

Lecture 10

Convolutional and

Deep Neural Networks

EE-UY 4563/EL-GY 9123: INTRODUCTION TO MACHINE LEARNING

PROF. SUNDEEP RANGAN WITH MODIFICATION BY YAO WANG

Outline

→ Motivation: ImageNet Large-Scale Visual Recognition Challenge (ILSVR)

- ❑ Deep Networks and Feature Hierarchies
- ❑ 2D convolutions
- ❑ Convolutional neural networks
- ❑ Creating and visualizing convolutional layers in Keras
- ❑ Backpropagation training in CNNs
- ❑ Exploring VGG16: A state-of-the-art deep network

Large-Scale Image Classification

- ❑ Pre-2009, many image recognition systems worked on relatively small datasets

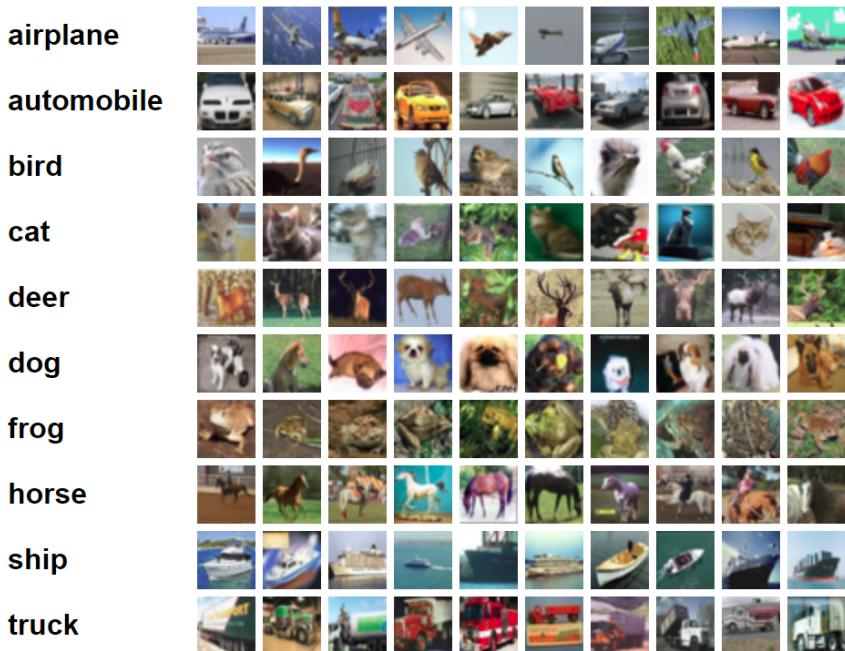
- MNIST: 10 digits
- CIFAR 10 (right)
- CIFAR 100
- ...

- ❑ Small number of classes (10-100)

- ❑ Low resolution (eg. 32 x 32 x 3)

- ❑ Performance saturated
 - Difficult to make significant advancements

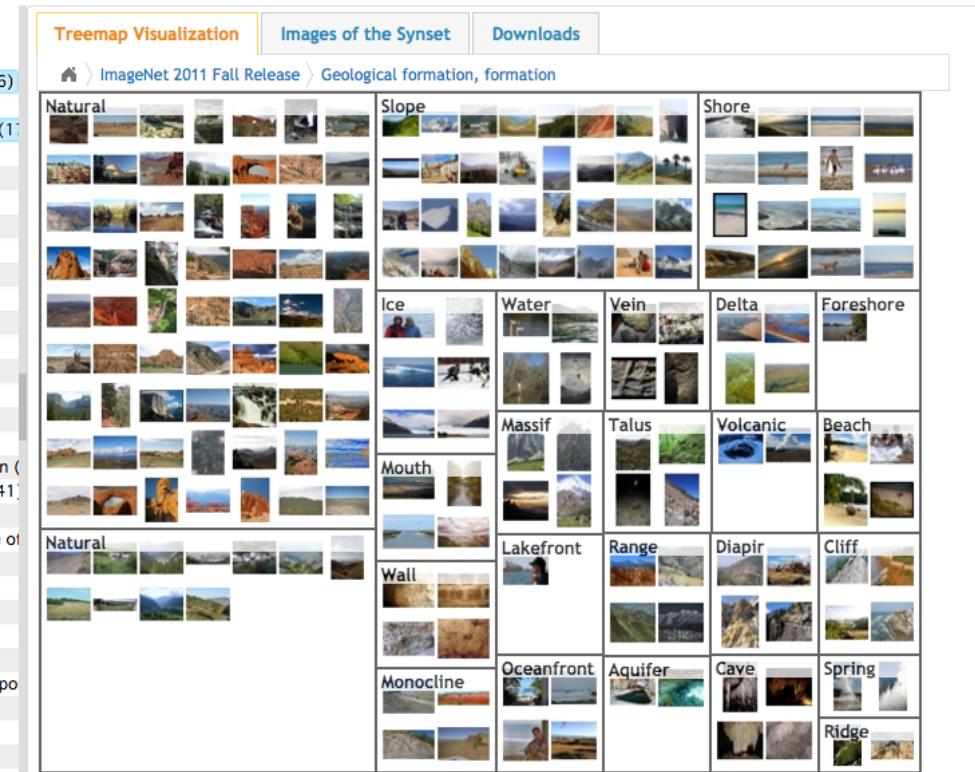
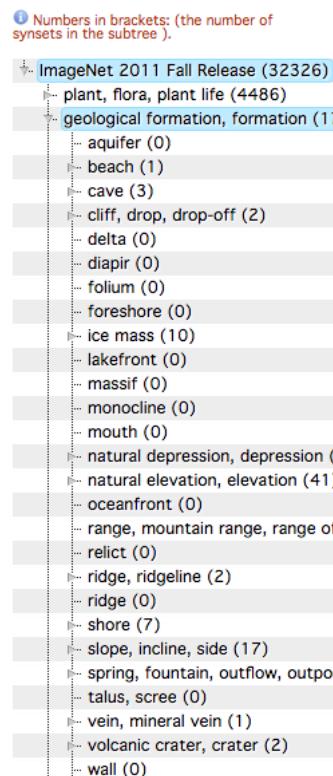
<https://www.cs.toronto.edu/~kriz/cifar.html>



ImageNet (2009)

- ❑ Better algorithms need better data
- ❑ Build a large-scale image dataset
- ❑ 2009 CVPR paper:
 - 3.2 million images
 - Annotated by mechanical turk
 - Much larger scale than any previous
- ❑ Hierarchical categories

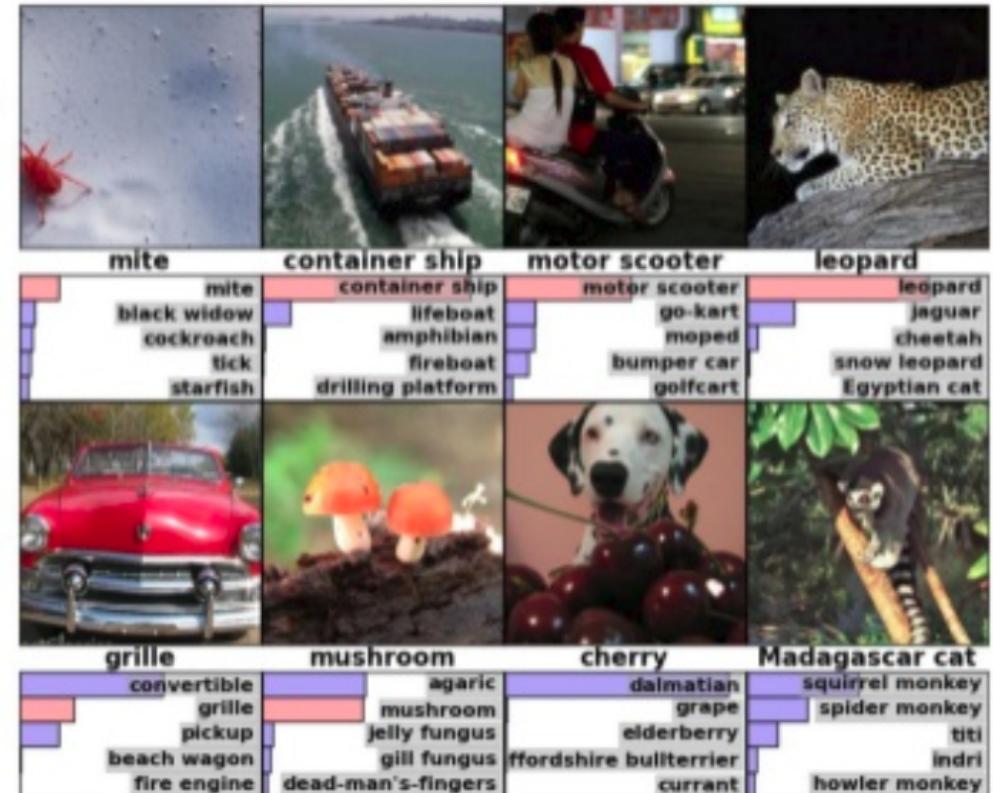
Geological formation, formation
(geology) the geological features of the earth



Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on* (pp. 248-255). IEEE.

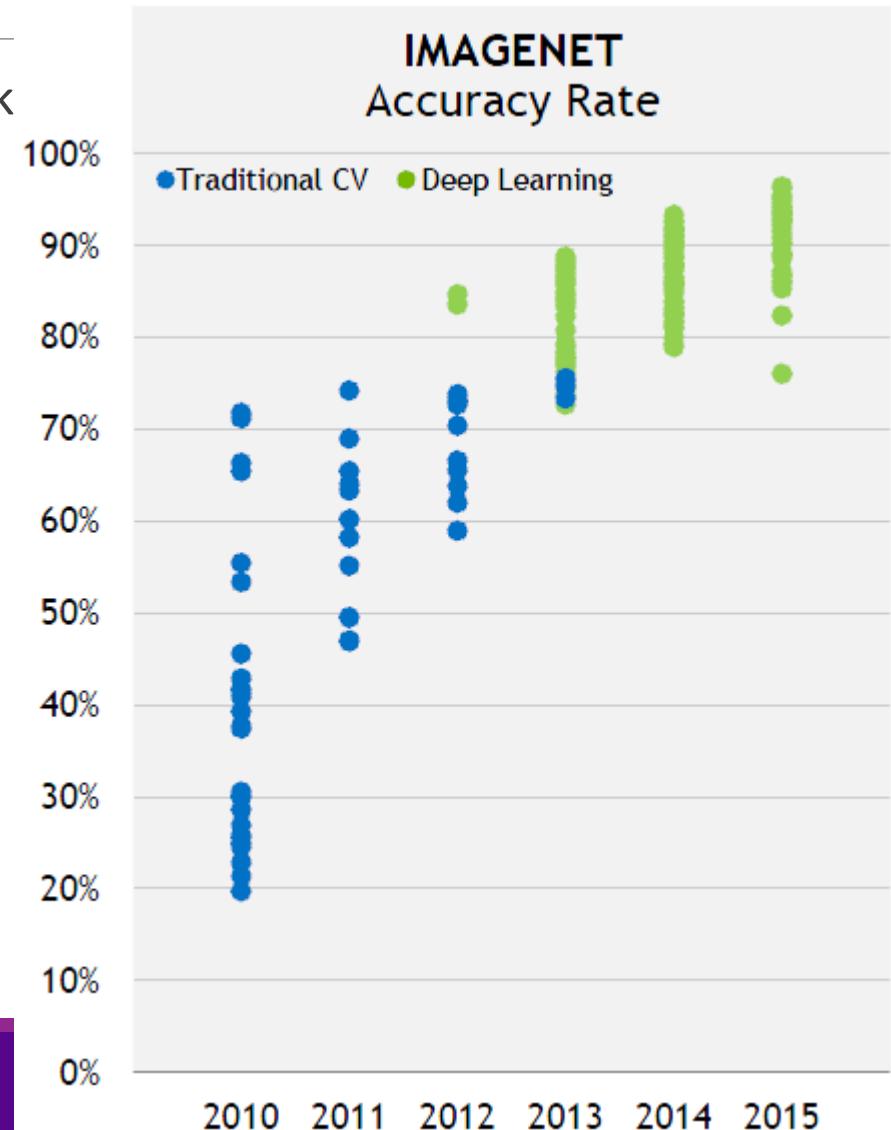
ILSVRC

- ❑ ImageNet Large-Scale Visual Recognition Challenge
- ❑ First year of competition in 2010
- ❑ Many developers tried their algorithms
- ❑ Many challenges:
 - Objects in variety of positions, lighting
 - Occlusions
 - Fine-grained categories
(e.g. African elephants vs. Indian elephants)
 - ...



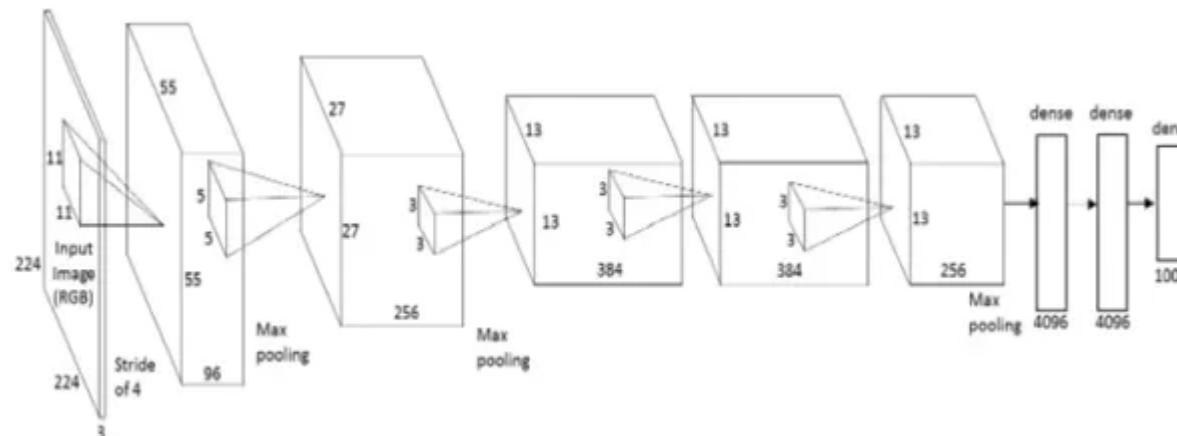
Deep Networks Enter 2012

- ❑ 2012: Stunning breakthrough by the first deep network
- ❑ “AlexNet” from U Toronto
- ❑ Easily won ILSVRC competition
 - Top-5 error rate: 15.3%, second place: 25.6%
- ❑ Soon, all competitive methods are deep networks



Alex Net

- ❑ Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, University of Toronto, 2012
- ❑ Key idea: Build a very deep neural network
- ❑ 60 million parameters, 650000 neurons
- ❑ 5 conv layers + 3 FC layers
- ❑ Final is 1000-way softmax



Outline

❑ Motivation: ImageNet Large-Scale Visual Recognition Challenge (ILSVR)

→ Deep Networks and Feature Hierarchies

❑ 2D convolutions

❑ Convolutional neural networks

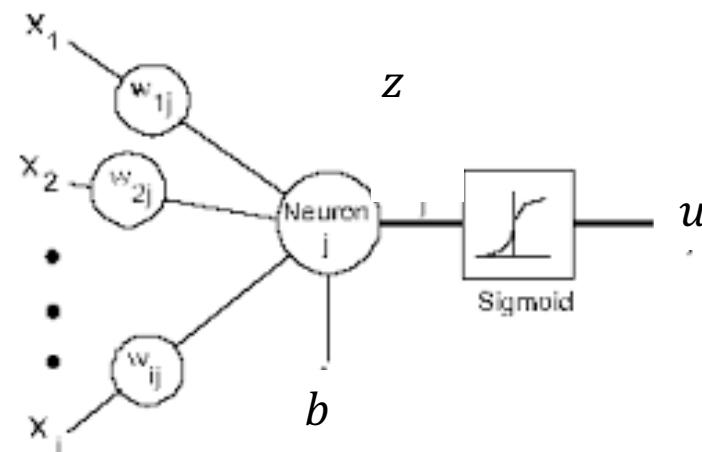
❑ Backpropagation training in CNNs

❑ Exploring VGG16: A state-of-the-art deep network

Neural Network Units

- ❑ Why do deep networks work?
- ❑ Recap: Neural networks composed of basic units:

- z = one linear output (in a hidden or output layer)
- $x = (x_1, \dots, x_N)$ = input to the layer
- w = weight vector
- b = bias

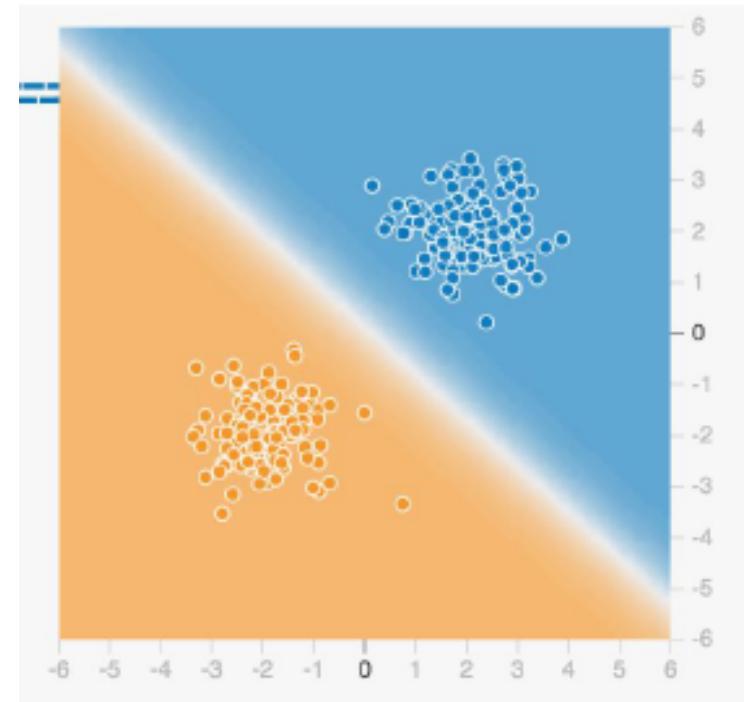


Linear Feature

- ❑ Suppose activation is a hard threshold:

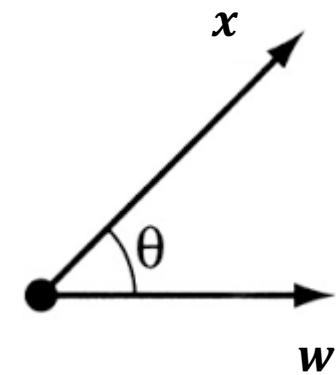
$$g_{act}(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

- ❑ Then, hidden unit divides input space into two **half spaces**
- ❑ Linearly separated
- ❑ Each unit can learn a **linear feature** $z_m = \mathbf{w}_m^T \mathbf{x} + b_m$
 - Classifies its input by being in a half space
- ❑ Shape of the feature defined by weight \mathbf{w}_m

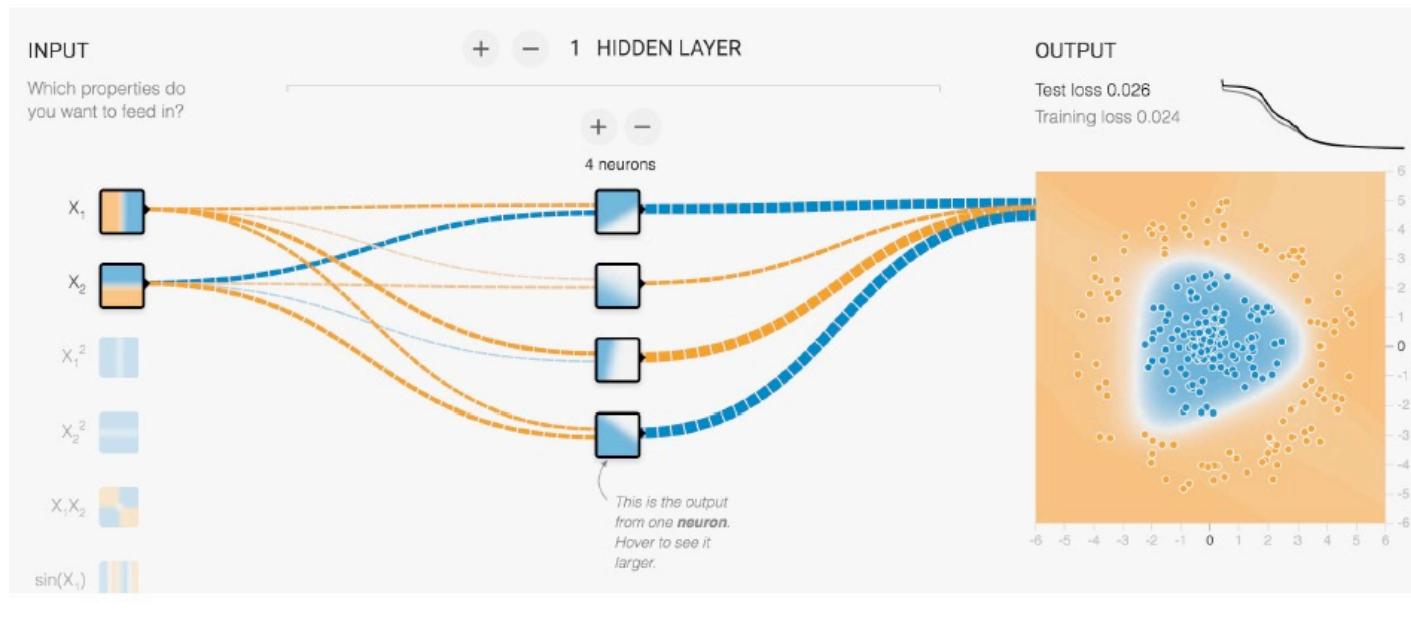


Tuning to a Feature

- ❑ Each unit will output a large value when z is large.
 - ❑ When is z large?
 - ❑ Recall: $z = \mathbf{w}^T \mathbf{x} + b = \|\mathbf{w}\| \|\mathbf{x}\| \cos \theta + b$
 - ❑ Conclusion: z is maximized when $\theta = 0 \Rightarrow \mathbf{x} = \alpha \mathbf{w}$
 - \mathbf{x} should be aligned with \mathbf{w}
 - ❑ Say that unit is **tuned** to **feature** \mathbf{w}
-
- ❑ The weight for a unit describes one desirable “pattern” in the previous layer output



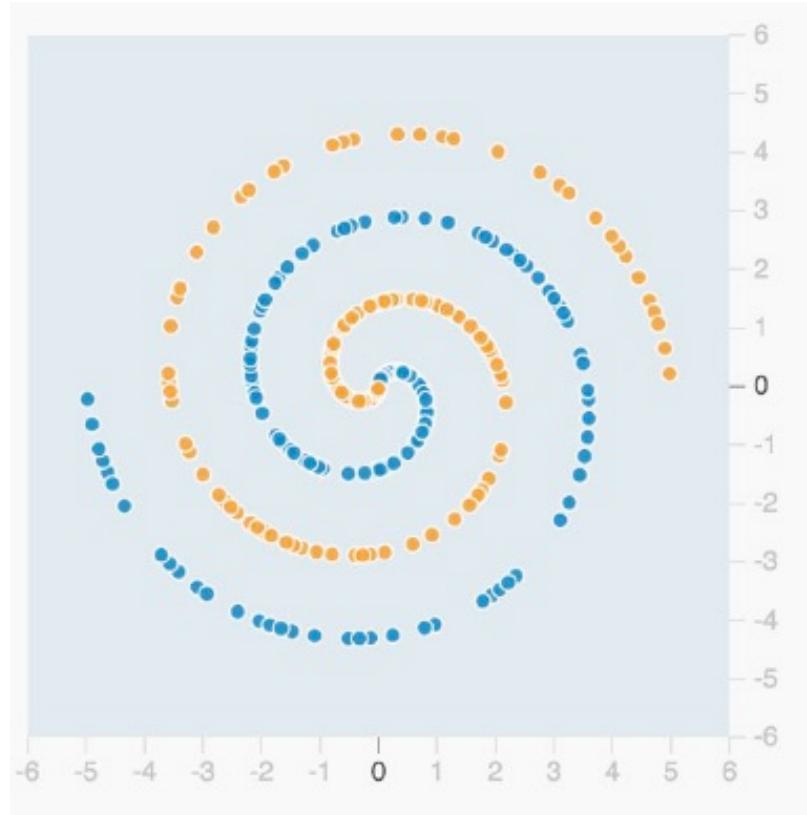
Classifying a Nonlinear Region



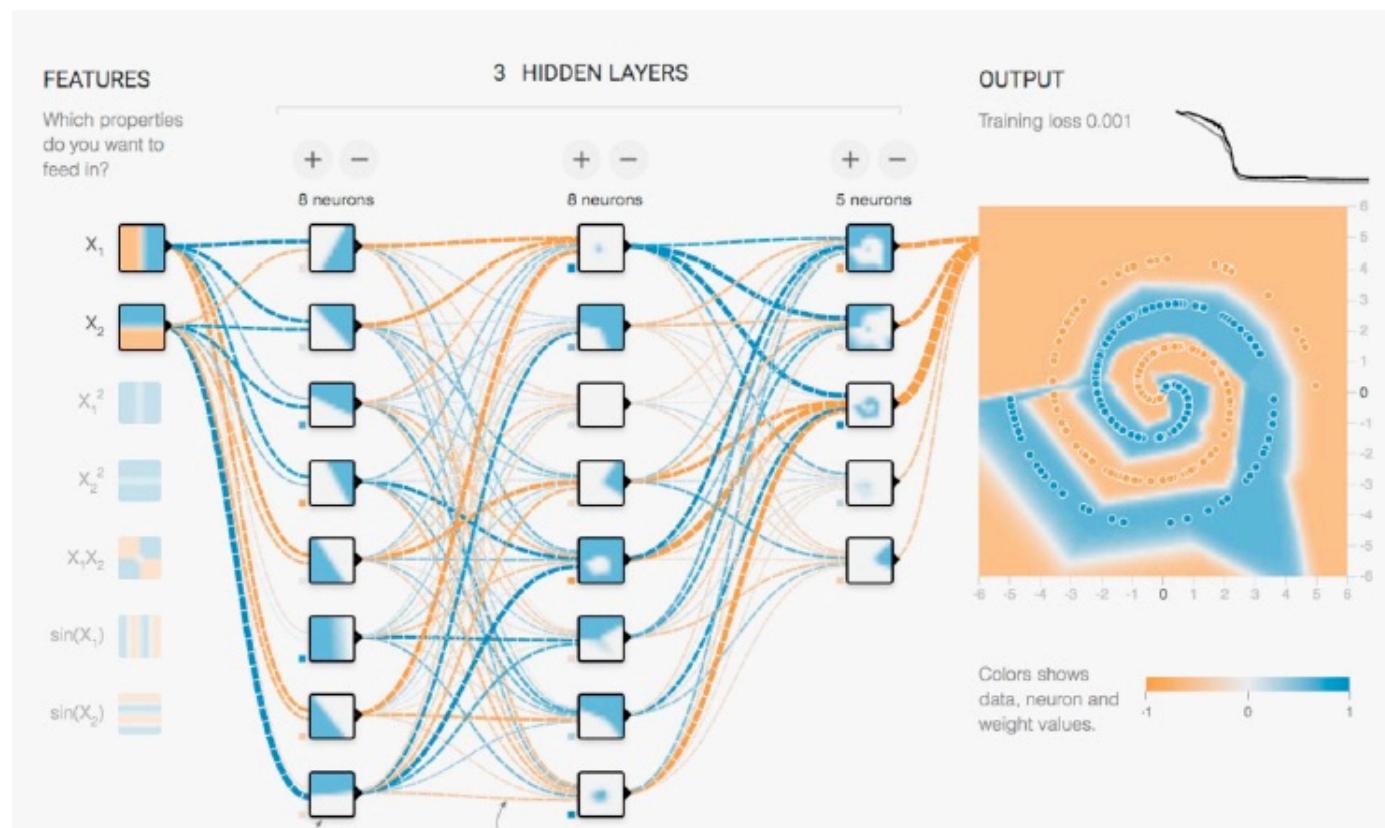
- ❑ Nonlinear regions
- ❑ Build from linear regions
- ❑ Picture to left:
 - Output of Tensorboard
 - Tool in TensorFlow
 - Provided for visualizing neural nets

From Kaz Sato, "Google Cloud Platform Empowers TensorFlow and Machine Learning"

What about a More Complicated Region?



Use Multiple Layers

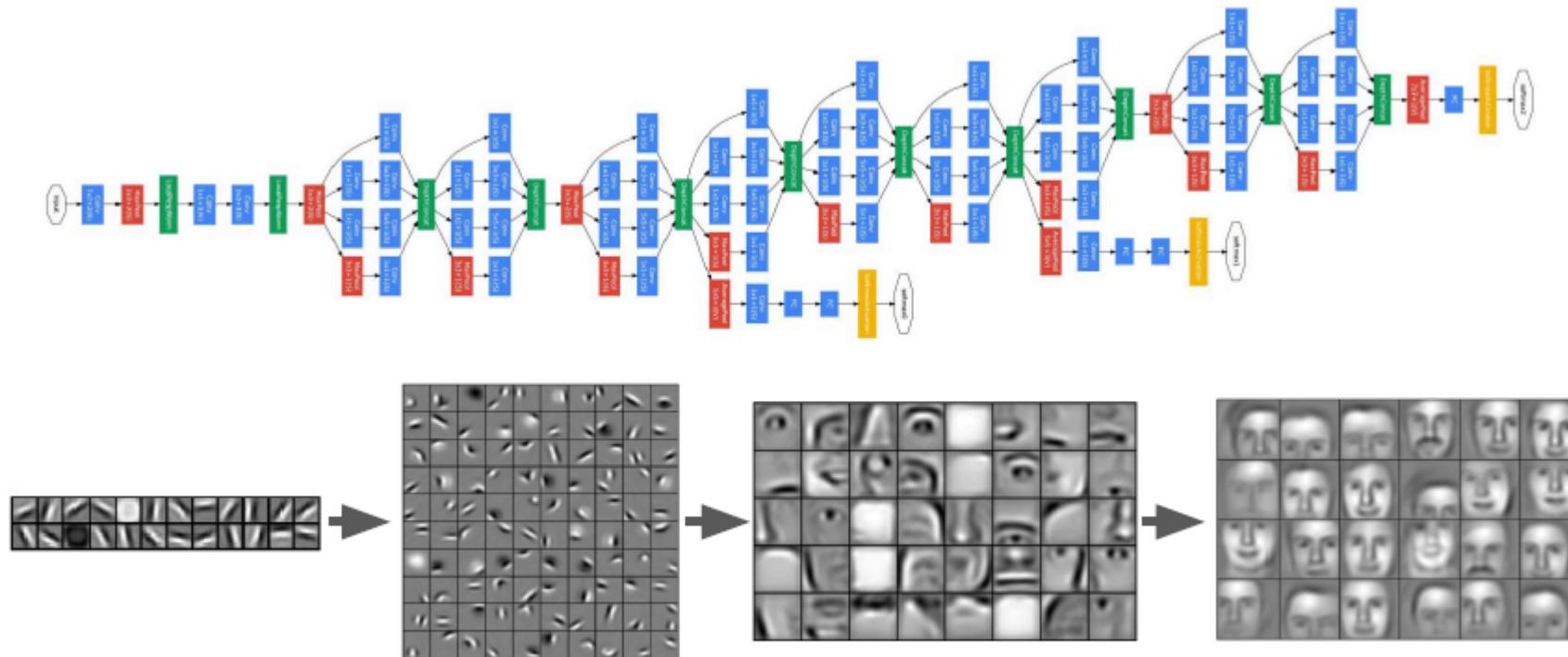


- More hidden layers
- Hierarchies of features
- Generate very complex shapes

Can you Classify This?



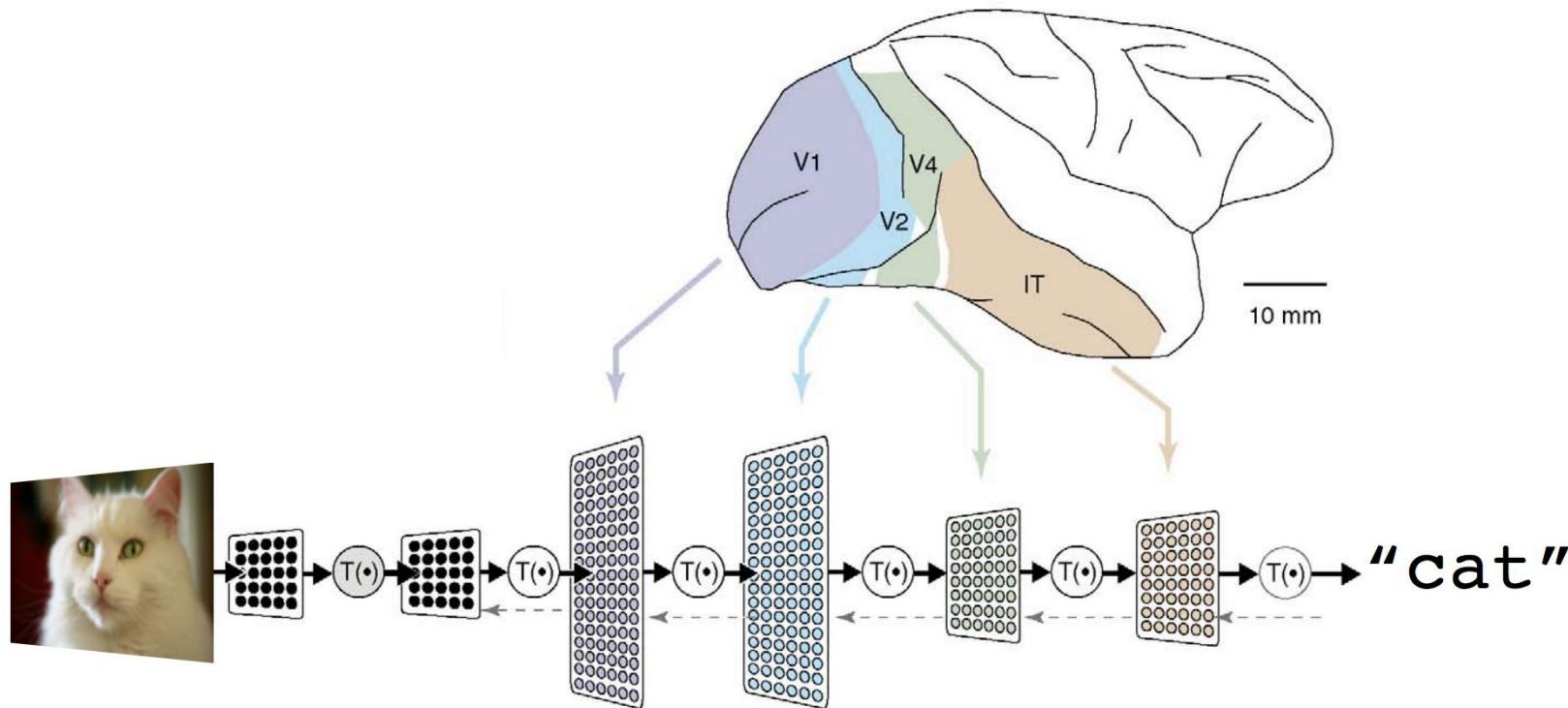
Build a Deep Neural Network



From: [Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations](#), Honglak Lee et al.

Biological Inspiration

- Processing in the brain uses multi-layer processing



History and Why Now?

❑ Early works:

- Using multiple layers dates to 1965 (Ivakhenko and Lapa)
- Convolutional networks with pooling (Fukushima, 1979)
- Back-propagation with a CNN on MNIST (LeCun, 1993)

❑ But, larger networks were a challenge:

- Vanishing gradient
- Lack of data, over-fitting
- Computational power

❑ AlexNet:

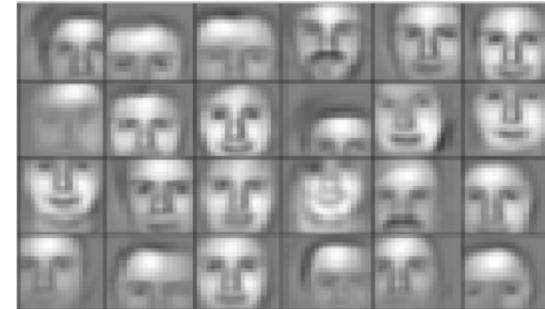
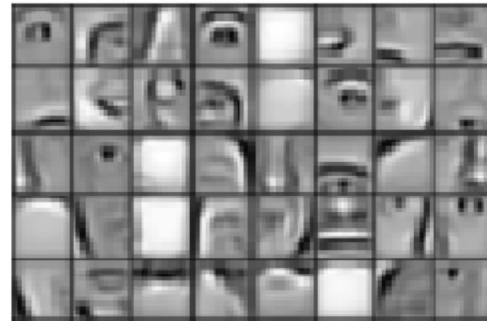
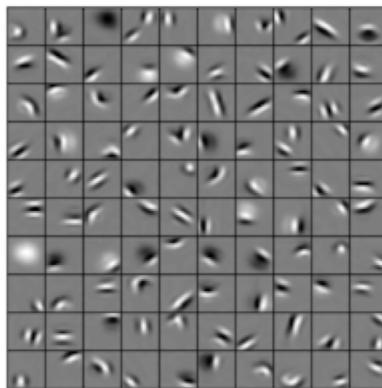
- ReLU and dropout

Outline

- ❑ Motivation: ImageNet Large-Scale Visual Recognition Challenge (ILSVR)
- ❑ Deep Networks and Feature Hierarchies
- 2D convolutions
 - ❑ Convolutional neural networks
 - ❑ Backpropagation training in CNNs
 - ❑ Exploring VGG16: A state-of-the-art deep network

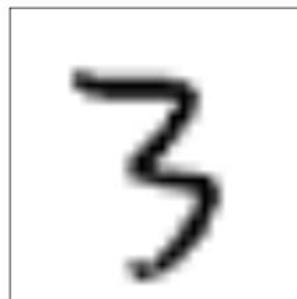
Local Features

- ❑ Early layers in deep neural networks often find local features
- ❑ Small patterns in larger image
 - Examples: Small lines, curves, edges
- ❑ Build more complex classification from the local features



Local Features

- ❑ How do we find local features?
- ❑ A localization problem.
- ❑ Example: Find the digit “3” in the form



HANDWRITING SAMPLE FORM

NAME	DATE	CITY	STATE ZIP
[REDACTED]	8/23/89	Leominster, MA	01453
This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below:			
0123456789	0123456789	0123456789	0123456789
0123456789	0123456789	0123456789	0123456789
07	508	4188	13183
07	508	4188	13183
793094	793094	793094	793094
407	4298	72478	931465
407	4298	72478	931465
22	22	22	22
2567	87516	492935	36
2567	87516	492935	36
600	600	600	600
25649	274951	02	236
25649	274951	02	236
1538	1538	1538	1538
035006	16	953	9458
035006	16	953	9458
67117	67117	67117	67117
x h b e g i l a d j w n f k x a y m i p a e e q			

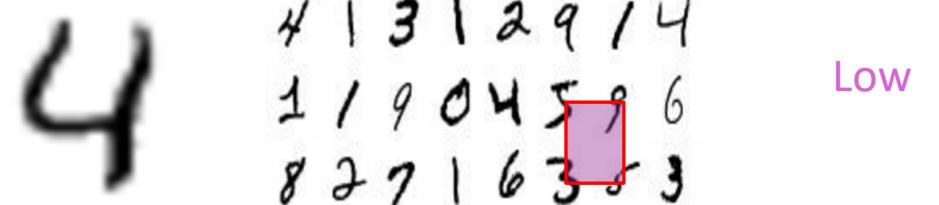
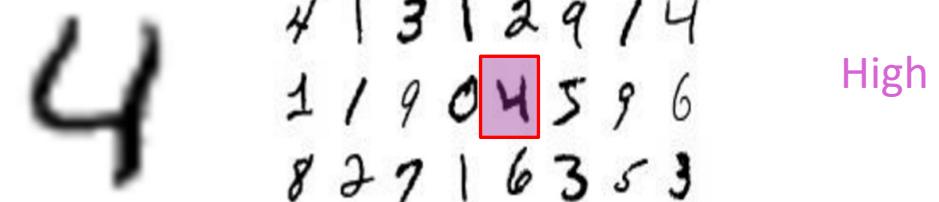
Localization via a Sliding Window

- Simple idea: Find local feature by sliding window
- Large image: $X \ N_1 \times N_2$ (e.g. 512 x 512)
- Small filter: $W \ K_1 \times K_2$ (e.g. 8 x 8)
- At each offset (i, j) compute:

$$Z[i, j] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} W[k_1, k_2] X[i + k_1, j + k_2]$$

- Correlation of W with image box starting at (i, j)
- $Z[i, j]$ is large if feature is present around (i, j)

Filter W Image X $Z[i, j]$



Convolution in 1D

❑ Sliding window is similar to convolution (will make connection precise below)

❑ Given two signals:

- x , length N
- w , length K

❑ Convolution is:

$$z[n] = \sum_{k=0}^{K-1} w[k]x[n - k]$$

- Typically zero pad for samples outside boundary
- Output length is $M = N + K - 1$

❑ Write $z = w * x$

1D Convolution Example

- ❑ Example $x = [1,2,3,4]$, $w = [1,2]$

- ❑ Number outputs = $4+2-1=5$

- ❑ Computations:

$$z[0] = w[0]x[0] = (1)(1) = 1$$

$$z[1] = w[0]x[1] + w[1]x[0] = 2 + 2 = 4$$

$$z[2] = w[0]x[2] + w[1]x[1] = 3 + 4 = 7$$

$$z[3] = w[0]x[3] + w[1]x[2] = 4 + 6 = 10$$

$$z[4] = w[1]x[3] = 8$$

- ❑ Can be computed via flip and shift method

Properties of Convolution

- ❑ Linearity
- ❑ Delta function:

$$x_k * \delta_k = x_k, \quad \delta_k = \begin{cases} 1 & k = 0 \\ 0 & k \neq 0 \end{cases}$$

- ❑ Commutative: $x_k * y_k = y_k * x_k$
- ❑ Shifting: Shift input by $L \Rightarrow$ Output shifted by L

$$z_k = x_k * y_k \Rightarrow z_{k-L} = x_{k-L} * y_k$$

- ❑ Many more: See a signals & systems class

Convolution in 2D

□ Easily extends to higher dimensions

□ Given two images:

- x , size $N_1 \times N_2$
- w , size $K_1 \times K_2$

□ Convolution in 2D is:

$$z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 - k_1, n_2 - k_2]$$

- Output size is $M_1 \times M_2$, $M_1 = N_1 + K_1 - 1$, $M_2 = N_2 + K_2 - 1$

□ Write $z = w * x$

Convolution and Matched Filter

- Recall, we want the sliding correlation:

$$Z[i, j] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} W[k_1, k_2] X[i + k_1, j + k_2]$$

- Given kernel W , define the **matched filter**: $\tilde{W}[k_1, k_2] = W[K_1 - k_1 - 1, K_2 - k_2 - 1]$
 - Flip horizontally and vertically

- Then,

$$Z[i, j] = (\tilde{W} * X)[i, j]$$

- Conclusion: Sliding correlation with $W[k_1, k_2]$ = Convolution with $\tilde{W}[k_1, k_2]$

Terminology

- ❑ In signal processing and math, convolution includes flipping:

$$z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 - k_1, n_2 - k_2]$$

- For this class, we will call this **convolution with reversal**

- ❑ But, in many neural network packages (including Keras), convolution does not include flipping:

$$z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 + k_1, n_2 + k_2]$$

- Will call this **convolution without reversal (= correlation)**

Boundary Conditions

❑ Suppose inputs are

- x , size $N_1 \times N_2$, w : size $K_1 \times K_2$, $K_1 \leq N_1, K_2 \leq N_2$
- $z = x * w$ (without reversal)

$$z[n_1, n_2] = \sum_{k_2=0}^{K_2-1} \sum_{k_1=0}^{K_2-1} w[k_1, k_2] x[n_1 + k_1, n_2 + k_2]$$

❑ Different ways to define outputs

❑ **Valid** mode: $0 \leq n_1 < N_1 - K_1 + 1, 0 \leq n_2 < N_2 - K_2 + 1$

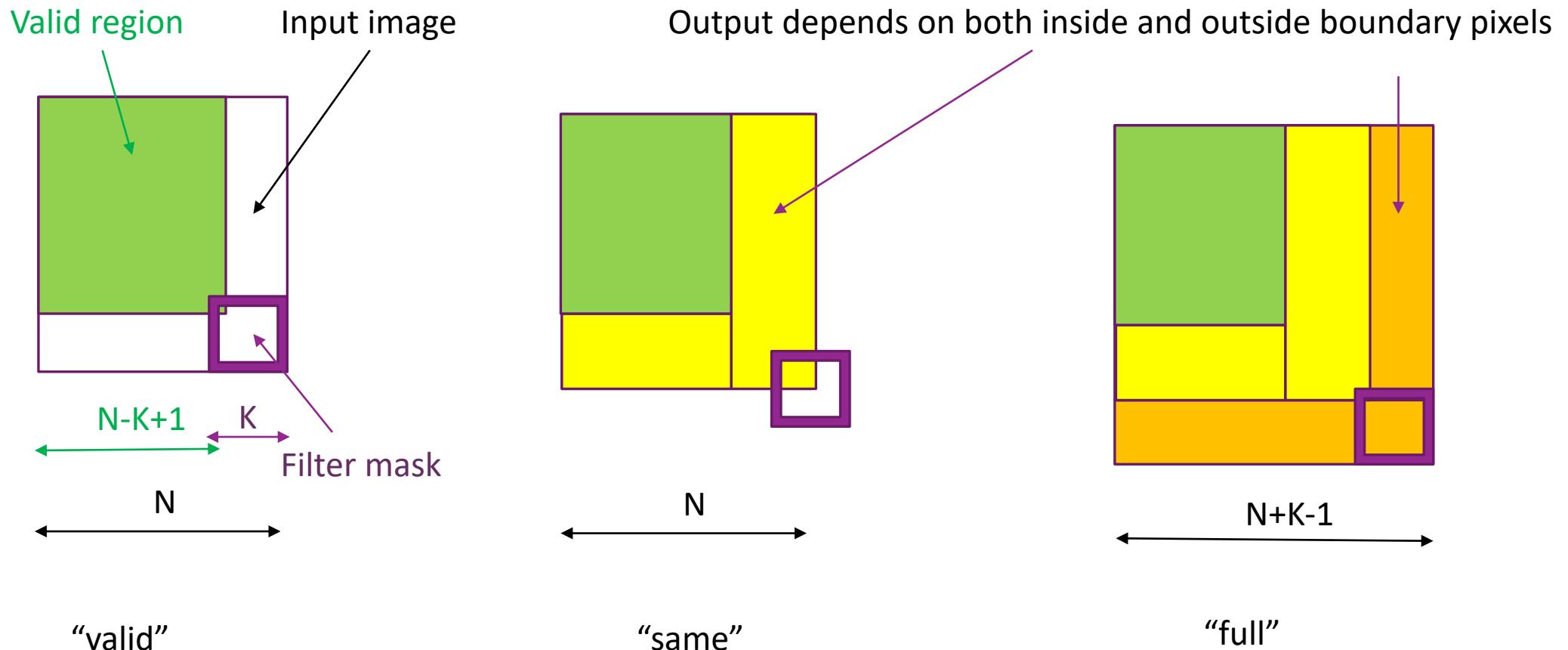
- Requires no zero padding

❑ **Same** mode: Output size $N_1 \times N_2$

- Usually use zero padding for neural networks

❑ **Full** mode: Output size $(N_1 + K_1 - 1) \times (N_2 + K_2 - 1)$

- Not used often in neural networks



Note that with convolution with reversal, the boundary effect will be observed at the top and left sides (see demo_conv)

Convolution 2D Example

❑ Kernel

$$W = \tilde{W} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

- ❑ Compute convolution in valid region
- ❑ Partially compute in class
- ❑ Finish on your own

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

<https://stats.stackexchange.com/questions/199702/1d-convolution-in-neural-networks>

Example Convolution in Python

- ❑ Load an image
- ❑ Use skimage package
 - Many routines for image processing
- ❑ Also need scipy signal package

```
import scipy.signal  
import skimage.data
```

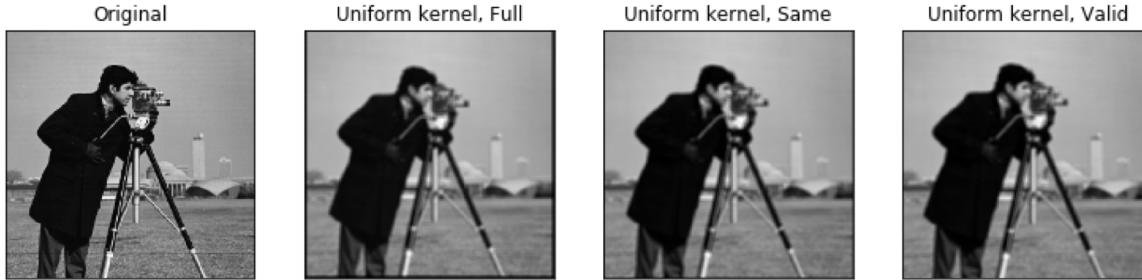
- ❑ Convolution:

```
scipy.signal.convolve2d(im, G, mode='valid')
```

```
im = skimage.data.camera()  
disp_image(im)
```



Using convolution for averaging with different boundary options

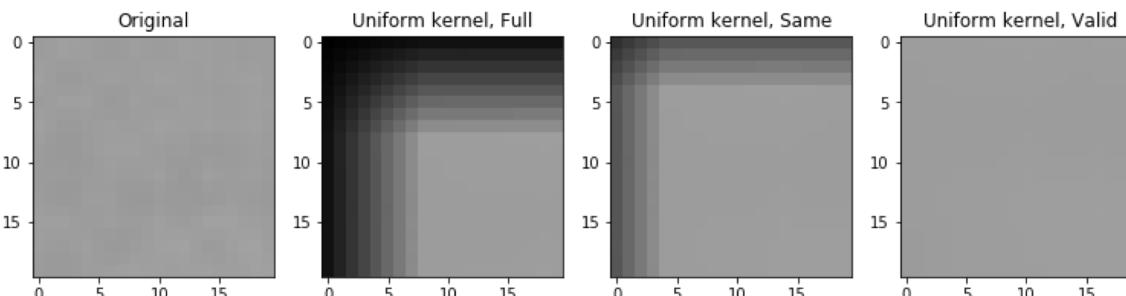


```
kx = 9
ky = 9
sig = 3
G_unif = np.ones((kx,ky))/(kx*ky)
im_unif_full = scipy.signal.convolve2d(im, G_unif, mode='full')
im_unif_same = scipy.signal.convolve2d(im, G_unif, mode='same')
im_unif_valid = scipy.signal.convolve2d(im, G_unif, mode='valid')
```

```
1 print("Input shape = " + str(im.shape))
2 print("Output shape (Full) = " + str(im_unif_full.shape))
3 print("Output shape (Same) = " + str(im_unif_same.shape))
4 print("Output shape (valid) = " + str(im_unif_valid.shape))
```

```
Input shape = (512, 512)
Output shape (Full) = (520, 520)
Output shape (Same) = (512, 512)
Output shape (valid) = (504, 504)
```

```
# Plot the original image and the three outputs at the top left corner
plt.figure(figsize=(13,13))
plt.subplot(1,4,1)
plt.imshow(im[0:20,0:20], vmin=0, vmax=255, cmap='gray')
plt.title('Original')
plt.subplot(1,4,2)
plt.imshow(im_unif_full[0:20,0:20], vmin=0, vmax=255, cmap='gray')
plt.title('Uniform kernel, Full')
plt.subplot(1,4,3)
plt.imshow(im_unif_same[0:20,0:20], vmin=0, vmax=255, cmap='gray')
plt.title('Uniform kernel, Same')
plt.subplot(1,4,4)
plt.imshow(im_unif_valid[0:20,0:20], vmin=0, vmax=255, cmap='gray')
plt.title('Uniform kernel, Valid')
```

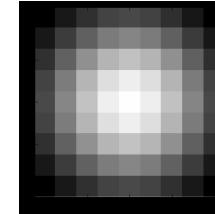


Averaging with different window sizes



Averaging vs. Gaussian Filtering for Noise Removal

- Gaussian filter: Using standard deviation (σ) to control the amount of blurring
Window size $K \geq 2\sigma + 1$



9x9 Gaussian
blur kernel



Gradient filter (edge detection)

- ❑ Sobel filters:

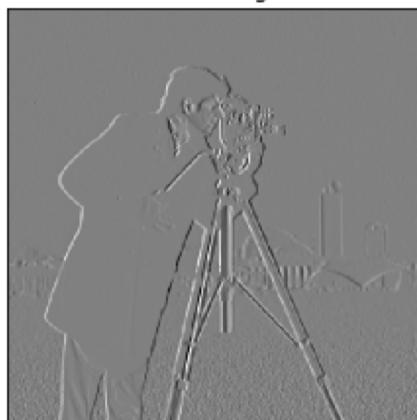
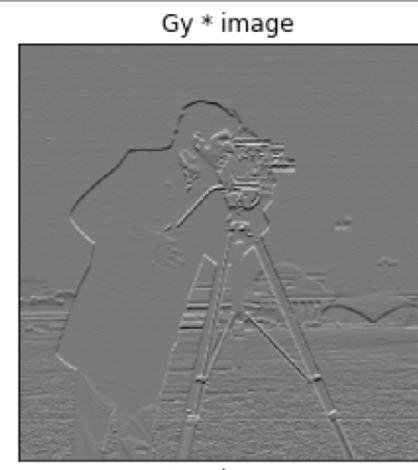
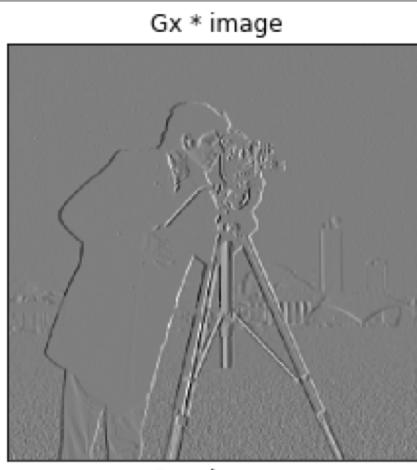
$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- ❑ Define $Z_x = G_x * X$, $Z_y = G_y * X$ (without reversal)

- ❑ Called gradient filters since:

- $Z_x[i, j] = Z_y[i, j] = 0$ in areas where image is constant
- $Z_x[i, j]$ = large positive on strong decrease in x-direction = vertical edge from white to black
- $Z_x[i, j]$ = large negative on strong increase in x-direction = vertical edge from black to white
- $Z_y[i, j]$ is similarly sensitive to horizontal edges

Computing gradients using Sobel filter



Conv. with reversal (using original filter)

```
Gx = np.array([[1,0,-1],[2,0,-2],[1,0,-1]]) # Gradient X
Gy = np.array([[1,2,1],[0,0,0],[-1,-2,-1]]) # Gradient Y
```

Perform the convolutions

```
imx = scipy.signal.convolve2d(im, Gx, mode='valid')
imy = scipy.signal.convolve2d(im, Gy, mode='valid')
```

Input shape = (512, 512)
Output shape = (510, 510)

Conv. without reversal (first flipping the filter)

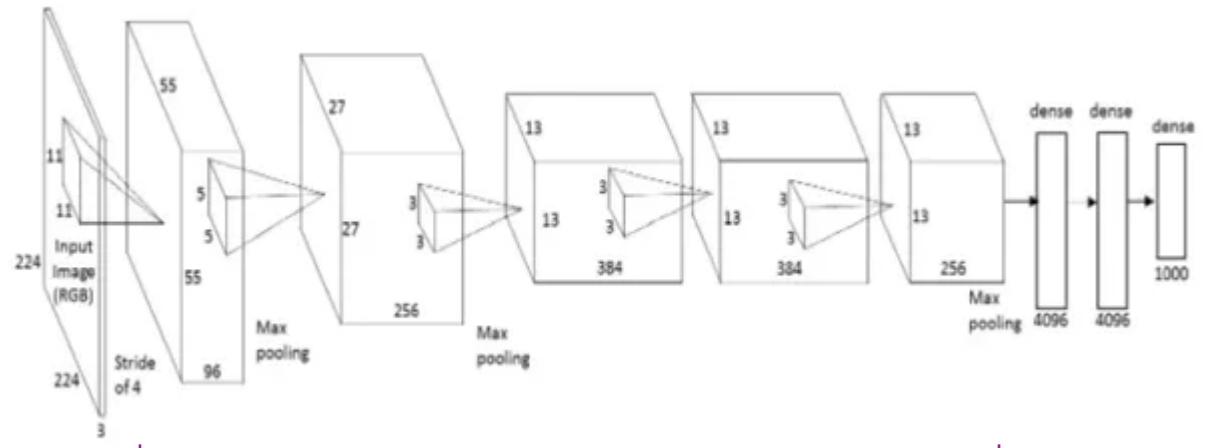
```
Gxflip = np.fliplr(np.flipud(Gx))
Gyflip = np.fliplr(np.flipud(Gy))
```

```
imx = scipy.signal.convolve2d(im, Gxflip, mode='valid')
imy = scipy.signal.convolve2d(im, Gyflip, mode='valid')
```

Outline

- ❑ Motivation: ImageNet Large-Scale Visual Recognition Challenge (ILSVR)
- ❑ Deep Networks and Feature Hierarchies
- ❑ 2D convolutions
-  ❑ Convolutional neural networks
 - ❑ Creating and visualizing convolutional layers in Keras
 - ❑ Backpropagation training in CNNs
 - ❑ Exploring VGG16: A state-of-the-art deep network

Classic CNN Structure



Convolutional layers

2D convolution with
Activation and
pooling / sub-sampling

Fully connected layers

Matrix multiplication &
activation

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

Convolutional Inputs & Outputs

- ❑ Inputs and outputs are images with multiple **channels**

- Number of channels also called the **depth**

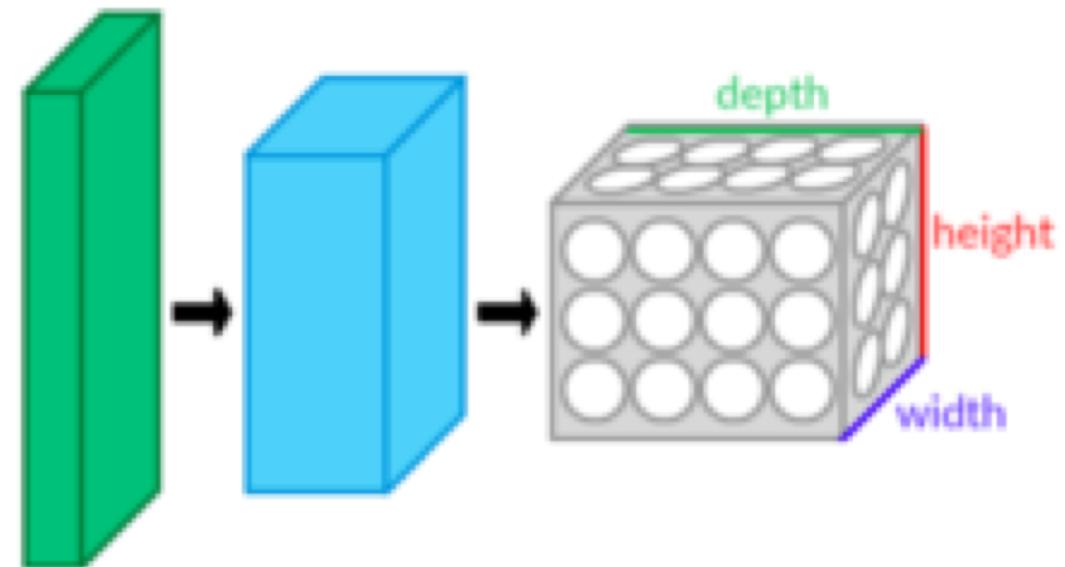
- ❑ Can be described as tensors

- ❑ Input tensor, X shape (N_1, N_2, N_{in})

- N_1, N_2 = input image size
 - N_{in} = number of input channels

- ❑ Output tensor, Z shape (M_1, M_2, N_{out})

- M_1, M_2 = output image size
 - N_{out} = number of output channels



Convolutions with Multiple Channels

□ Weight and bias:

- W : Weight tensor, size $(K_1, K_2, N_{in}, N_{out})$
- b : Bias vector, size N_{out}

□ Convolutions performed over space and added over channels

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

□ For each output channel m , input channel n

- Computes 2D convolution with $W[:, :, n, m]$ (2D filters of size $K_1 \times K_2$)
- Sums results over n
- Different 2D filter for each input channel and output channel pair

Activation and Sub-Sampling

- ❑ Convolution typically followed by activation and pooling
- ❑ Activation, typically ReLU
 - Zeros out negative values
- ❑ Sub-sampling
 - Downsample output after activation
 - Different methods (striding, sub-sampling or max-pooling)
 - Output combines local features from adjacent regions
 - Creates more complex features over wider areas
- ❑ Details for sub-sampling not covered in this class
 - See web for more info

Convolution vs Fully Connected

- ❑ Convolution exploits translational invariance
 - Same features is scanned over whole image
- ❑ Greatly reduces number of parameters
 - N_{in} input channels of size $M_1 \times N_1$, N_{out} output channels with size $M_2 \times N_2$
 - Fully connected network: $N_{in} * N_{out} * M_1 * N_1 * M_2 * N_2 + N_{out} * M_2 * N_2$
 - Convolutional network with $K_1 \times K_2$ filter: $N_{in} * N_{out} * K_1 * K_2 + N_{out}$
- ❑ Example: Consider first layer in LeNet
 - 32×32 image (1 channel) to 6 channels using 5×5 filters
 - Creates $6 \times 28 \times 28$ outputs (keeping only the valid region)
 - Fully connected would require $32 \times 32 \times 6 \times 28 \times 28 + 6 \times 28 \times 28 = 4.9$ million parameters!
 - Convolutional layer requires only $6 \times 5 \times 5 + 6 = 156$ parameters
 - Reserve fully connected layers for last few layers (for non-image output such as classification).

Outline

- ❑ Motivation: ImageNet Large-Scale Visual Recognition Challenge (ILSVR)
- ❑ Deep Networks and Feature Hierarchies
- ❑ 2D convolutions
- ❑ Convolutional neural networks
- Creating and visualizing convolutional layers in Keras
- ❑ Backpropagation training in CNNs
- ❑ Exploring VGG16: A state-of-the-art deep network

Creating Convolutional Layers in Keras

❑ Done easily with Conv2d

- Specify input_shape (if first layer), kernel size and number of output channels

❑ To illustrate:

- We create a network with a single convolutional layer
- Set the weights and biases (normally these would be learned)
- Run input through the layer (using the predict command)
- Look at the output

```
# Create network
K.clear_session()
model = Sequential()
model.add(Conv2D(input_shape=input_shape,filters=nchan_out,
                 kernel_size=kernel_size,name='conv2d'))
```

Example 1: Gradients of a BW image

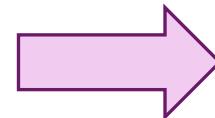
- ❑ Create simple convolutional layer
- ❑ Input: BW image, $N_{in} = 1$ input channel
- ❑ Two output channels: x- and y-gradient, $N_{out} = 2$



Input.
One channel
Shape = (512,512,1)

$$* \quad G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Filters
Two gradient



Create a Layer in Keras

```
K.clear_session()  
model = Sequential()  
kernel_size = Gx.shape  
nchan_out = 2  
model.add(Conv2D(input_shape=input_shape,filters=nchan_out,  
                 kernel_size=kernel_size,name='conv2d'))
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 510, 510, 2)	20

Total params: 20
Trainable params: 20
Non-trainable params: 0

❑ Create a single layer model

❑ Use the Conv2D layer

❑ Specify

- Kernel size
- Number of output channels
- Input shape

❑ Why do we have 20 parameters?

Set the Weights

```
layer = model.get_layer('conv2d')
W, b = layer.get_weights()
print("W shape = " + str(W.shape))
print("b shape = " + str(b.shape))
```

```
W shape = (3, 3, 1, 2)
b shape = (2,)
```

```
W[:, :, 0, 0] = Gx
W[:, :, 0, 1] = Gy
b = np.zeros(nchan_out)
layer.set_weights((W,b))
```

```
x = im.reshape(batch_shape)
y = model.predict(x)
```

□ Read the weights and the shapes

□ Set the weights to the two filters

□ Normally, these would be trained

□ Run the input through the network

Perform Convolution in Keras

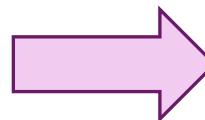
- ❑ Create input x
 - Need to reshape
- ❑ Use predict command to compute output
- ❑ Generates two output channels y

```
x = im.reshape(batch_shape)  
y = model.predict(x)
```

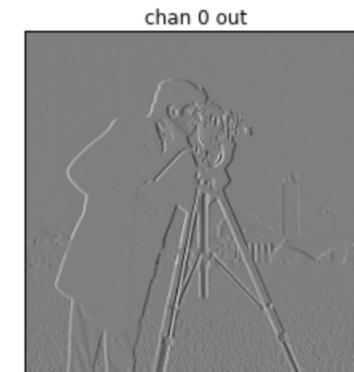


*

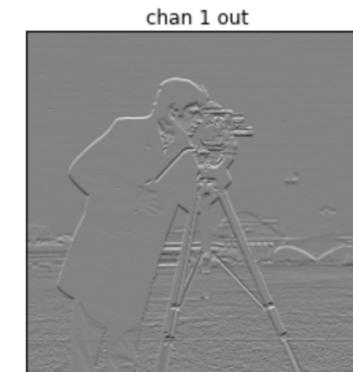
Filters
Two gradients



y[:, :, 0]



y[:, :, 1]



Example 2: Color Input

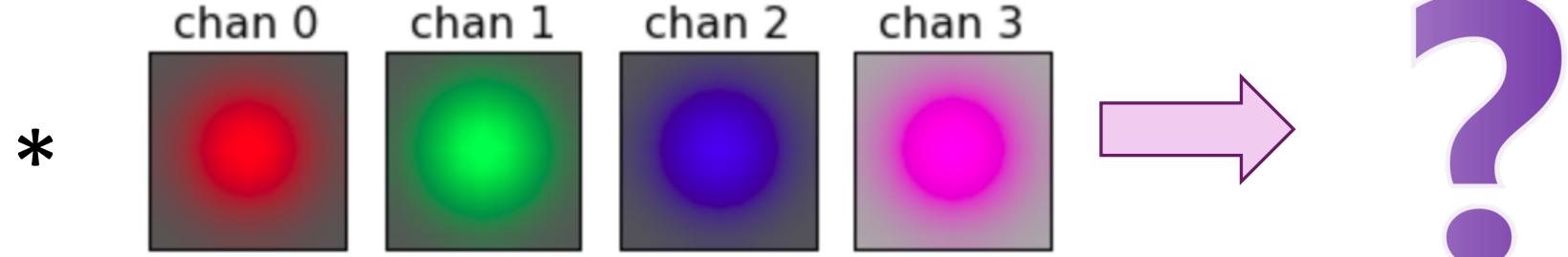
❑ Input: Single color input

- $N_{in} = 3$ input channels
- Input size per sample = $368 \times 487 \times 3$

❑ Output: Filter with four different color filters

- Each kernel is 9×9
- $N_{out} = 4$ output channels

Image shape is $(368, 487, 3)$



Create the Layer in Keras

```
# Dimensions  
nchan_out = 4  
kernel_size = (9,9)  
  
# Create network  
K.clear_session()  
model = Sequential()  
model.add(Conv2D(input_shape=input_shape,filters=nchan_out,  
                 kernel_size=kernel_size,name='conv2d'))
```

```
model.summary()
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 360, 479, 4)	976
Total params:	976	
Trainable params:	976	
Non-trainable params:	0	

- ❑ Model with single layer
- ❑ Input shape = (368, 488, 3)

- ❑ Output shape = (360, 479, 4)
- ❑ What the number of parameters is 976?

Set the Weights

```
# Color weights
color_wt = np.array([
    [1, -0.5, -0.5],    # Sensitive to red
    [-0.5, 1, -0.5],    # Sensitive to green
    [-0.5, -0.5, 1],    # Sensitive to blue
    [0.5, -1, 0.5],    # Sensitive to red-blue mix
])

# Gaussian kernel over space
krow, kcol = kernel_size
G = gauss_kernel(krow,kcol,sig=2)

# Multiply by weighting color
W = G[:, :, None, None]*color_wt.T[None, None, :, :]
b = np.zeros(b.shape)
layer.set_weights((W,b))
```

- Consider weight of the form:

$$W[i, j, k, \ell] = G[i, j]C[\ell, k]$$

- $G[i, j]$ = filter over space
 - Use Gaussian blur

- $C[\ell, k]$ = filter over channel
 - Weighting of color k in output channel ℓ

- Each filter:
 - Average over space and selects color

- Again, normally we would train the weights

Perform Convolution

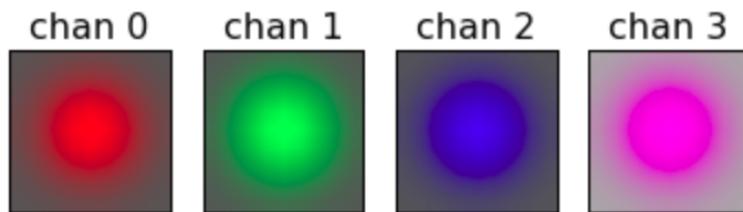
Input, x

(3 channels)

Image shape is (368, 487, 3)

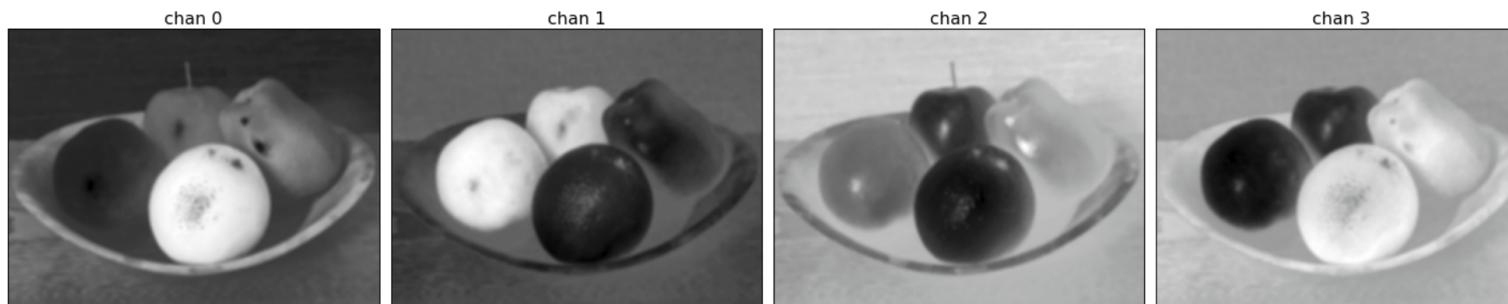


*



Filters, W
(9,9,3,4)

$y = \text{model.predict}(x)$



Output y
4 channels

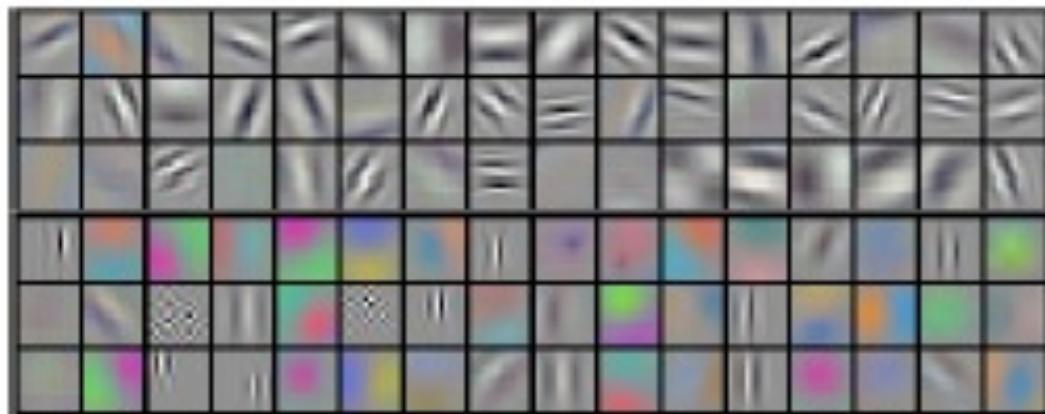
$y[:, :, 0]$

$y[:, :, 1]$

$y[:, :, 2]$

$y[:, :, 3]$

First layer filter in AlexNet



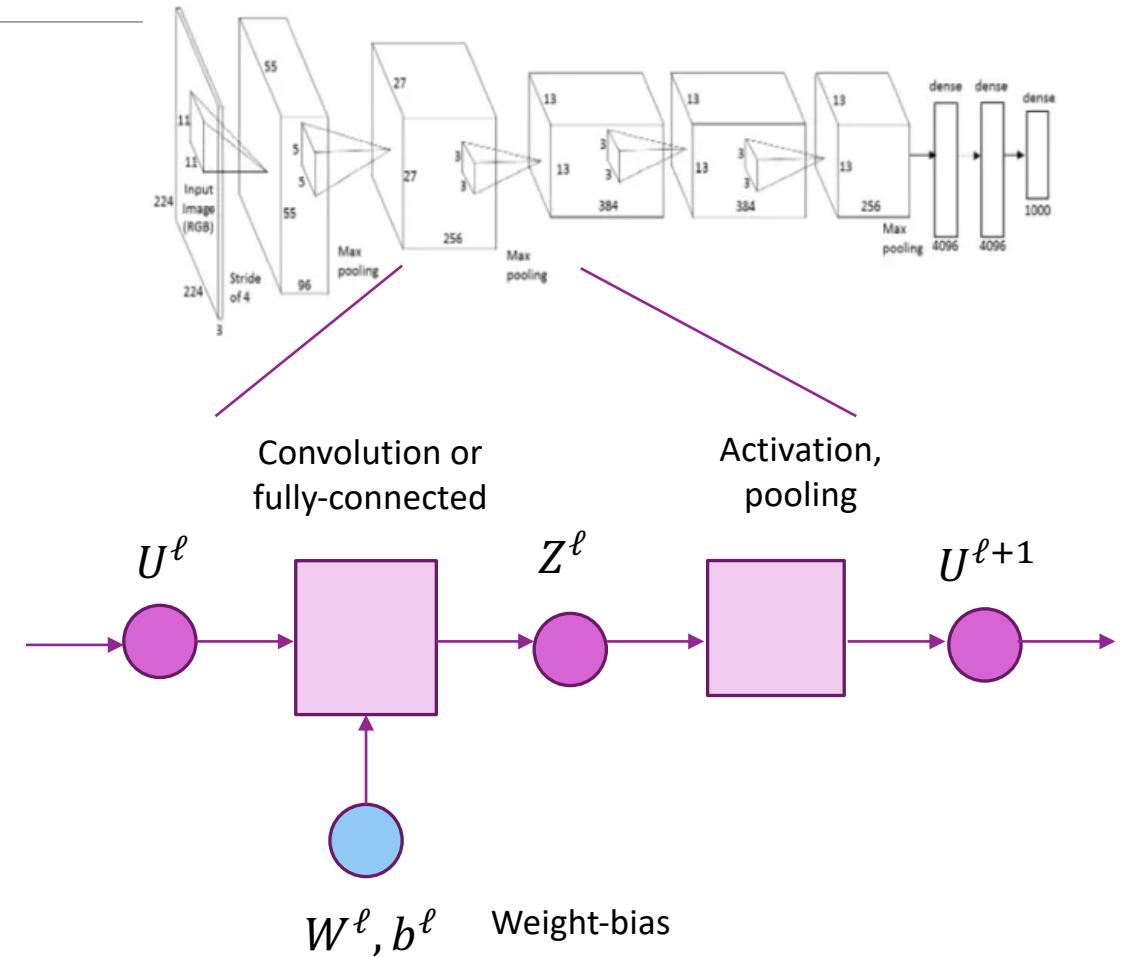
- ❑ AlexNet first layer
 - 96 filters
 - Size $11 \times 11 \times 3$
 - Applied to image of $224 \times 224 \times 3$
- ❑ What do these learned features look like?
- ❑ Selective to basic low-level features
 - Curves, edges, color transitions, ...

Outline

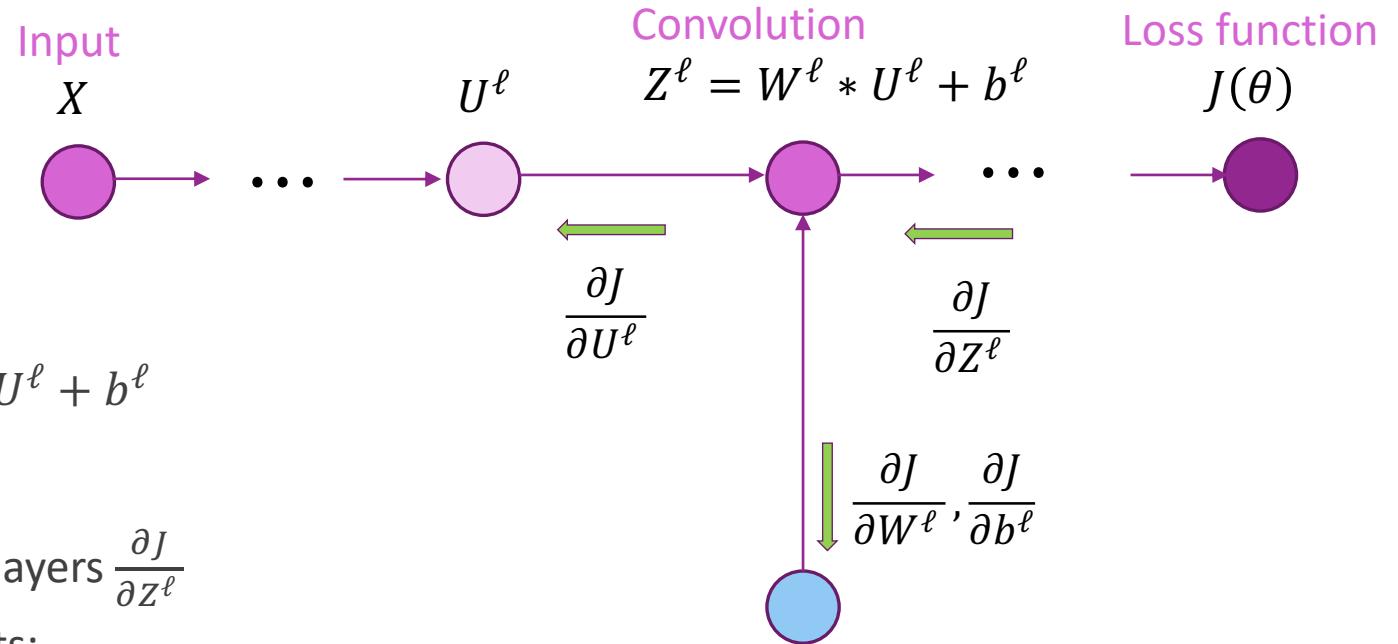
- ❑ Motivation: ImageNet Large-Scale Visual Recognition Challenge (ILSVR)
- ❑ Deep Networks and Feature Hierarchies
- ❑ 2D convolutions
- ❑ Convolutional neural networks
- ❑ Creating and visualizing convolutional layers in Keras
-  Backpropagation training in CNNs
- ❑ Exploring VGG16: A state-of-the-art deep network

Indexing Multi-Layer Networks

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



Back-Propagation in Convolutional Layers



□ Convolutional layer in forward path

$$Z^\ell = W^\ell * U^\ell + b^\ell$$

□ During back-propagation:

- Obtain gradient tensor from upstream layers $\frac{\partial J}{\partial Z^\ell}$
- Need to compute downstream gradients:

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

W^ℓ, b^ℓ

Gradient Details

- Write convolution as:

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

- Drop layer index ℓ and sample index i
- In backpropagation, we receive gradient tensor: $\frac{\partial J}{\partial Z[i_1, i_2, m]}$
- First compute gradient wrt weights: $\frac{\partial J}{\partial W[k_1, k_2, n, m]}$

Gradient With Respect to Weights

- ❑ Gradient wrt weights:

$$\frac{\partial Z[i_1, i_2, m]}{\partial W[k_1, k_2, n, m]} = U[i_1 + k_1, i_2 + k_2, n]$$

- ❑ By chain rule:

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

- ❑ Gradient wrt weights can be computed via convolution

- Convolve input U with gradient tensor $\frac{\partial J}{\partial Z[i_1, i_2, m]}$

- ❑ Similar computations for gradients with respect to $\frac{\partial J}{\partial b}$, $\frac{\partial J}{\partial U}$

- See homework

GPUs

- ❑ State-of-the-art networks involve millions of parameters
- ❑ Require enormous datasets
- ❑ Conventional processors cannot train in reasonable time
- ❑ Use **Graphics Processor Units**
 - Originally for graphics acceleration
 - Now essential for deep learning
- ❑ Cannot use the GPU on your laptop
- ❑ But, can:
 - Rent GPU instances in cloud (~\$0.80 / hour)
 - Purchase GPU workstation (~\$2000)



Batch Size	Training Time CPU	Training Time GPU	GPU Speed Up
64 images	64 s	7.5 s	8.5X
128 images	124 s	14.5 s	8.5X
256 images	257 s	28.5 s	9.0X

Speed up on 2012 ImageNet winner using Nvidia Tesla K40
From http://www.nvidia.com/content/events/geoInt2015/LBrown_DL.pdf

Much faster results available today

Outline

- ❑ Motivation: ImageNet Large-Scale Visual Recognition Challenge (ILSVR)
 - ❑ Deep Networks and Feature Hierarchies
 - ❑ 2D convolutions
 - ❑ Convolutional neural networks
 - ❑ Creating and visualizing convolutional layers in Keras
 - ❑ Backpropagation training in CNNs
-  Exploring VGG16: A state-of-the-art deep network

Pre-Trained Networks

- ❑ State-of-the-art networks take enormous resources to train
 - Millions of parameters
 - Often days of training, clusters of GPUs
 - Extremely expensive
- ❑ Pre-trained networks in Keras
 - Load network architecture and weights
 - Models available for many state-of-the-art networks
- ❑ Can be used for:
 - Making predictions
 - Building new, powerful networks (see lab)

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88

<https://keras.io/applications/>

VGG16

- ❑ From the Visual Geometry Group

- Oxford, UK

- ❑ Won ImageNet ILSVRC-2014

- ❑ Remains a very good network

- ❑ Will load this network today

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

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

K. Simonyan, A. Zisserman

[**Very Deep Convolutional Networks for Large-Scale Image Recognition**](#)
arXiv technical report, 2014

Loading the Pre-Trained Network

```
from keras.applications.vgg16 import VGG16  
from keras.preprocessing import image  
from keras.applications.vgg16 import preprocess_input, decode_predictions
```

❑ Load the packages

```
model = VGG16(weights='imagenet')
```

❑ Create the model

- Downloads the h5 file
- First time, may be a while
- 500 MB file

Display the Network

```
: model.summary()
```

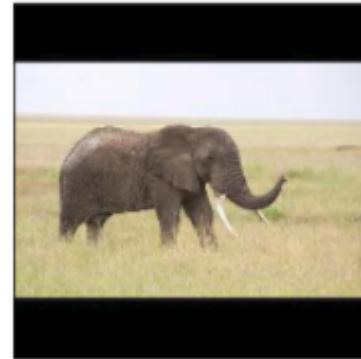
Layer (type)	Output Shape	Param #
<hr/>		
input_5 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0

block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000
<hr/>		
Total params: 138,357,544		
Trainable params: 138,357,544		
Non-trainable params: 0		

- ❑ Very deep: 16 layers
- ❑ 130 million parameters!

Get Some Test Images

- ❑ Get images from the web of some category (e.g. elephants)
- ❑ Many possible sources.
 - Example: Flickr API (see Demo in github)
- ❑ Re-size / pad images so that they match expected input of VGG16
 - Input shape (224, 224, 3)



Make Predictions

```
x = preprocess_input(x)
```

```
preds = model.predict(x)
preds_decoded = decode_predictions(preds, top=3)
```

	class 0	class 1	class 2	prob 0	prob 1	prob 2
0	Indian_elephant	African_elephant	tusker	0.776757	0.196798	0.026314
1	African_elephant	tusker	Indian_elephant	0.514596	0.414825	0.057157
2	tusker	Indian_elephant	African_elephant	0.682218	0.217784	0.099942
3	African_elephant	tusker	Indian_elephant	0.736568	0.228160	0.035263
4	African_elephant	tusker	Indian_elephant	0.409717	0.301944	0.287880
5	water_buffalo	African_elephant	warthog	0.737919	0.129731	0.037343
6	African_elephant	tusker	Indian_elephant	0.745698	0.140428	0.103136
7	Indian_elephant	tusker	African_elephant	0.970890	0.026875	0.002234
8	African_elephant	tusker	Indian_elephant	0.819497	0.108567	0.067853
9	tusker	African_elephant	Indian_elephant	0.499149	0.338156	0.162537

❑ Pre-process

❑ Predict

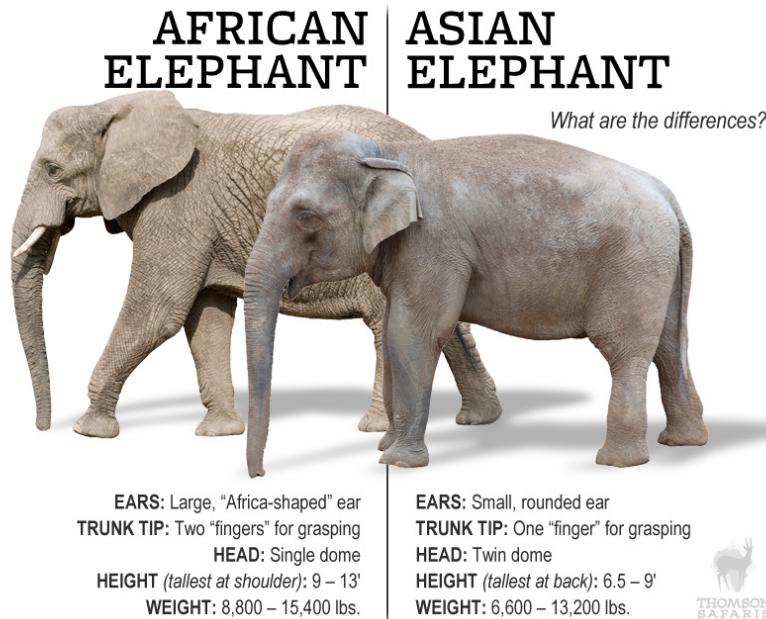
- Runs input through network

❑ Decode predictions

- Creates data structure for outputs

ImageNet Classification can be Hard

- Some categories differences are subtle



	class 0	class 1	class 2	prob 0	prob 1	prob 2
0	Indian_elephant	African_elephant	tusker	0.776757	0.196798	0.026314
1	African_elephant	tusker	Indian_elephant	0.514596	0.414825	0.057157
2	tusker	Indian_elephant	African_elephant	0.682218	0.217784	0.099942
3	African_elephant	tusker	Indian_elephant	0.736568	0.228160	0.035263
4	African_elephant	tusker	Indian_elephant	0.409717	0.301944	0.287880
5	water_buffalo	African_elephant	warthog	0.737919	0.129731	0.037343
6	African_elephant	tusker	Indian_elephant	0.745698	0.140428	0.103136
7	Indian_elephant	tusker	African_elephant	0.970890	0.026875	0.002234
8	African_elephant	tusker	Indian_elephant	0.819497	0.108567	0.067853
9	tusker	African_elephant	Indian_elephant	0.499149	0.338156	0.162537

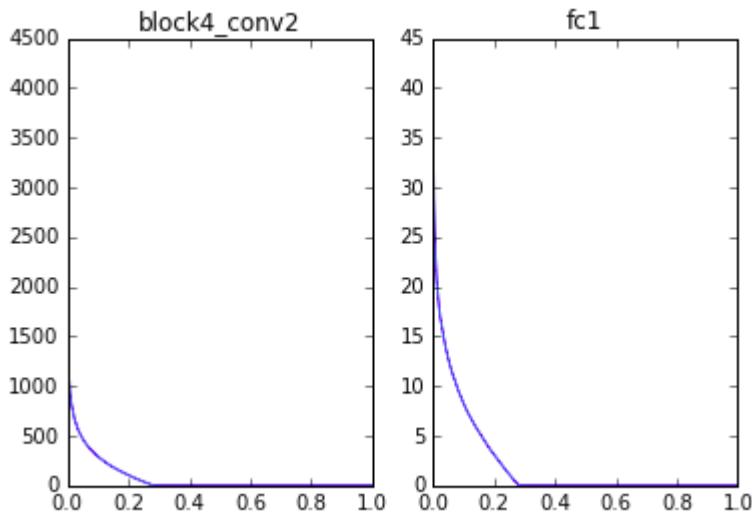
Intermediate Layers

```
from keras.models import Model

# Construct list of layers
layer_names = ['block4_conv2', 'fc1']
out_list = []
for name in layer_names:
    out_list.append(model.get_layer(name).output)

# Create the model with the intermediate layers
model_int = Model(inputs=model.input, outputs=out_list)

y = model_int.predict(x)
```



- ❑ Often need outputs of hidden layers
- ❑ Provides “latent” representation of image
 - Can be useful for other tasks
 - See lab
- ❑ In Keras, create new model
 - Specify output layers
- ❑ Predict with new model to extract hidden outputs
- ❑ Hidden layers see high level of **sparsity**
 - Many coefficients are zero

Try It Yourself!

In-Class Exercise

Find any image of your choice and use the pre-trained network to make a prediction.

- Download the image to your directory
- Load it into an image batch
- Predict the class label and decode the predictions



= ?