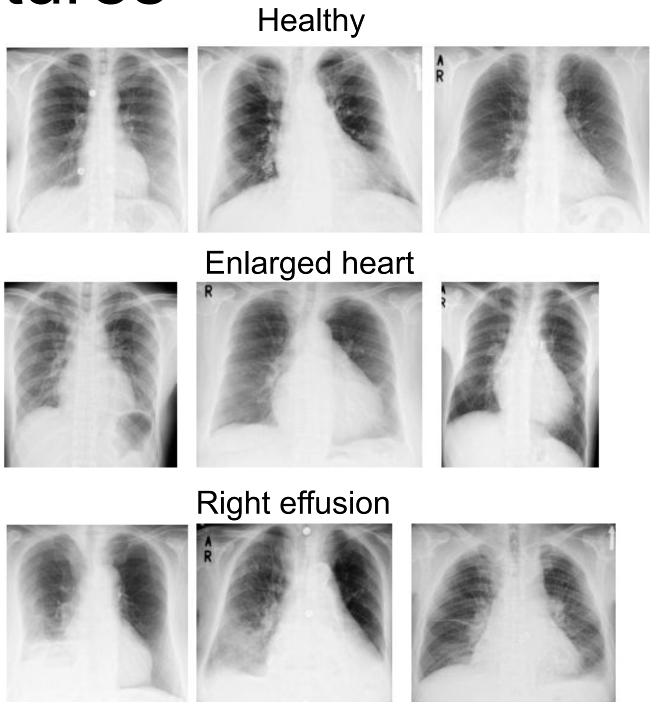
Figure credit: https://en.wikipedia.org/wiki/File:ROC_Curve_Youden_J.png

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

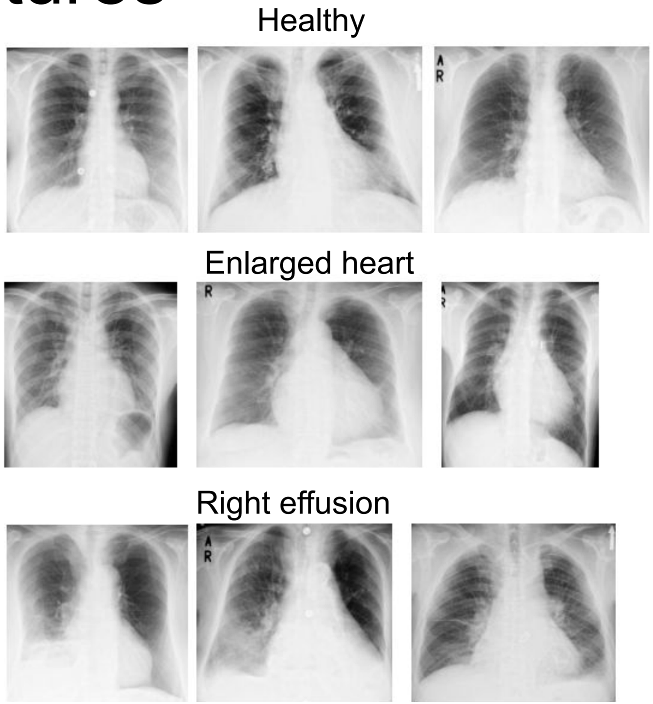


Bar et al. Deep learning with non-medical training used for chest pathology identification. SPIE, 2015.

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

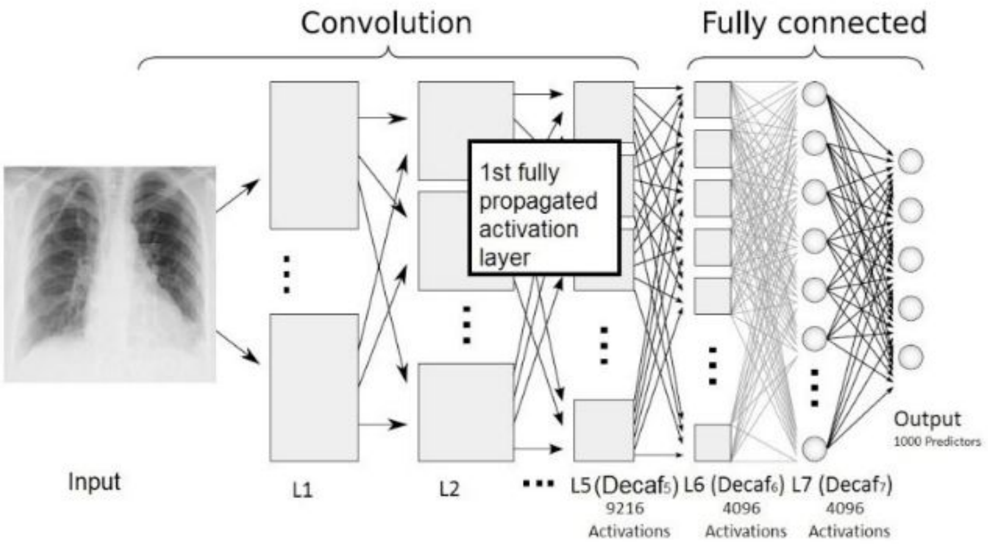


Q: How might we approach this problem?

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.

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**

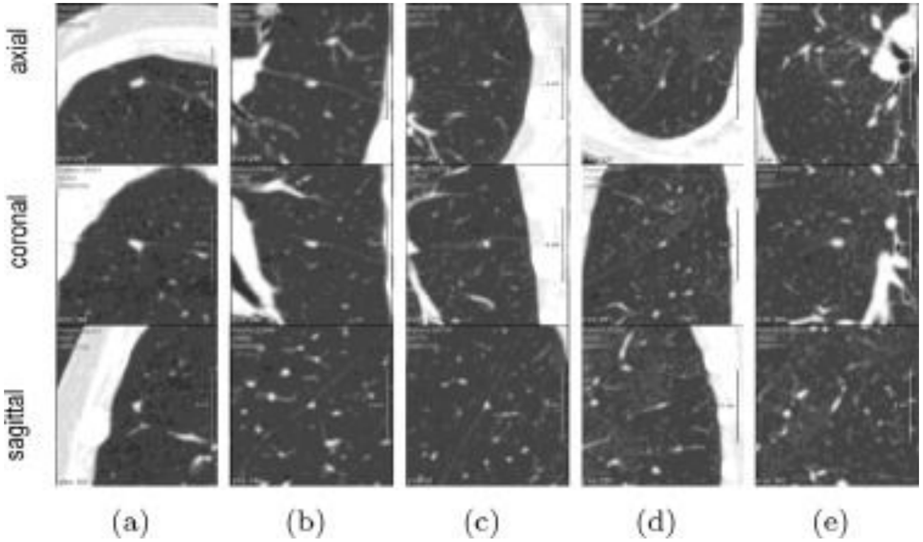
Table 3. Enlarged Heart Condition.

	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

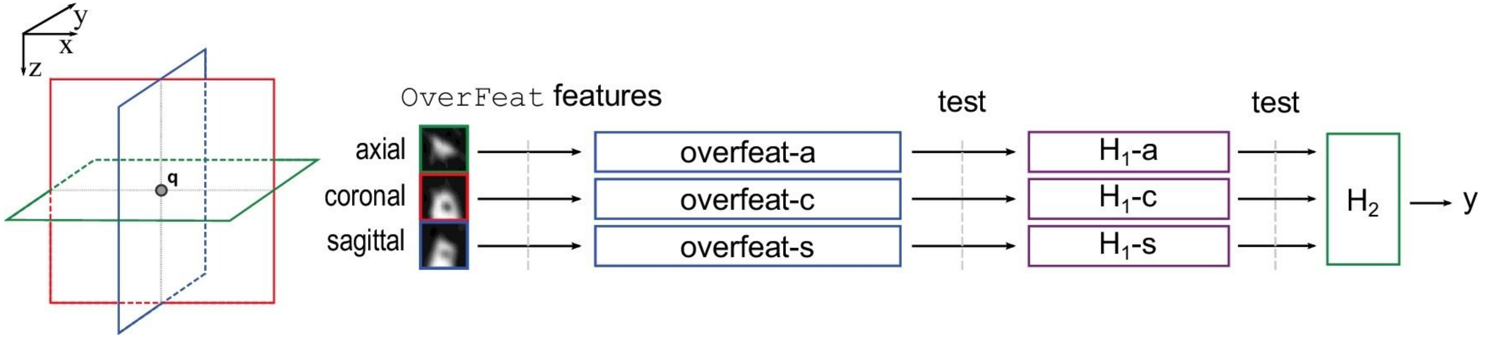
- Task: classification of lung nodules in **3D CT scans** as peri-fissural nodules (PFN, likely to be benign) or not
- Dataset: 568 nodules from 1729 scans at a single institution. (65 typical PFNs, 19 atypical PFNs, 484 non-PFNs).
- Data pre-processing: prescaling from CT hounsfield units (HU) into [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

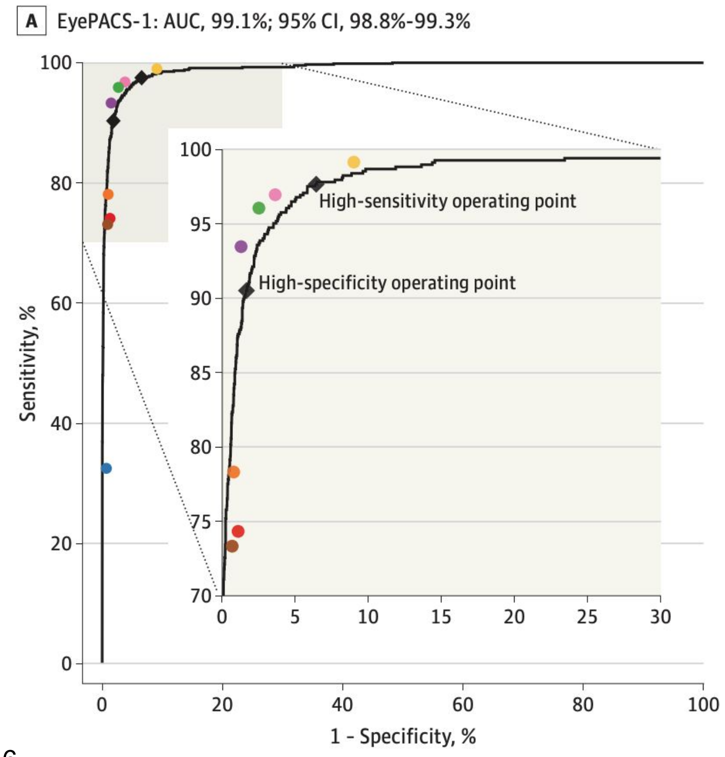
- **Dataset**:
  - 128,175 images, each graded by 3-7 ophthalmologists.
  - 54 total graders, each paid to grade between 20 to 62508 images.
- **Data preprocessing**:
  - Circular mask of each image was detected and rescaled to be 299 pixels wide
- **Model**:
  - Inception-v3 CNN, with ImageNet pre-training
  - 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.

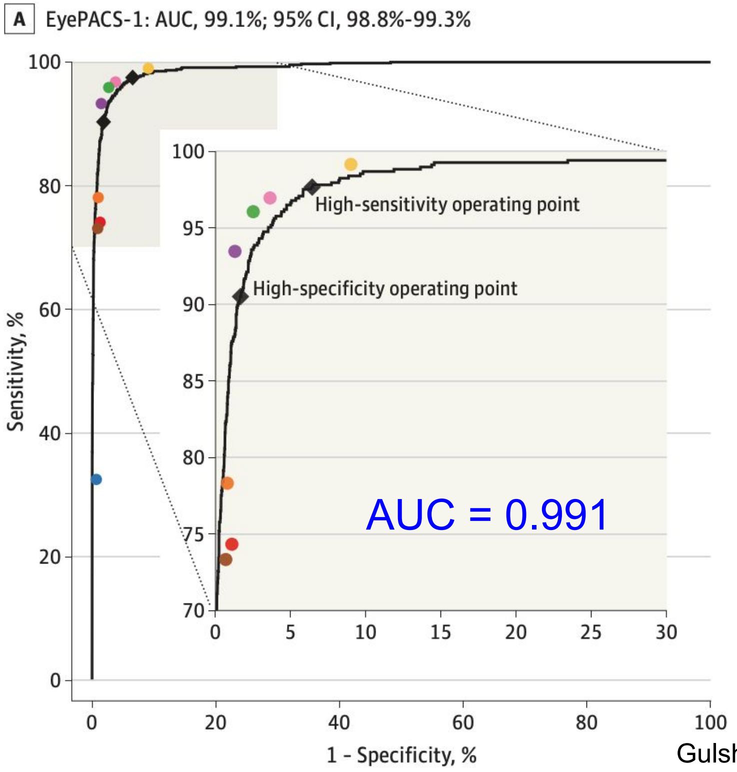
Gulshan et al. 2016

- **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

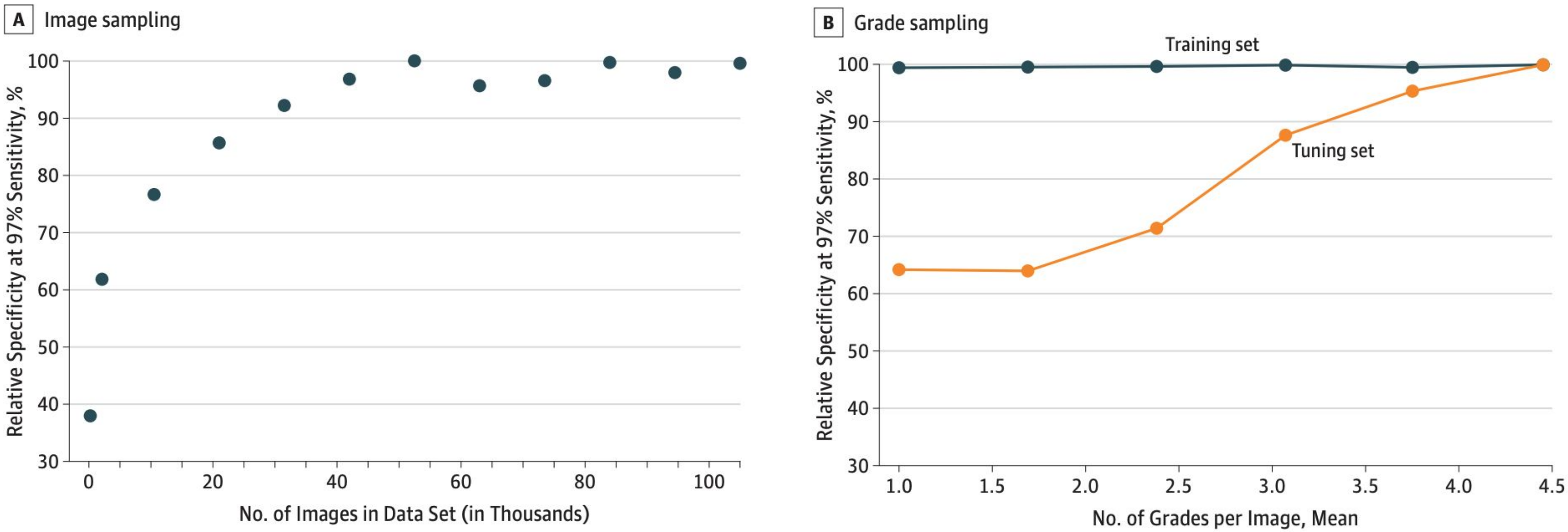


Looked at different operating points

- 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.

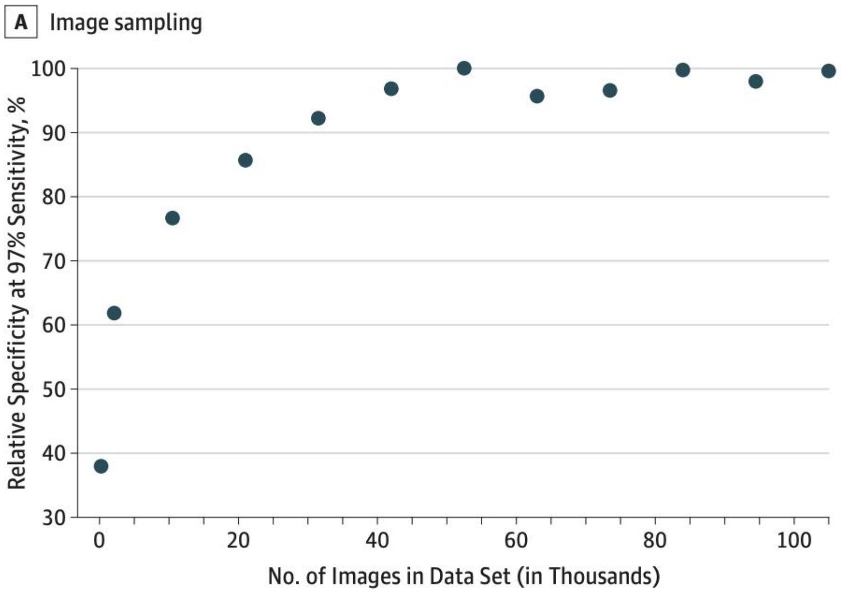
Gulshan et al. 2016

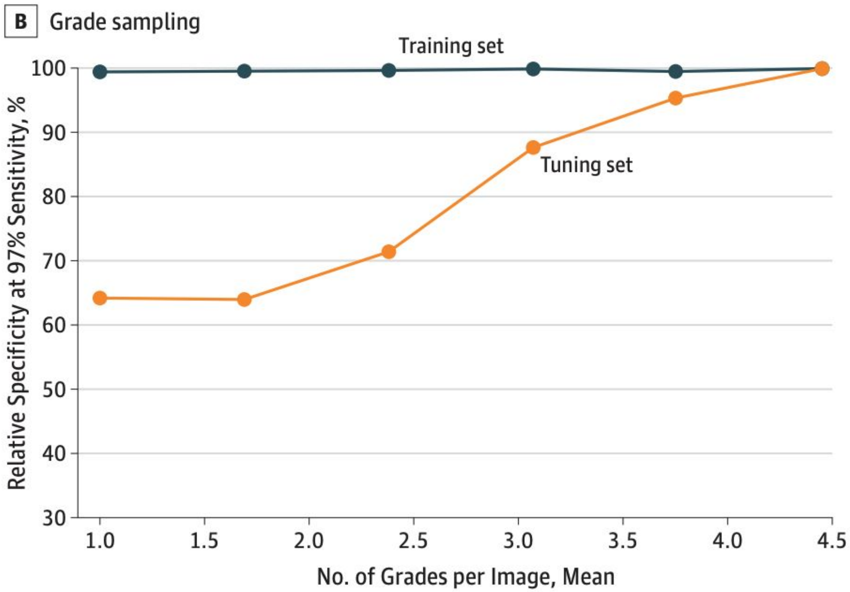


Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

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?

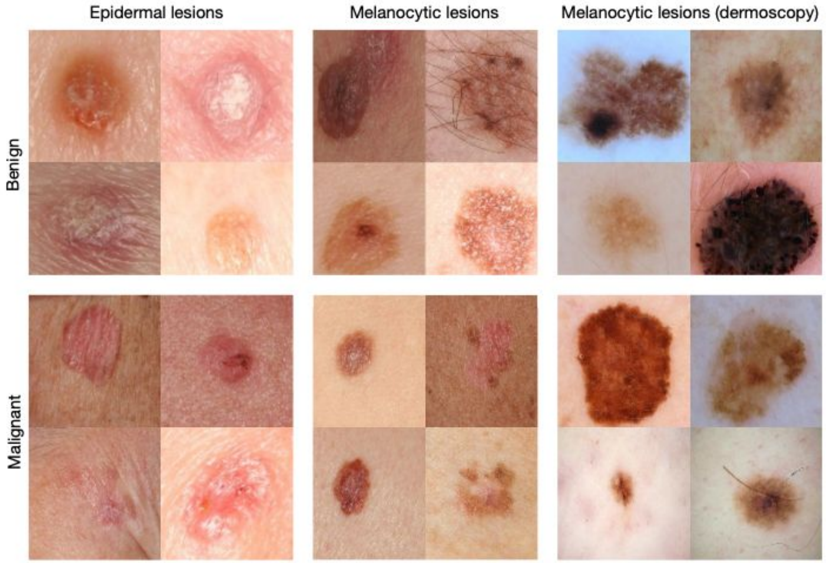




Gulshan, et al. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA, 2016.

Esteva et al. 2017

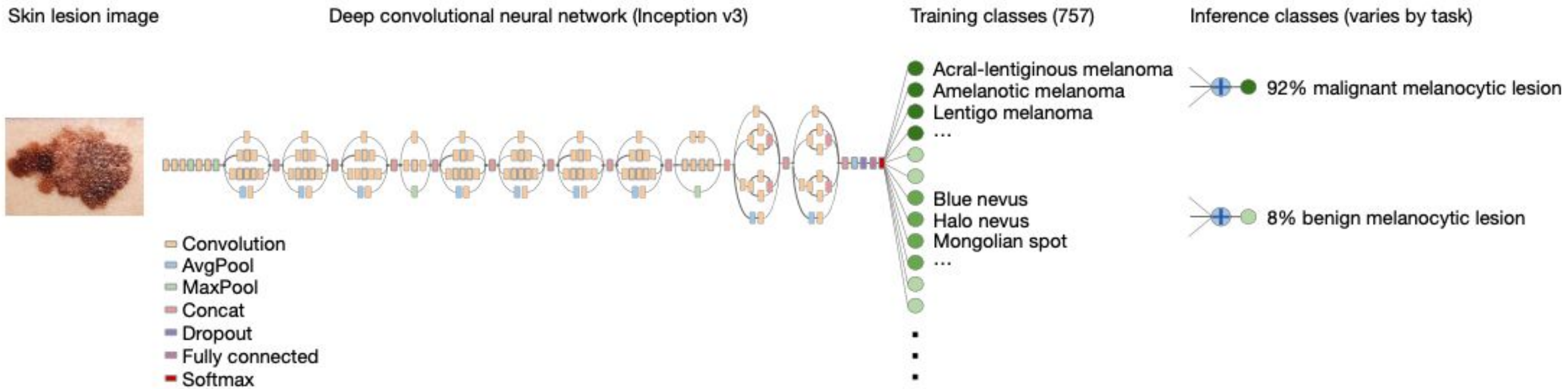
- Two binary classification tasks: malignant vs. benign lesions of epidermal or melanocytic origin
- Inception-v3 (GoogLeNet) CNN with ImageNet pre-training
- Fine-tuned on dataset of 129,450 lesions (from several sources) comprising 2,032 diseases
- Evaluated model vs. 21 or more dermatologists in various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



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

Esteva et al. 2017

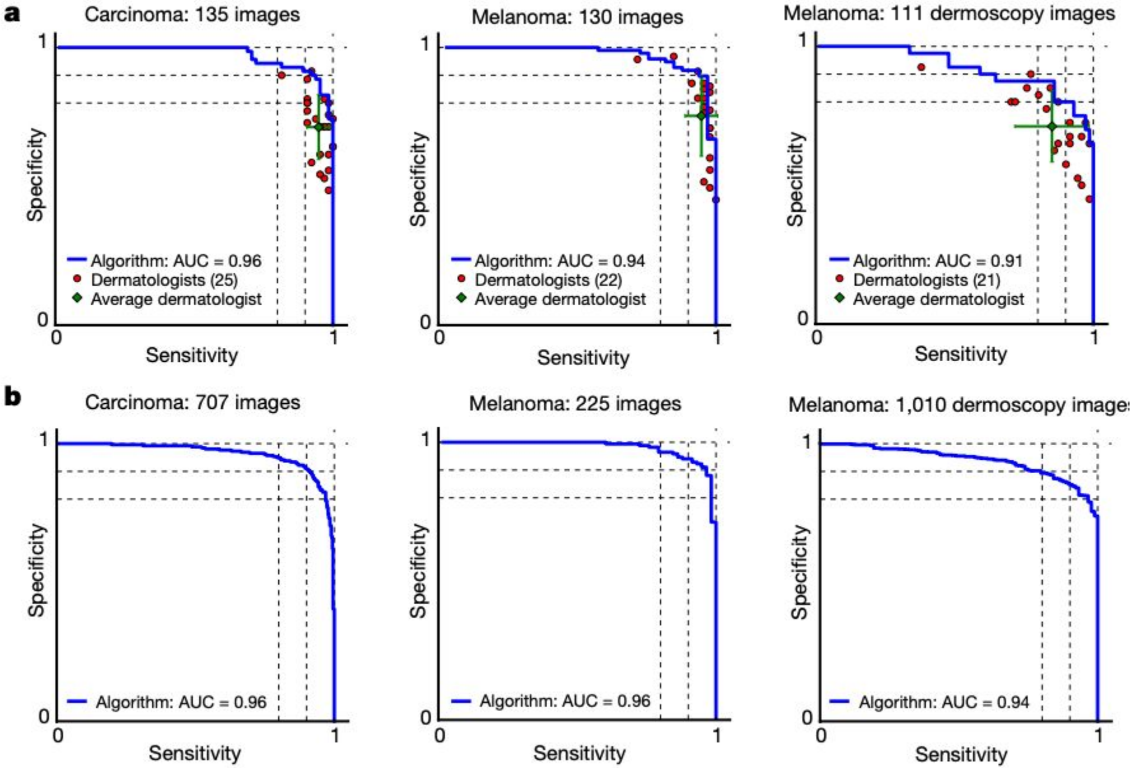
- Train on finer-grained classification (757 classes) but perform binary classification at inference time by summing probabilities of fine-grained sub-classes
- The stronger fine-grained supervision during the training stage improves inference performance!



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.

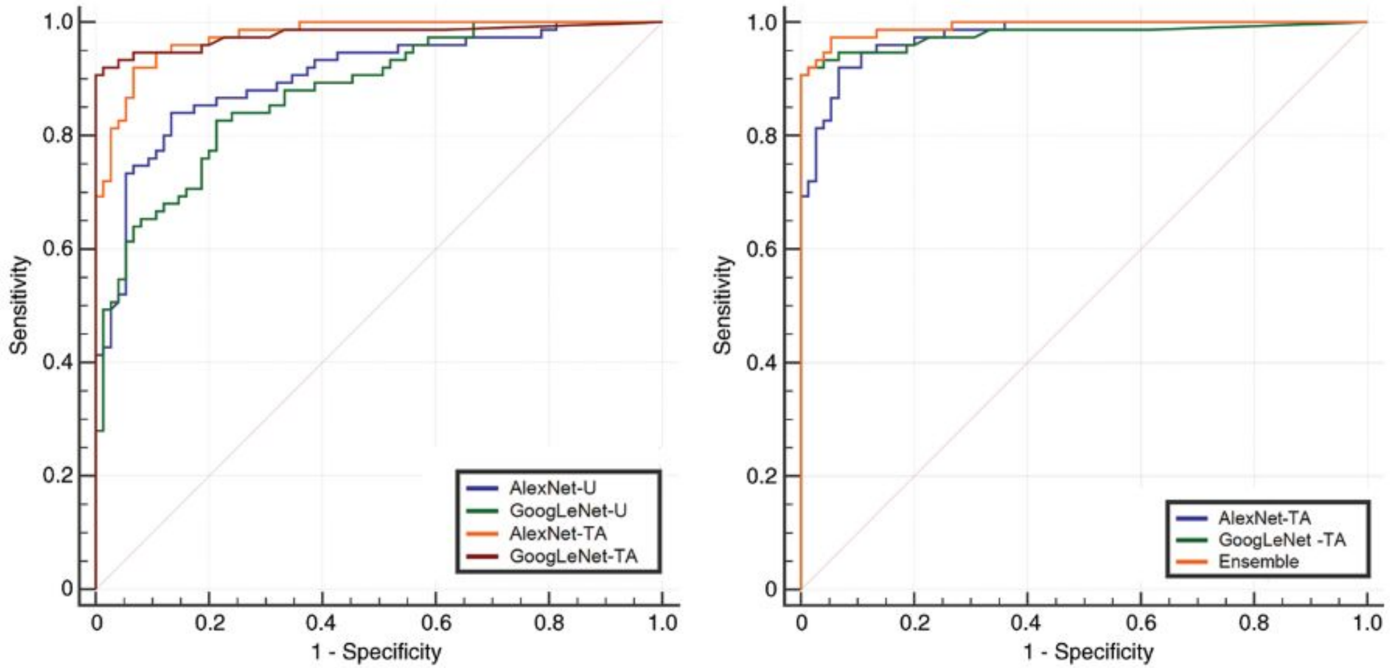
* Additional augmentation of 90, 180, 270 rotations, and Contrast Limited Adaptive Histogram Equalization processing.

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.

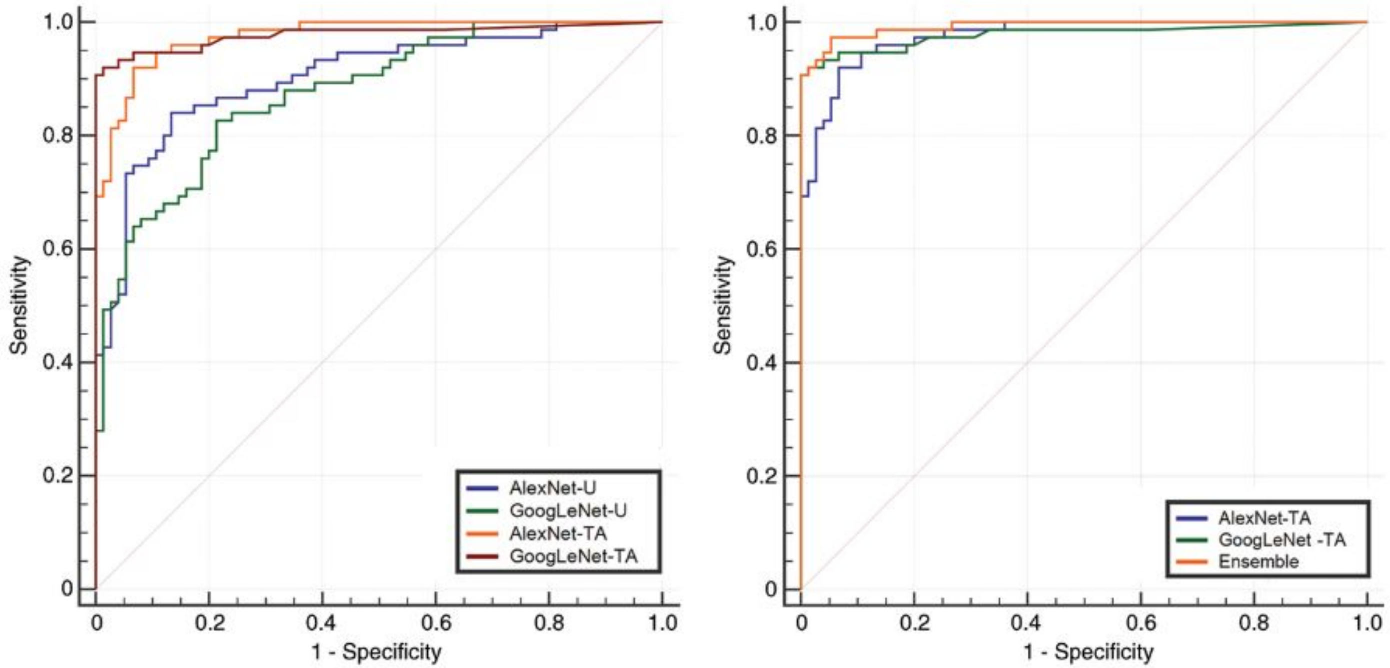
Lakhani and Sundaram 2017



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

Lakhani and Sundaram 2017

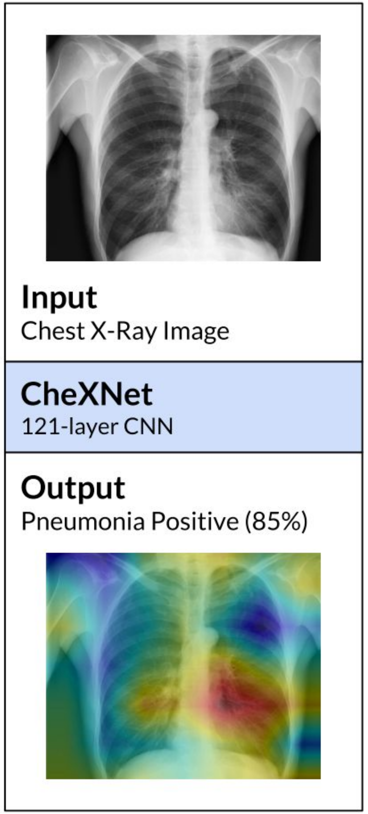
Performed further analysis at optimal threshold determined by the Youden Index.



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

Rajpurkar et al. 2017

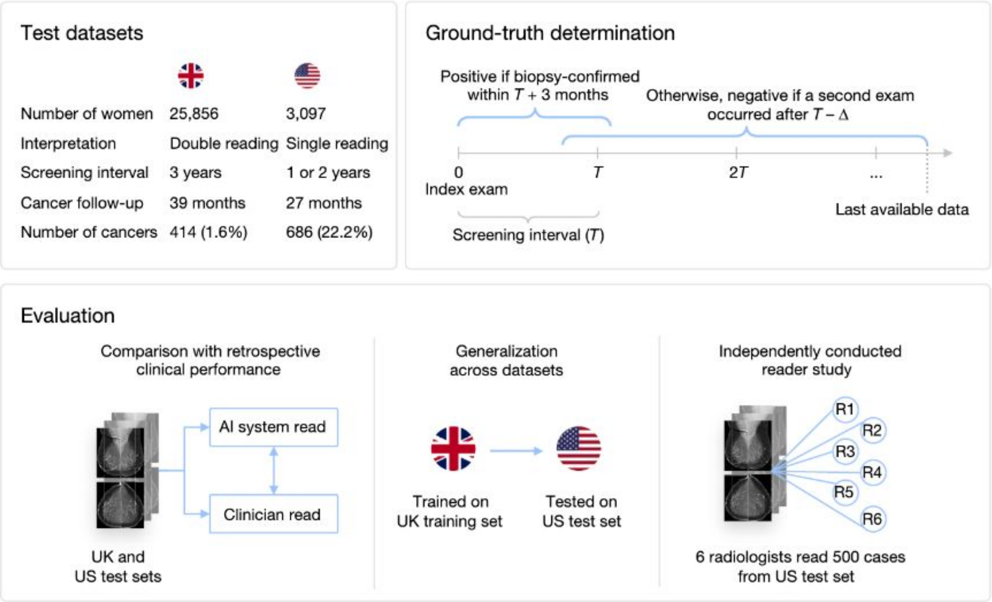
- Binary classification of pneumonia presence in chest X-rays
- Used ChestX-ray14 dataset with over 100,000 frontal X-ray images with 14 diseases
- 121-layer DenseNet CNN
- Compared algorithm performance with 4 radiologists
- Also applied algorithm to other diseases to surpass previous state-of-the-art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



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

Summary

Today we saw:

- Deep learning models for image classification
- Data considerations for image classification models
- Evaluating image classification models
- Case studies

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