Do not distribute course material

You may not and may not allow others to reproduce or distribute lecture notes and course materials publicly whether or not a fee is charged.

Lecture 10 Convolutional and Deep Neural Networks

https://cs231n.github.io/convolutional-networks/

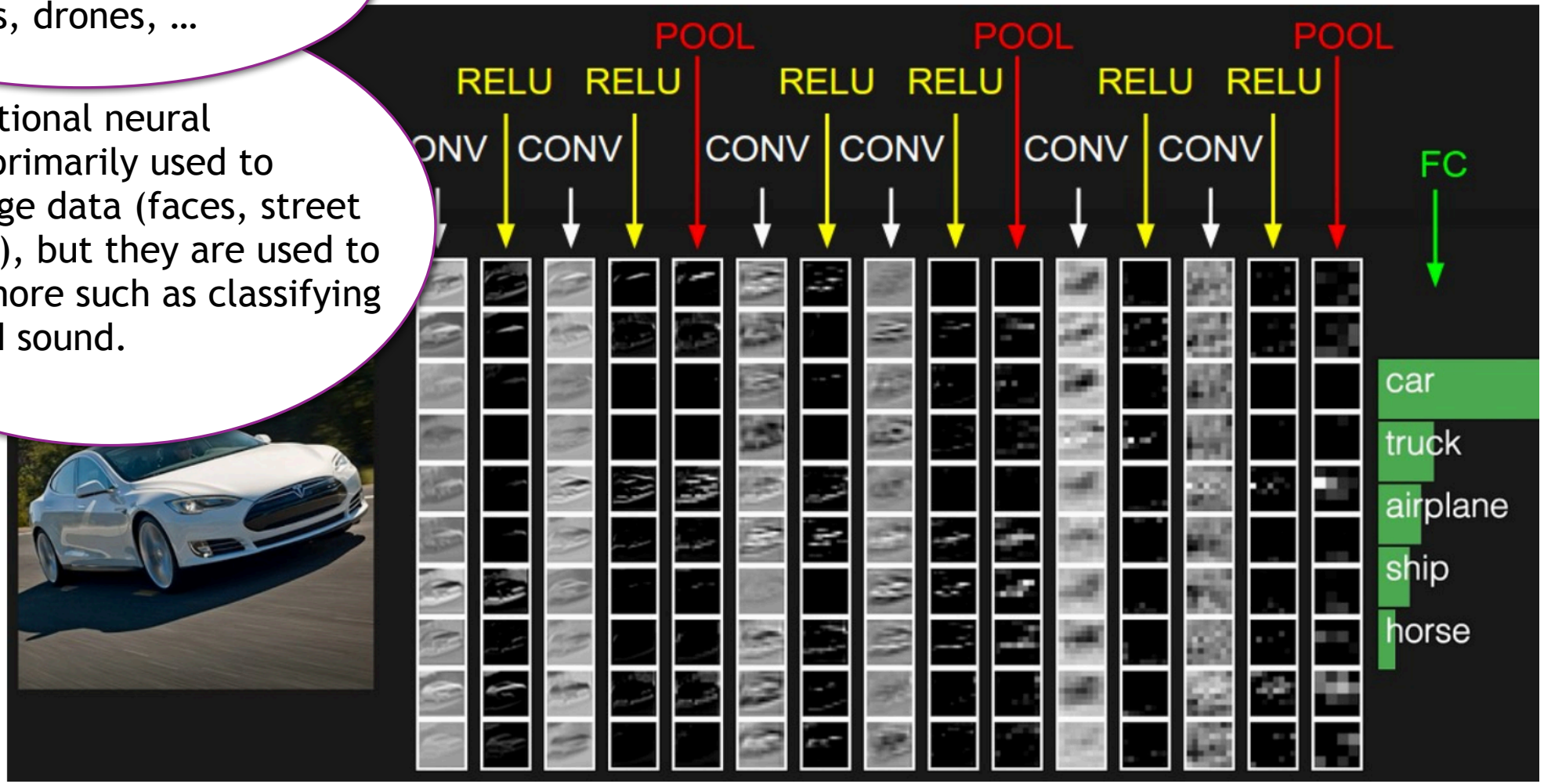
EE-UY 4563/EL-GY 9123: INTRODUCTION TO MACHINE LEARNING

PROF LINDA SELLIE

MANY OF THESE SLIDES ARE FROM PROF RANGAN, YAO WANG (NYU), PHIL SCHNITER (OSU)

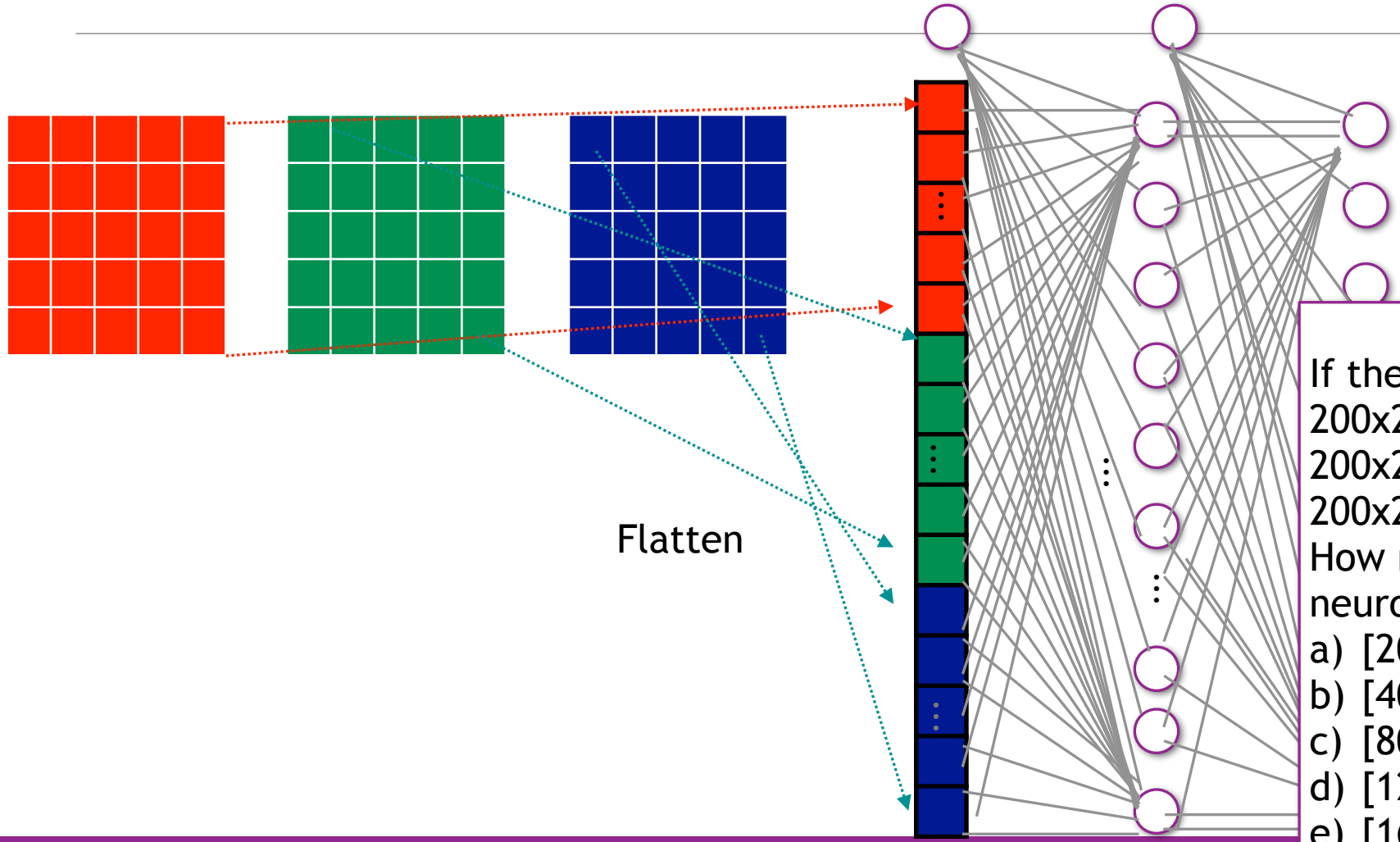
Used in self-driving cars, robotics, drones, medical diagnosis, drones, …

Convolutional neural networks primarily used to classify image data (faces, street signs, digits), but they are used to do much more such as classifying text and sound.



https://cs231n.github.io/convolutional-networks/

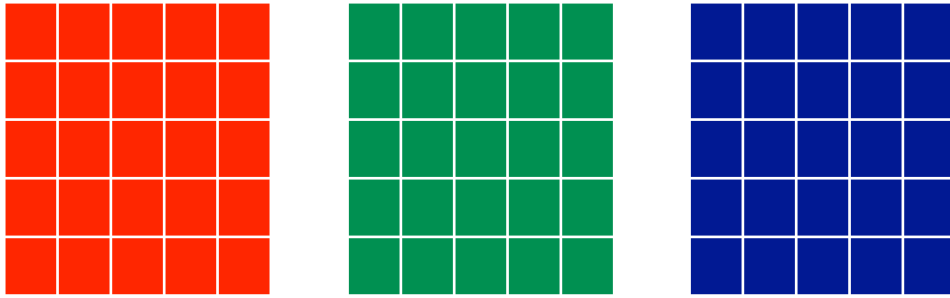
Fully connected neural networks



If the image is
200x200 red pixels
200x200 green pixels
200x200 blue pixels
How many weights lead to a single neuron on layer 2

a) [200, 400)
b) [400, 800)
c) [800, 1200)
d) [1200, 1600)
e) [1600, 2000)
f) More than 2000

Changing the architecture



- Last lecture we discussed *fully connected* neural networks
- We can learn hand written digits (given as an image), but when the image is larger the number of weights we have to train becomes infeasible! (If the input is a color (red, green, blue are stored separately) image of 200x200 pixels we would have 200x200x3+1 = **120,000 +1 weights** for ***a single neuron*** in the next layer. This leads to overfitting (and is expensive in both time and memory)
- How can we reduce the number of weights?

Example from https://cs231n.github.io/convolutional-networks/

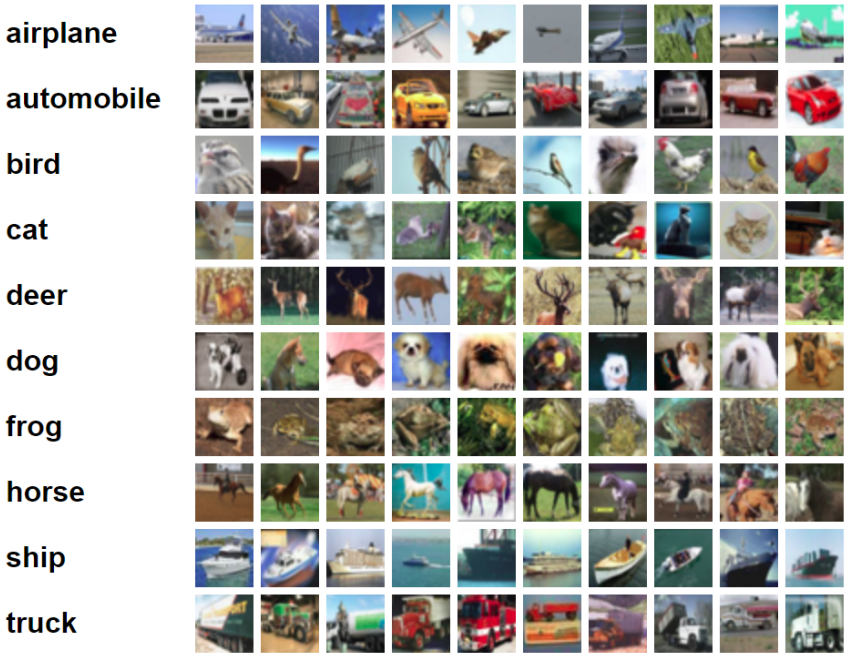
- Often in image data we can speed up learning (or any input that has "spacial locality" and "translation invariance") by **changing how the layers connect to each other**
  - Input layer
  - Hidden layers alter data in the hope of finding useful features

    how can we modify these layers?
  - Output layer

Large-Scale Image Classification

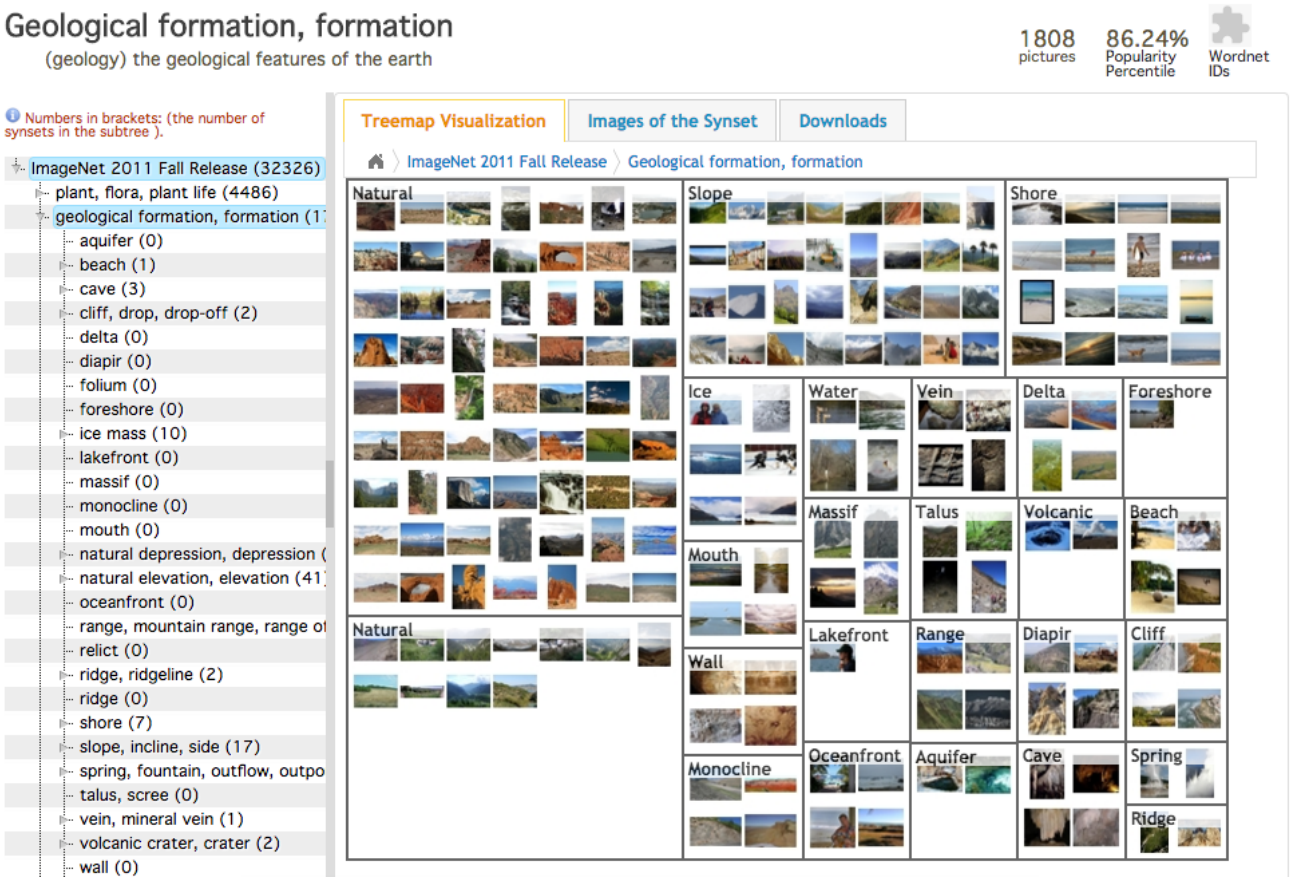
- Pre-2009, many image recognition systems worked on relatively small datasets
  - MNIST: 10 digits
  - CIFAR 10 (right)
  - CIFAR 100
  - …
- Small number of classes (10-100)
- Low resolution (eg. 32 x 32 x 3)
- Performance saturated
  - Difficult to make significant advancements

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



ImageNet (2009)

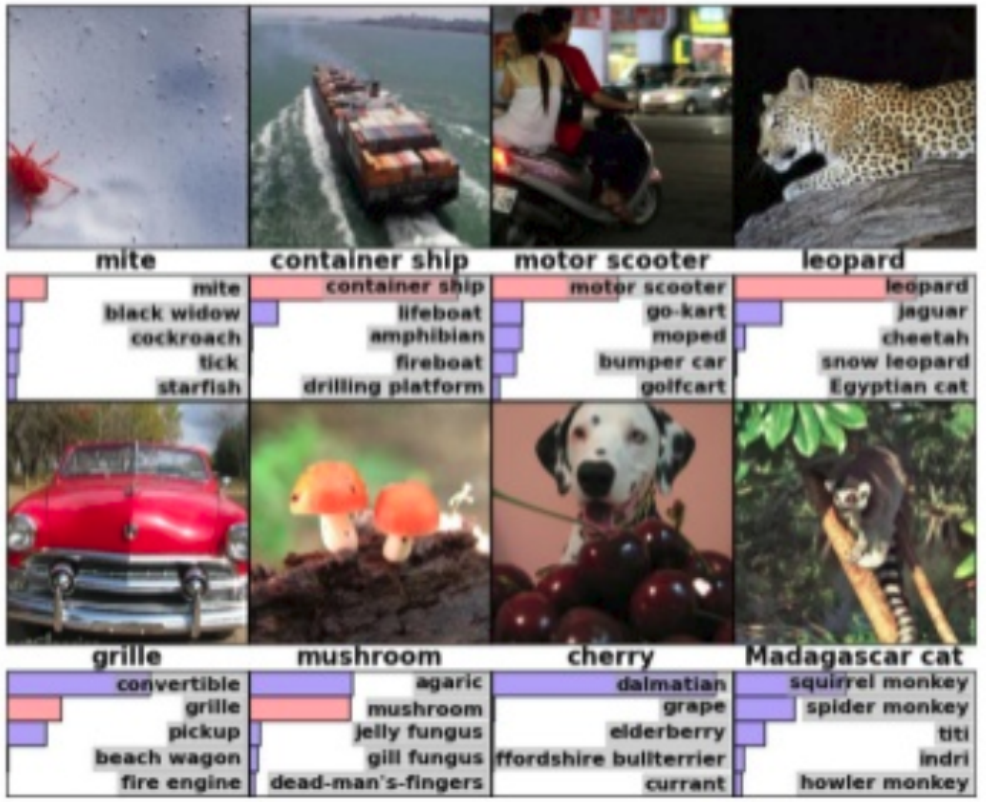
- Better algorithms need better data
- Build a large-scale image dataset
- 2009 CVPR paper:
  - 3.2 million images
  - Annotated by mechanical turk
  - Much larger scale than any previous
- Hierarchical categories



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.

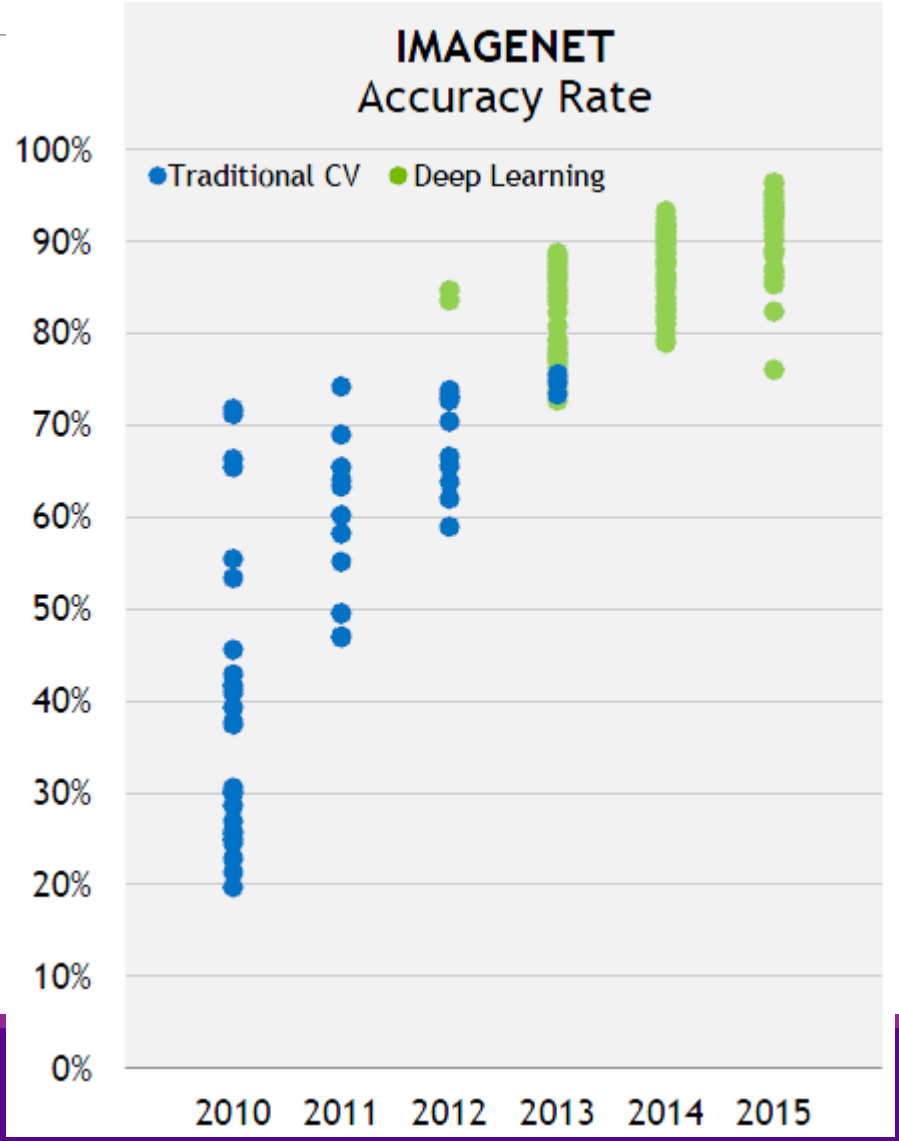
ILSVRC

- 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)
  - …



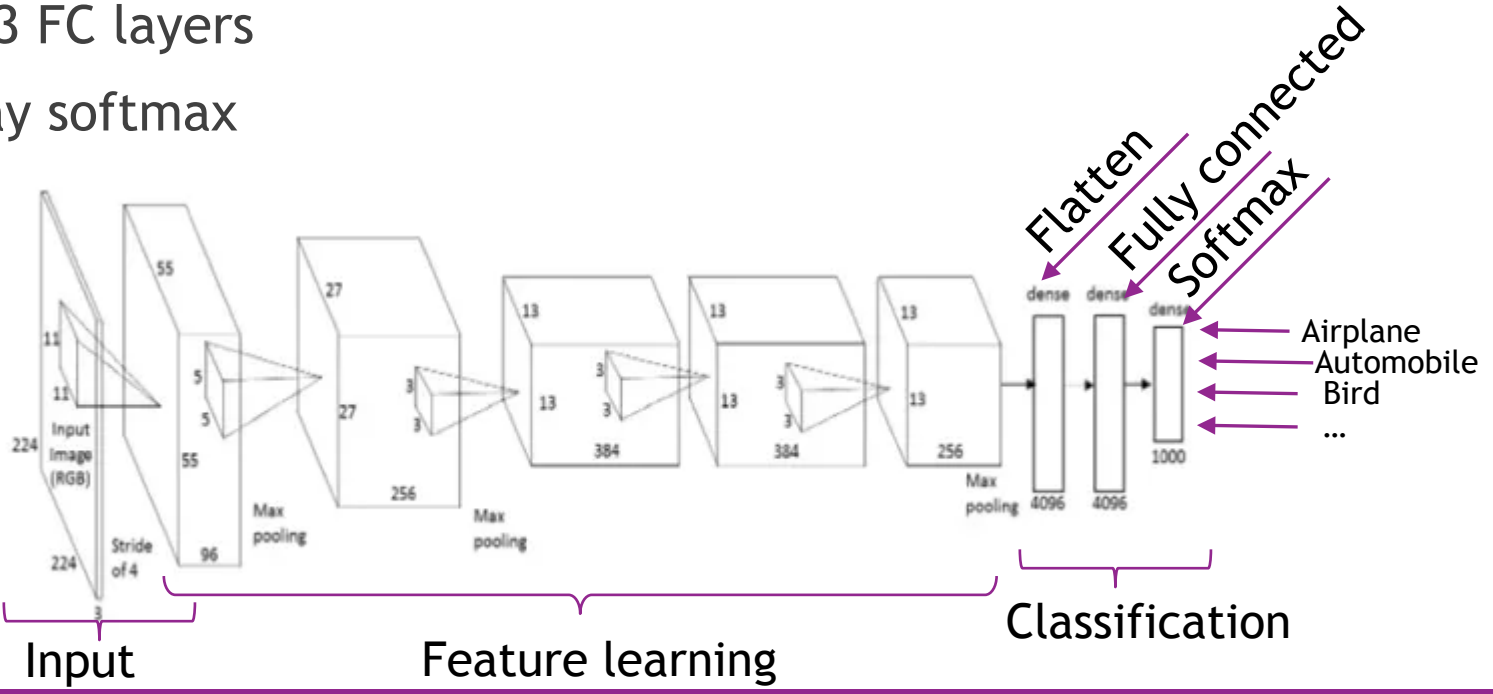
Deep Networks Enter 2012

- 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

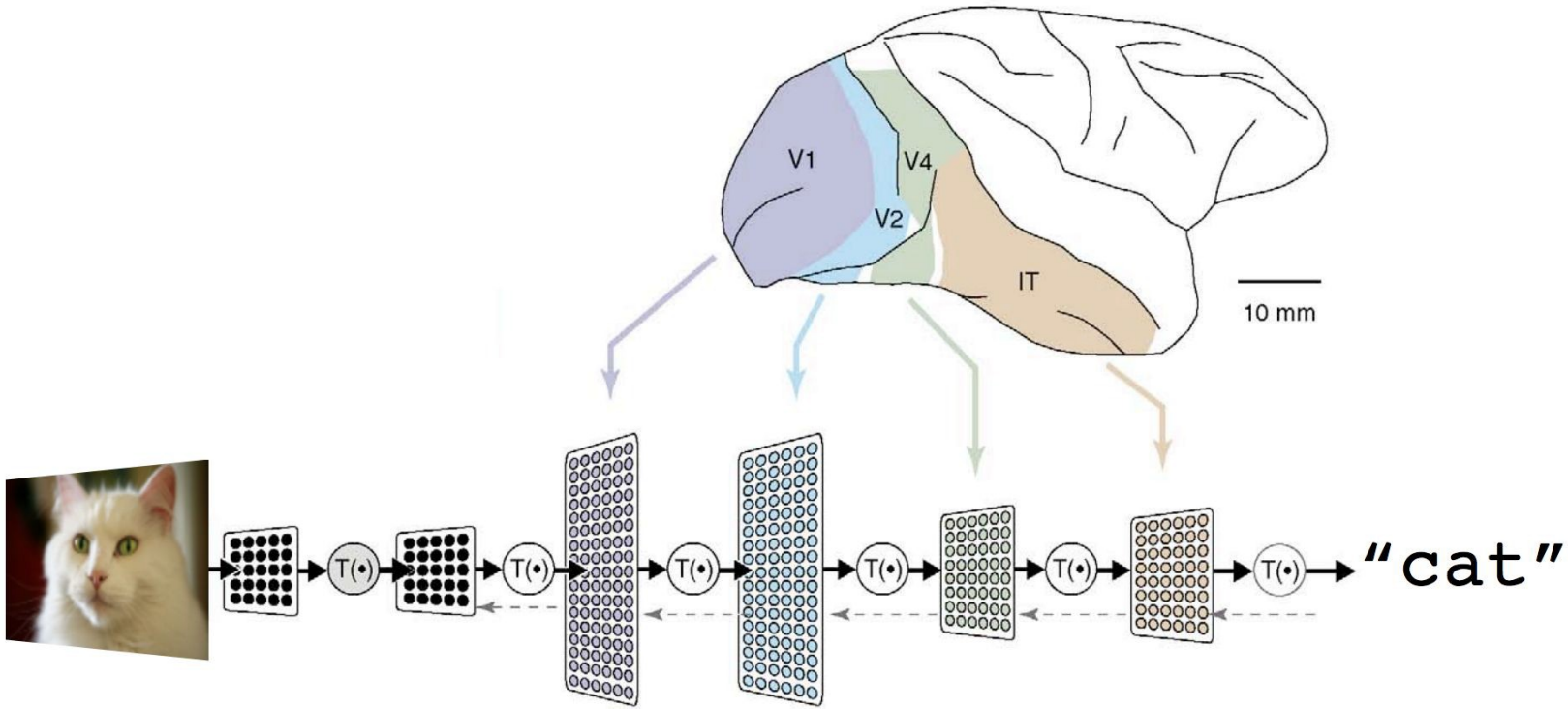


We could recognize characters… but can we see as well as a cat?

By David Corby Edited by: Arad (Image:Kittyplya03042006.JPG) [GFDL (http://www.gnu.org/copyleft/fdl.html), CC-BY-SA-3.0 (http://creativecommons.org/licenses/by-sa/3.0/) or CC BY 2.5 (https://creativecommons.org/licenses/by/2.5)], via Wikimedia Commons

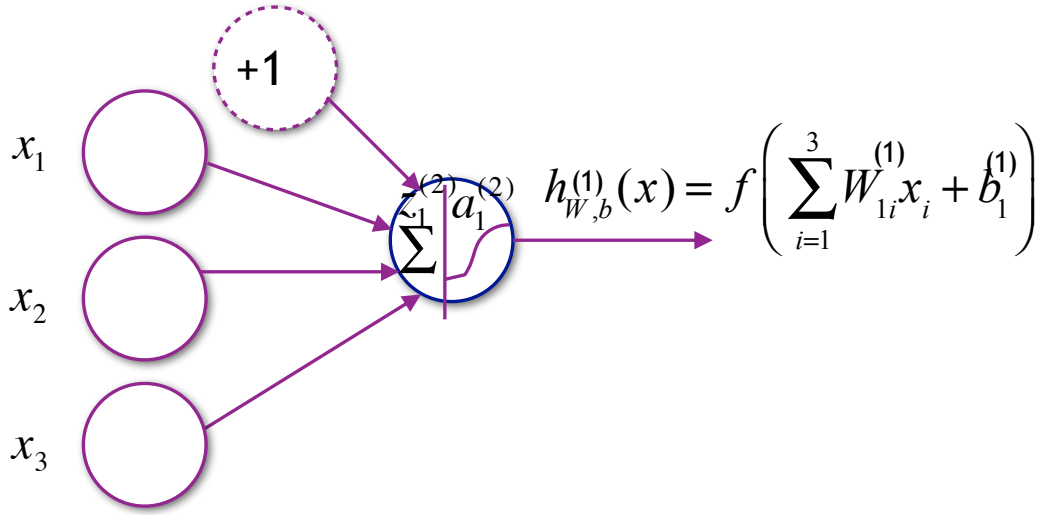
Biological Inspiration

- Processing in the brain uses multi-layer processing



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)
  - $\boldsymbol{x} = (x_1, \ldots, x_N)$ = input to the layer
  - W = weight vector
  - b = bias

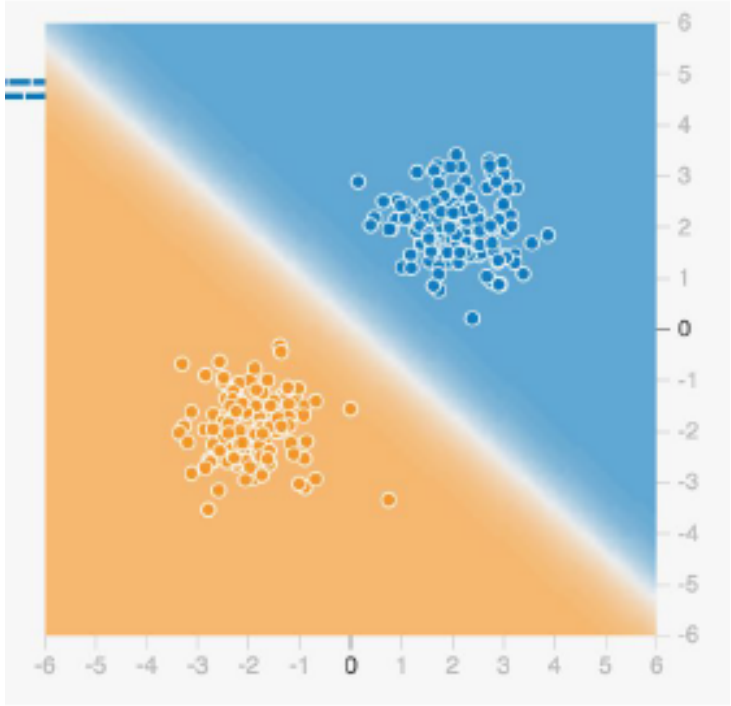


Linear Feature

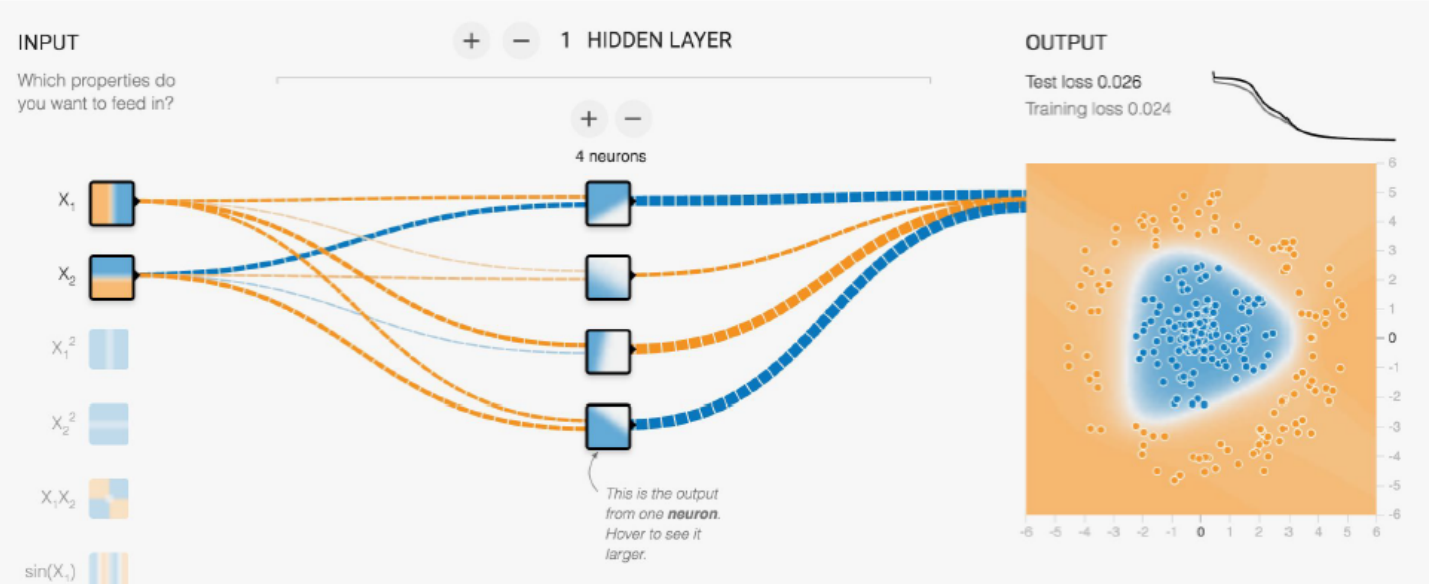
- Suppose activation is a hard threshold:

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

- Then, hidden unit divides input space into two half spaces
- Linearly separated
- Each unit can learn a linear feature
  - Classifies its input by being in a half space
- Shape of the feature defined by weight W



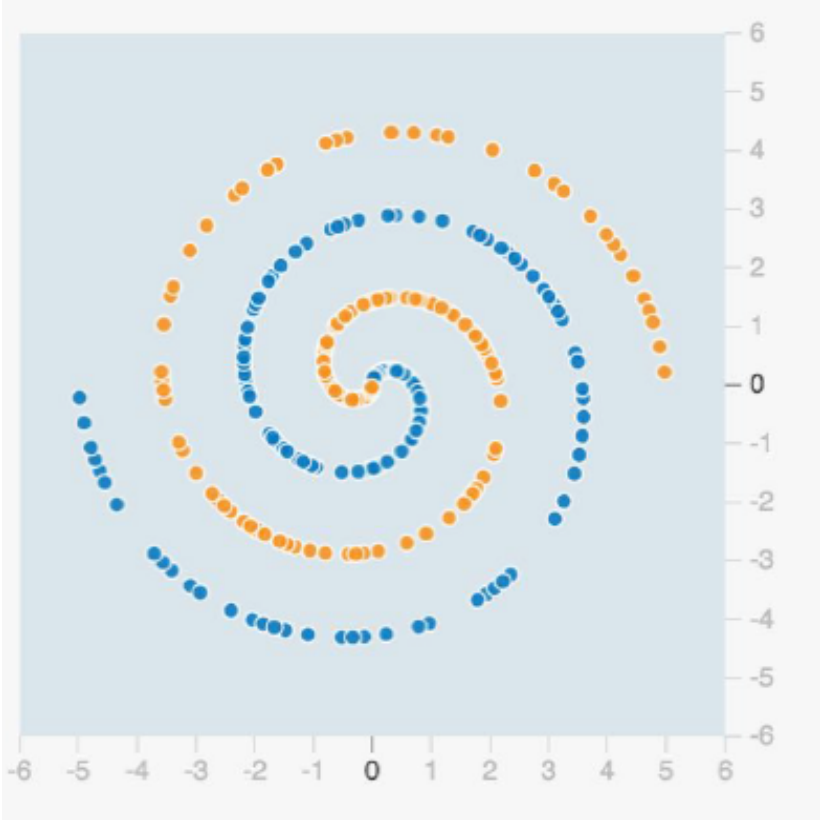
Classifying a Nonlinear Region



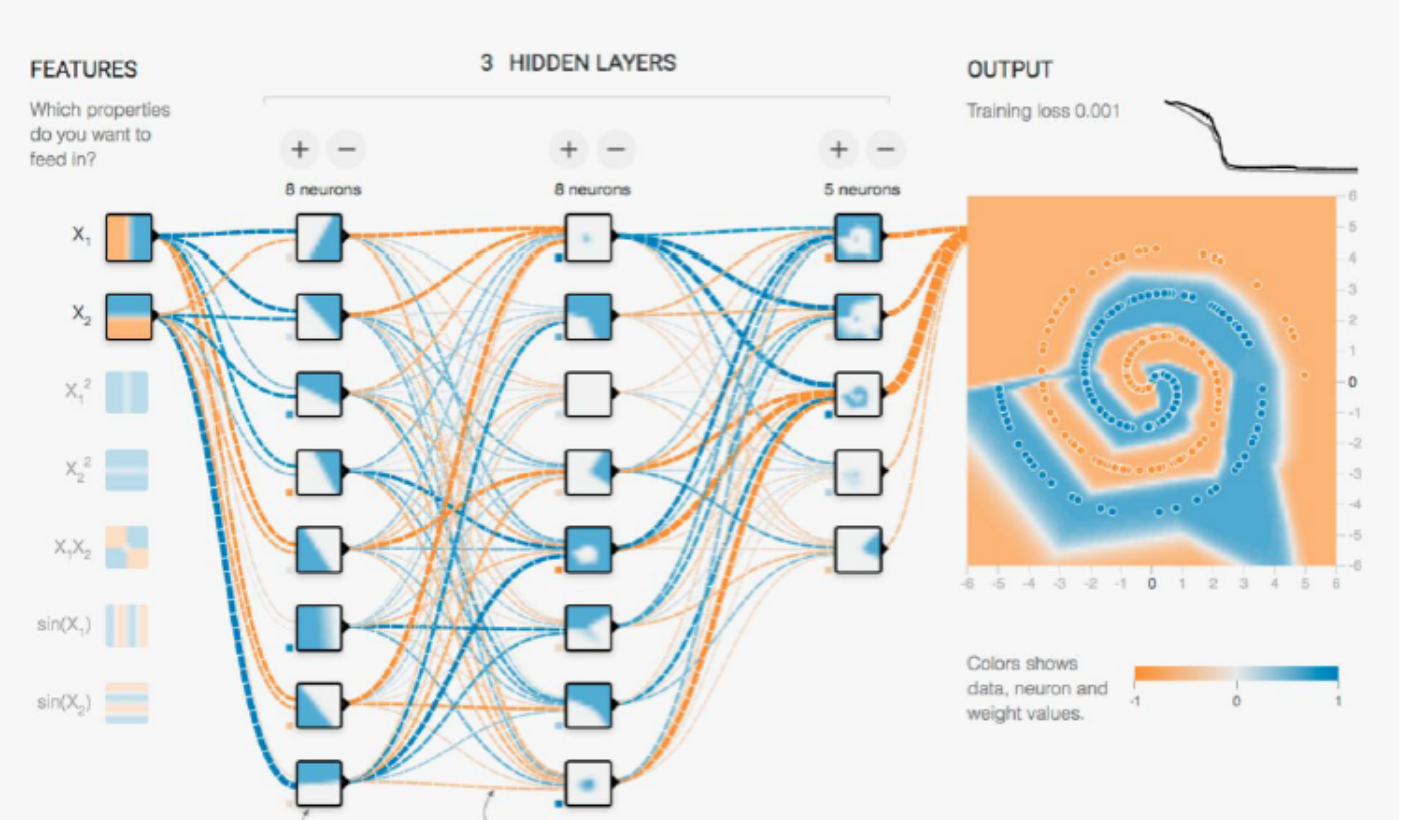
- Nonlinear regions
- Build from linear regions

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

What about a More Complicated Region?



Use Multiple Layers

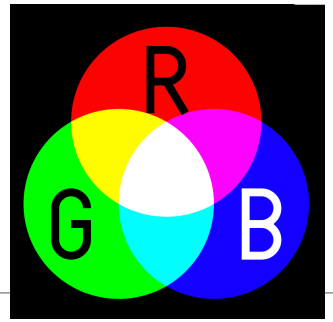


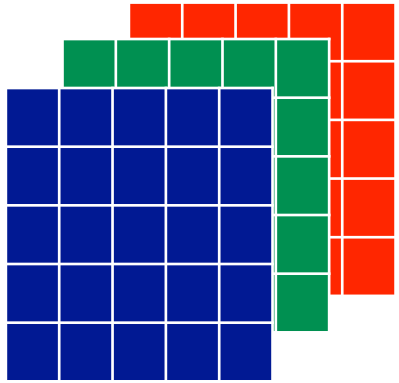
- More hidden layers
- Hierarchies of features
- Generate very complex shapes

"distributed representation" We can learn the different parts of a concept: Brown dog, brown horse, brown cat, white dog, white horse, white cat

We learn brown or white and cat, horse, dog, cat and then combine them

Can you Classify This?

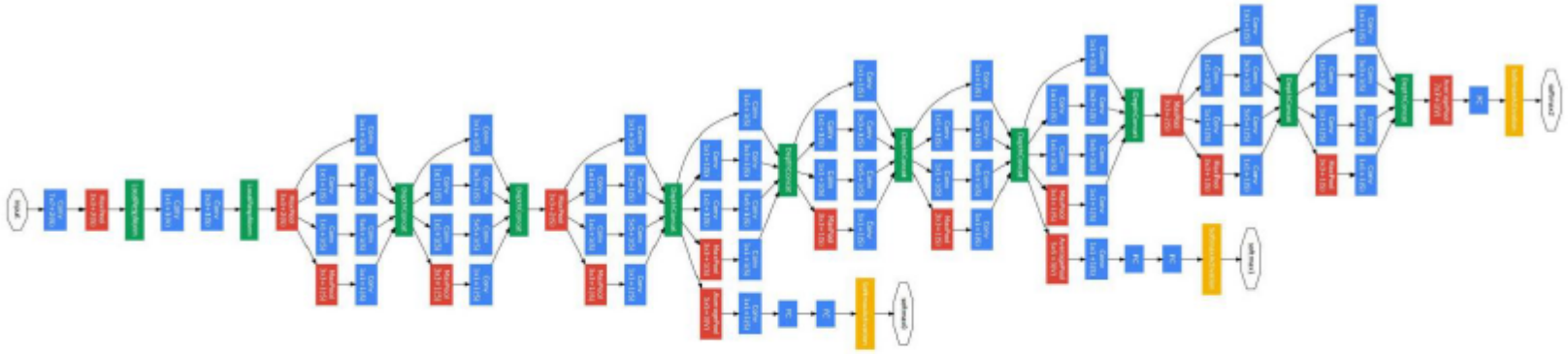




We want to add more layers, but more layers takes longer to train…

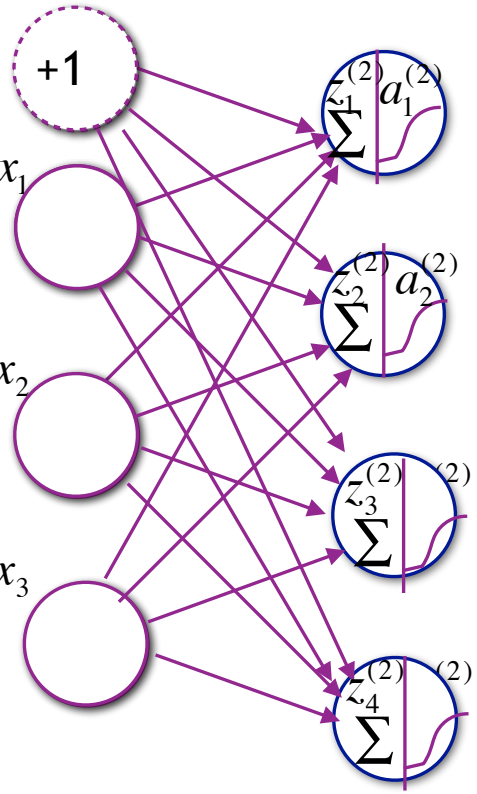
How can we add more layers without overfitting?

Build a Deep Neural Network



From: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations, Honglak Lee et al.

Convolutional Neural Network

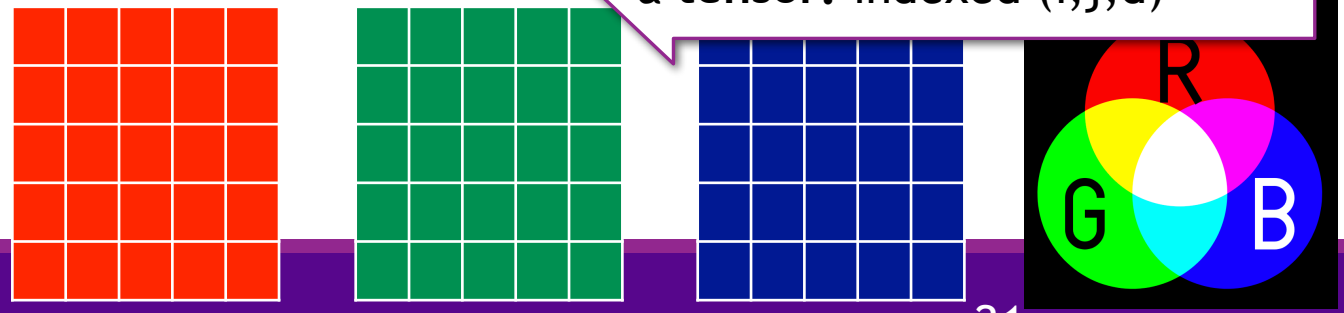


- Type of feed forward neural network
- CNN's (convolutional neural network's) have different types of layers

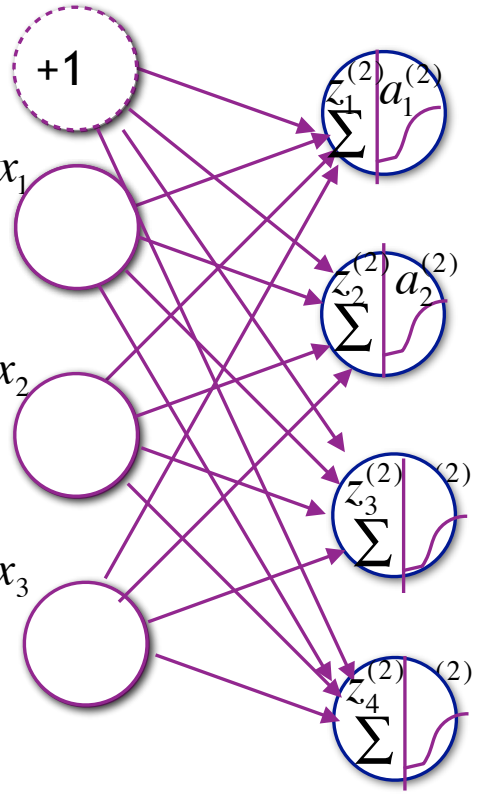
  Now there are ***three*** types of ***layers*** that might be used
  - **convolutional layer (cross-correlation - in CS we don't flip)**
  - **pooling layer**
  - **fully connected layer (the same as regular neural networks)**
- For most layers, we are not flattening the input to turn it into a vector as we previously did. Instead the input will be height ✖ width ✖ depth. Pictures have height, width and depth (e.g. depth is color: r, b, g ➡ depth = 3)
- The output is a 3D layer: height ✖ width ✖ depth called an **activation map**

A "high-order" matrix is called a **tensor:** indexed (i,j,d)

Preserve spacial structure



Convolutional Neural Network



- Type of feed forward neural network
- CNN's (convolutional neural network's) have different types of layers

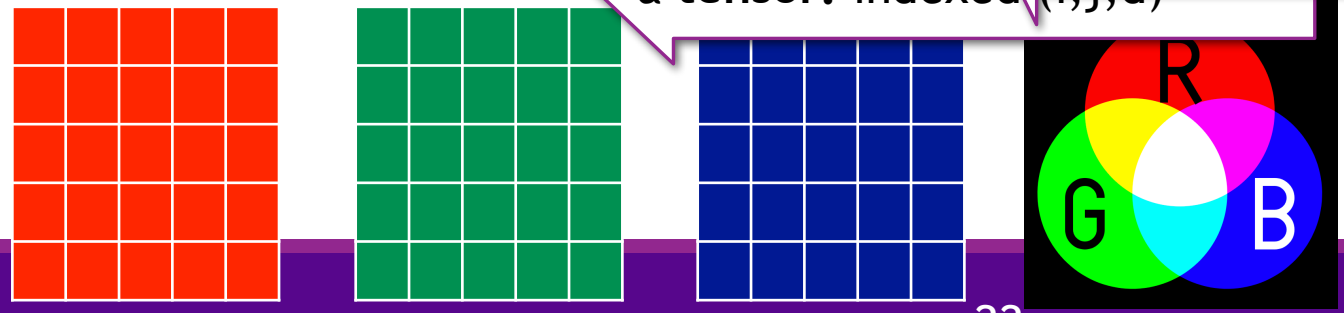
  Now there are ***three*** types of ***layers*** that might be used

  - **convolutional layer (cross-correlation - in CS we don't flip)**
  - **pooling layer**
  - **fully connected layer (the same as regular neural networks)**
- For most layers, we are not flattening the input to turn it into a v previously did. Instead the input will be height ✖ width ✖ depth. height, width and depth (e.g. depth is color: r, b, g ➡ depth = 3)
- The output is a 3D layer: height ✖ width ✖ depth called an **activation map**

Now we are using the word **depth** in two ways....
1) depth of a layer and 2) depth of the network

A "high-order" m a **tensor:** indexed (i,j,d)

Preserve spacial structure



Often one bias term is added to a filter

Convolution Layer

This layer is doing most of the work

- The convolution layer is the "eyes" of the network
- The CONV layer's *parameters* are a set of "learnable" *filters*.

Feature detector that looks over the entire image for the pattern

- Instead of using all the nodes from the previous layer to compute a *node* in the next layer, apply a ***filter*** to a chunk of the image
- Each filter has width ✖ height ✖ depth - typically *small width* and *height* but the depth *must be* the same as the input volume
- Each filter creates 2-dimensional **activation map**

7	1	2	4	3	2
6	3	3	1	3	4
2	0	1	0	0	0
1	3	5	…		

1	0	-1
1	0	-1
1	0	-1

$W0$

9	-1	0	-1
0	…		

activation map

The matrix is called a **feature map** or **activation map**
Each position of the matrix corresponds to a neuron

Convolution Layer

This layer is doing most of the work

- The convol… …f the network
- T… …able" *filters*.
- …r to compute a *node* in the next
- de… …all width and height but the
- Each filte… …tion map

When applying a filter, you can think of the filter as finding a feature in part of its input (sort of like a magnifying glass passing over the entire input looking for a latent feature). The output is the activation map which is like a "map" that shows where that feature occurred. An activation map is also called a feature map or output volume.

Fea… Extracts features - large look… positive values in areas ima… most similar to the filter

Often one bias term is added to a filter

7	1	2	4	3	2
6	3	3	1	3	4
2	0	1	0	0	0
1	3	5	…		

1	0	-1
1	0	-1
1	0	-1

$W0$

9	-1	0	-1
0	…		

activation map

If the original image is 7x7 and the filter is 3x3 what is the size of the activation map?

a) 6x6
b) 5x5
c) 4x4
d) 3x3
e) None of these

Convolution Layer

This layer is doing most of the work

If the original image is 7x7 and the filter is 3x3 what is the size of the activation map?

a) 6x6
b) 5x5
c) 4x4
d) 3x3
e) None of these

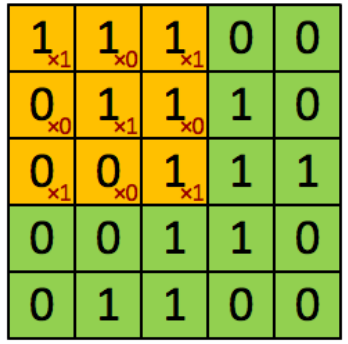
(7-3+1)x(7-3+1)

Convolution 2D Example

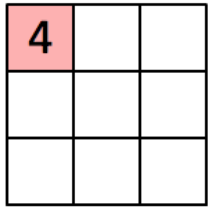
- Kernel

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

- Compute convolution in valid region



Image

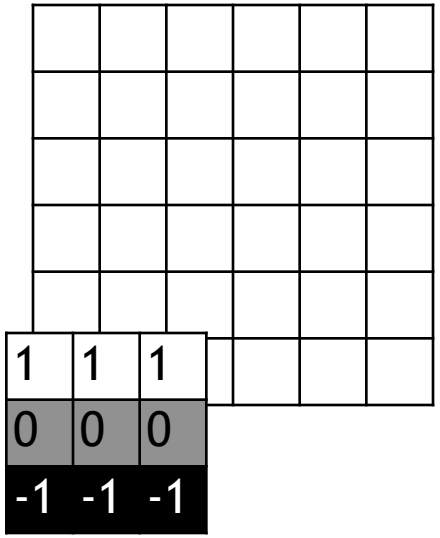


Convolved Feature

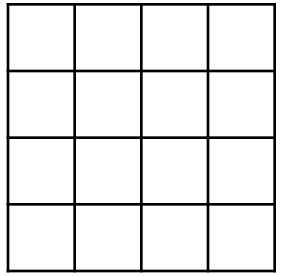
https://stats.stackexchange.com/questions/199702/1d-convolution-in-neural-networks

Edge Detection - using convolution operation

- Early layers of CNN might detect edges, later layers will use the features computed in early layers to detect parts of object, which then can be combined to produce the output
- Vertical edge detection



1	0	-1
1	0	-1
1	0	-1



- Horizontal edge detection

1	1	1
0	0	0
-1	-1	-1

- Vertical Sobel filter

1	0	-1
2	0	-2
1	0	-1

- Scharr filter

3	0	-3
10	0	-10
3	0	-3

Sorbel filter example

We can have the computer learn what an edge is and then find it all over the image

```
Gx = np.array([[1,0,-1],[2,0,-2],[1,0,-1]])  # Gradient operator in the x-direction
Gy = np.array([[1,2,1],[0,0,0],[-1,-2,-1]])  # Gradient operator in the y-direction
```

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

- Use scipy convolve2d function
- Note mode and shape

vertical Sobel filter

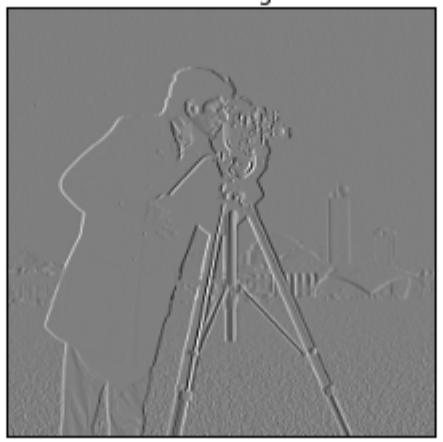
1	0	-1
2	0	-2
1	0	-1

Horizontal Sobel filter

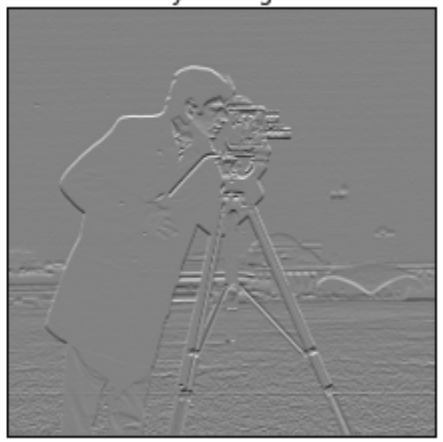
1	2	1
0	0	0
-1	-2	-1

Original

Gx * image



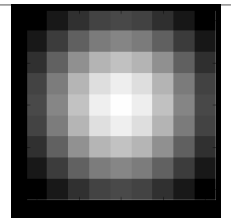
Gy * image



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

Using Convolutions for Averaging

- Filters can also take weighted averages
  - Several kernels: Gaussian, uniform, …



9x9 Gaussian blur kernel

- Convolution creates a blurred version of image

Padding when using Convolution

- After applying a convolution operation, our matrix "**shrunk**". We started with a 6✖6 volume and ended up with a 4✖4 volume after convolving with a 3✖3 filter (This will be a problem if the network is deep.)
- The size of the matrix after convolution for an n✖n matrix convolved with a F✖F filter(kernel) gives us a (W-F+1)✖(W-F+1) matrix
- Our matrix "shrinks" if the filter is F>1
- We can "pad" the matrix by adding extra rows and columns

0	0	0	0	0	0	0	0
0	1	0	-1				0
0	1	0	-1				0
0	1	0	-1				0
0							0
0							0
0							0
0	0	0	0	0	0	0	0

- The amount padded is P

So now our activation map is

(W + 2P -F+1)✖(W+2P-F+1)

1	0	-1
1	0	-1
1	0	-1

If the original image is 6x6
which we pad with P=1,
and the filter is 3x3
what is the size of the activation map?

a) 6x6
b) 5x5
c) 4x4
d) 3x3
e) 2x2
f) None of these

Padding when using Convolution

- The amount padded is P

So now our activation map is

$(W + 2P - F + 1) \times (W + 2P - F + 1)$

$(7 + 2*1 - 3 + 1) \times (7 + 2*1 - 3 + 1)$

If the original image is 7x7
which we pad with P=1,
and the filter is 3x3.
What is the size of the activation map?

a) 6x6
b) 5x5
c) 4x4
d) 3x3
e) 2x2
f) None of these

Striding when using Convolution

- We will call the **stride** S. I.e. How many pixels we jump over when convolving with the filter/kernel
- So now we get ((W + 2P - F)/S+1) ✖ ((W+2P-F)/S+1) (We can take the floor if it is a fraction…)

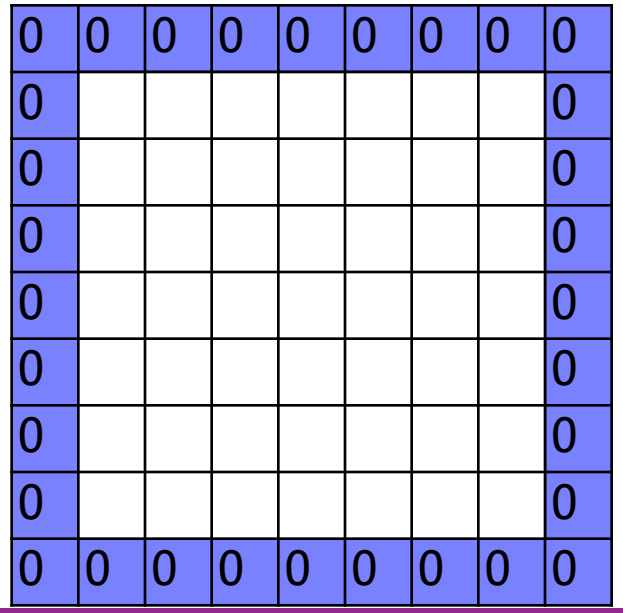
S = 2

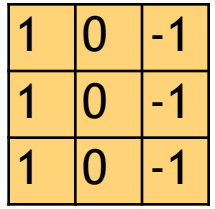
0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

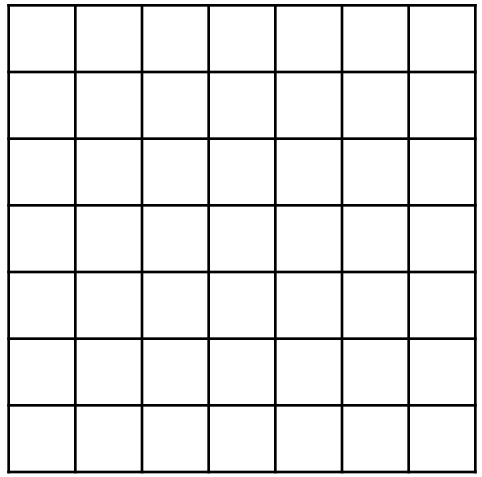
1	0	-1
1	0	-1
1	0	-1

Same convolution

- If the size of the matrix (with padding) is the same after a convolution, we call a same convolution
- P = (W*S-W+F-S)/2





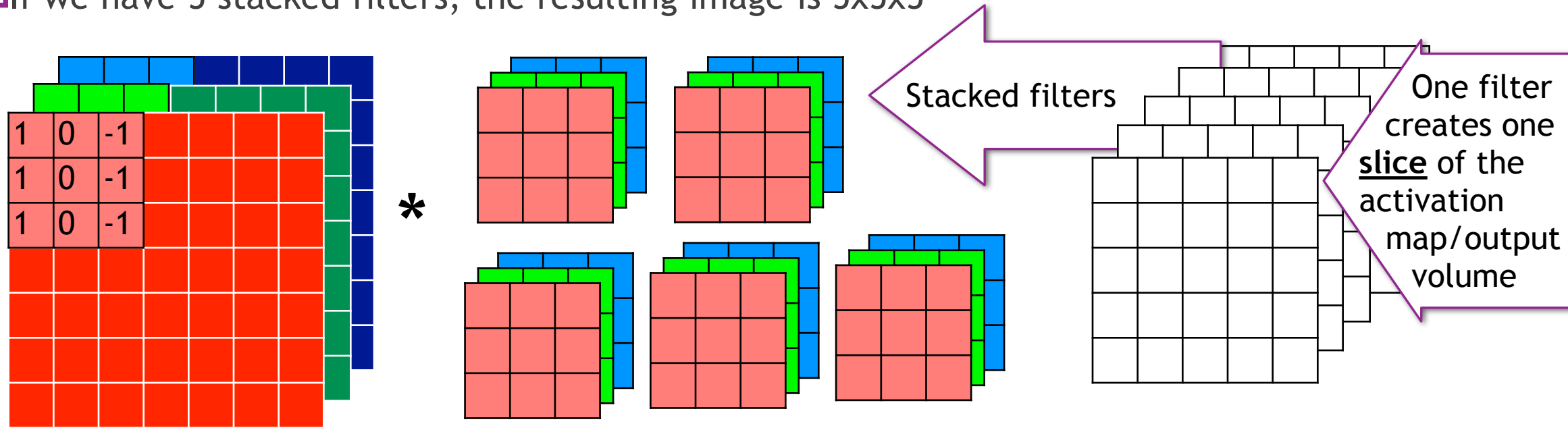


So now we get (W + 2P -F+1) ✖ (W+2P-F+1)

(7 + 2*1 -3+1) ✖ (7 + 2*1 -3+1)

Convolutions over volumes

- If we wanted to convolve a 3-dimensional image. We can create a 3-d filter
- Input a 7x7x3 image after we convolve it with a 3x3x3 filter we get a 5x5x1 image (this is 2D!)
- We can use multiple filters - thus detecting multiple features or edges
- If we have 5 stacked filters, the resulting image is 5x5x5



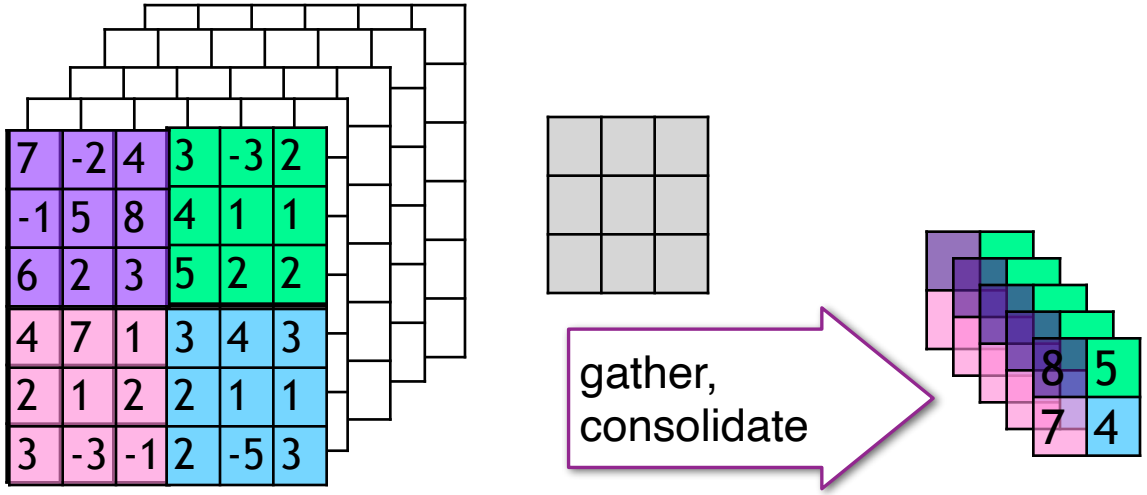
Example from Convolution Demo at:
https://cs231n.github.io/convolutional-networks/

Max Pooling layer - reduces the size

Most common is F=2, S=2, or F=3, S=2

- By reducing the size of the input, the computation speeds , and makes some features it detects more robust
- Each slice is independently operated on. It resizes (spatially) using the MAX operator
- If the feature occurs anywhere "keeps a high number" (works well and reduces computation)

Could use other functions e.g. average or min



- F=3, S=3, p=0 hyper parameters
- Output: 2x2x5

How many parameters do we need to train for a pooling filter of size 3x3?

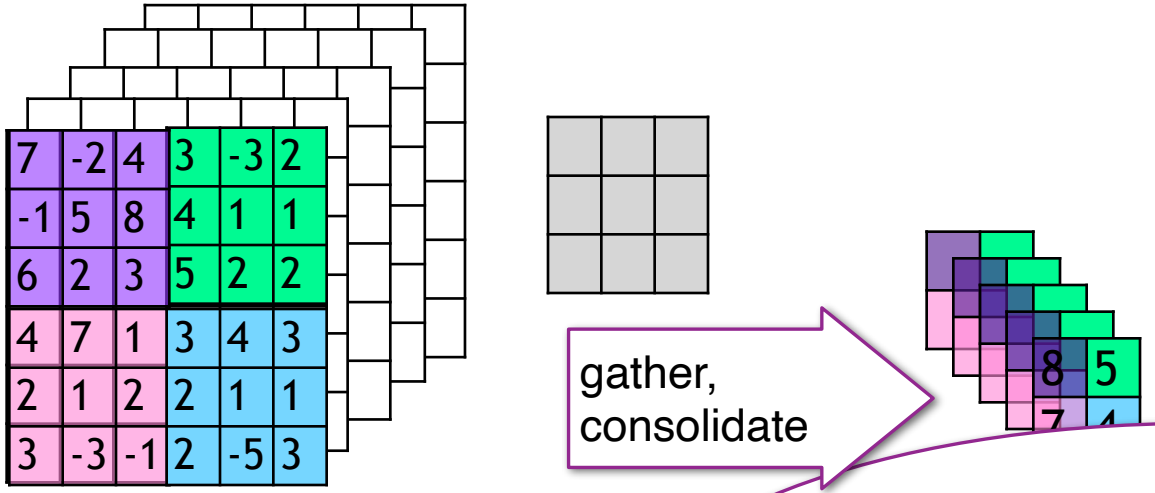
a) 0
b) 1
c) 3
d) 9
e) None of these

Max Pooling layer - reduces the size

Most common is F=2, S=2, or F=3, S=2

- By reducing the size of the input, the computation speeds , and makes some features it detects more robust
- Each slice is independently operated on. It resizes (spatially) using the MAX operator
- If the feature occurs anywhere "keeps a high number" (works well and reduces computation)

Could use other functions e.g. average or min



How many parameters do we need to train for a pooling filter of size 3x3?
a) 0
b) 1

- F=3, S=3, p=0 hyper parameters
- Output: 2x2x5

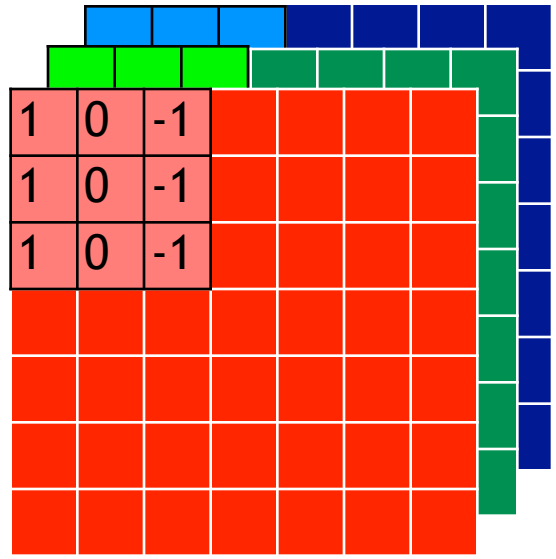
We lose information in this step. It remembers only the highest correlation of the feature in the nearby area. Thus allowing flexibility is where a feature is. Additional benefits are less storage and processing.

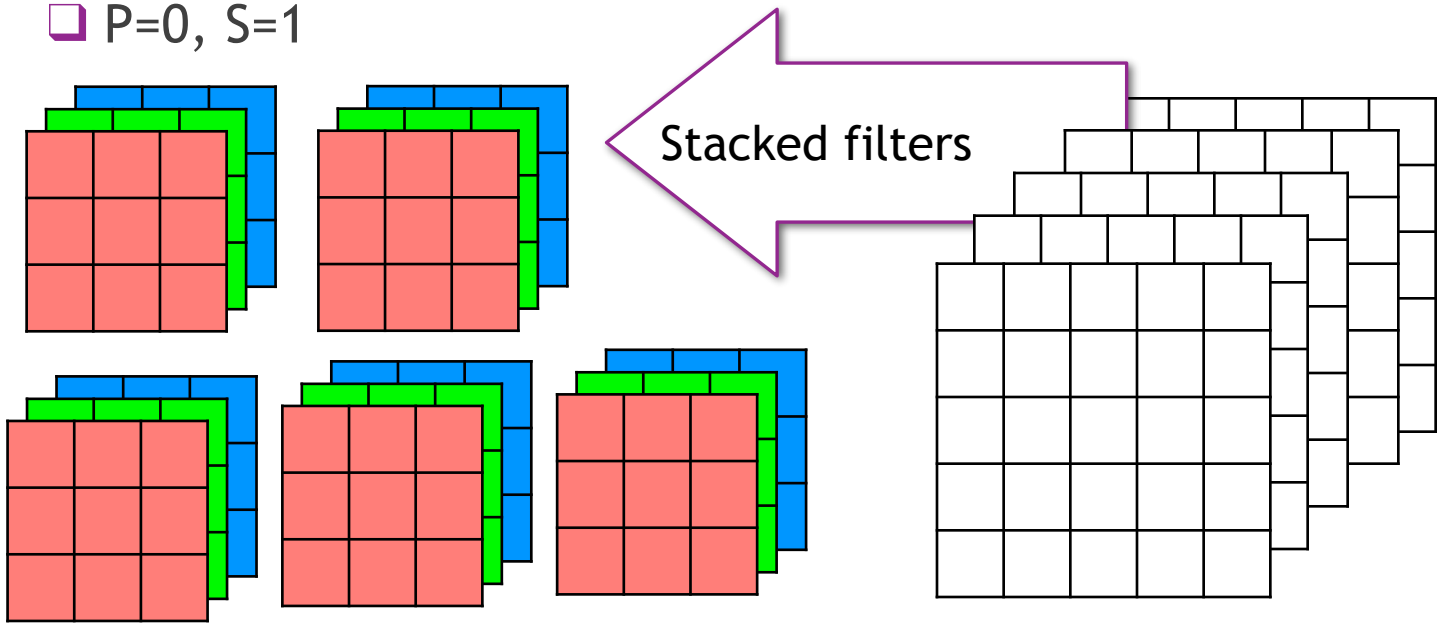
One Layer of a Convolutional Network

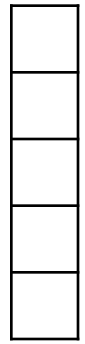
- After we convolve the image, we can add a bias and then apply an activation function
- Instead of the sigmoid, it is more common to apply ReLU: max(0,x)

- input image: 7x7x3
- 5 filters: 3x3x3
- resulting image: 5x5x5

- Add b (bias) 5x1
- Add ReLu
- P=0, S=1







One Layer of a Convolu

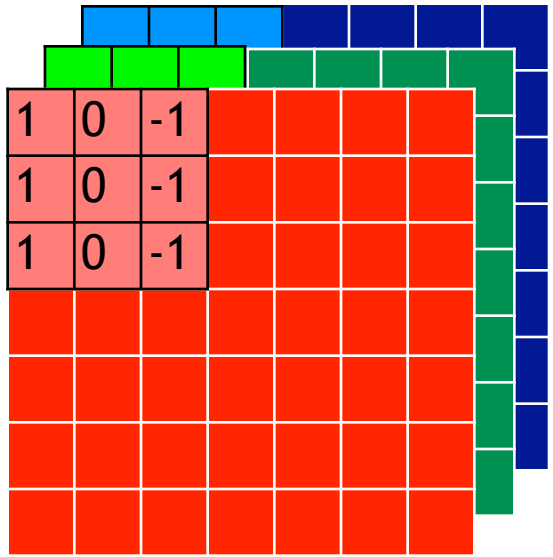
- After we convolve the image, we can add a bias and then
- Instead of the sigmoid, it is more common to apply ReLU: max(0,x)

Notice that the number of parameters *doesn't change* if the input increases. This helps prevent overfitting

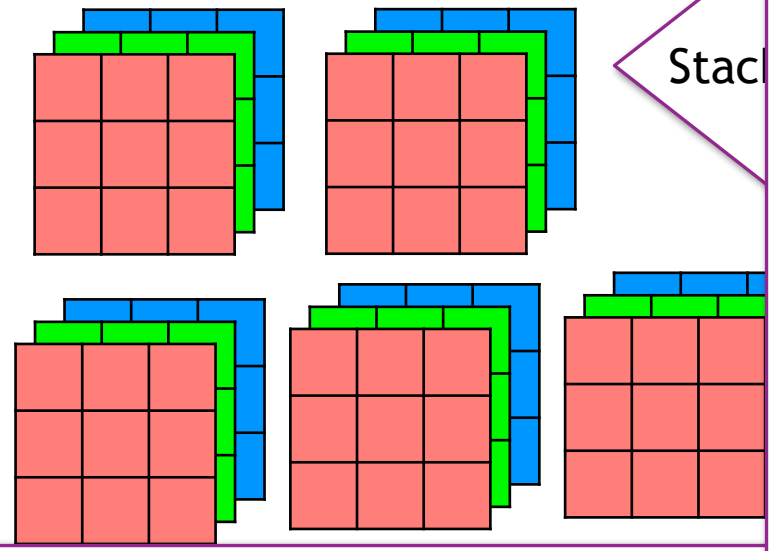
b

- input image: 7x7x3
- 5 filters: 3x3x3
- resulting image: 5x5x5

- Add b (bias) 5x1
- Add ReLu
- P=0, S=1



Stac



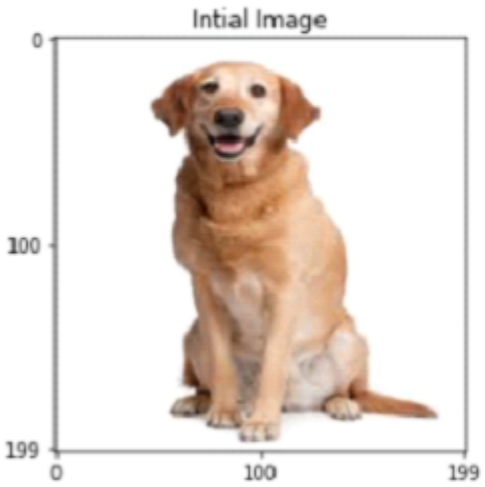
number of parameters: (3x3x3x5)+5

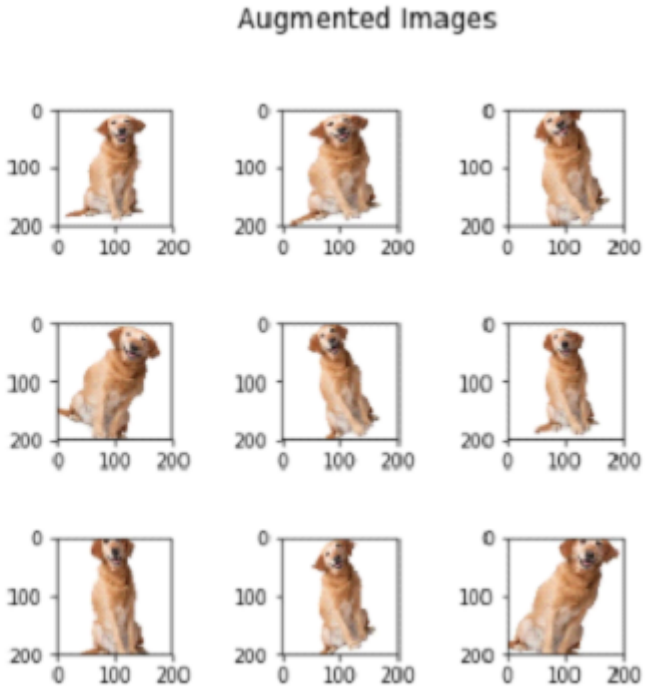
How many trainable parameters for this layer?

a) 0
b) 27
c) 135
d) 140
e) None of these

Data Augmentation

- Often our training dataset is smaller than we'd like
- Idea: Augment the dataset
- Generate additional images by flipping, shifting, scaling, rotating, and cropping





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 netwo
- Can be used for:
  - Making predictions
  - Building new, powerful networks

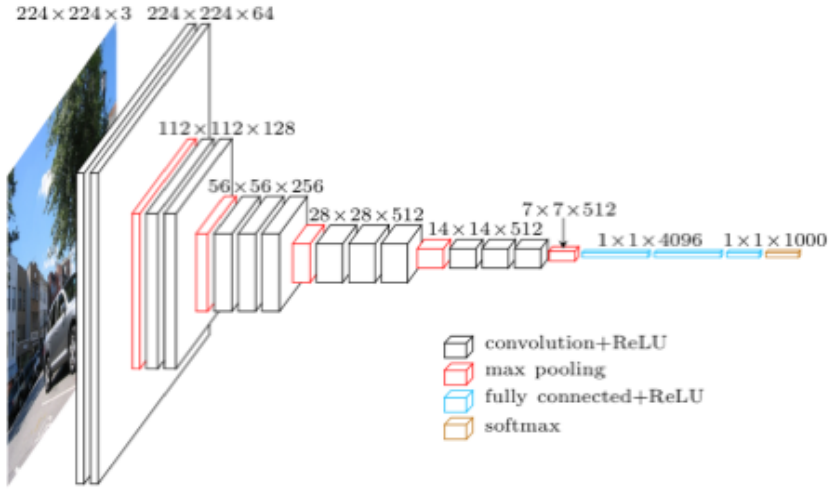
Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953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

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 Recog
arXiv technical report, 2014

Loading the Pre-Trained Network

```
# Load appropriate packages
from tensorflow.keras.applications.xception import Xception
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing import image
```

```
model = Xception(weights='imagenet',input_shape=(299,299,3))
```

- Load the packages
- Create the model
  - Downloads the h5 file
  - First time, may be a while (500 MB file)

Display the Network

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

Conv + relu

Layer (type)	Output Shape	Param #
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

```
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
```

- Very deep: 16 layers (Do not count pooling 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)
```

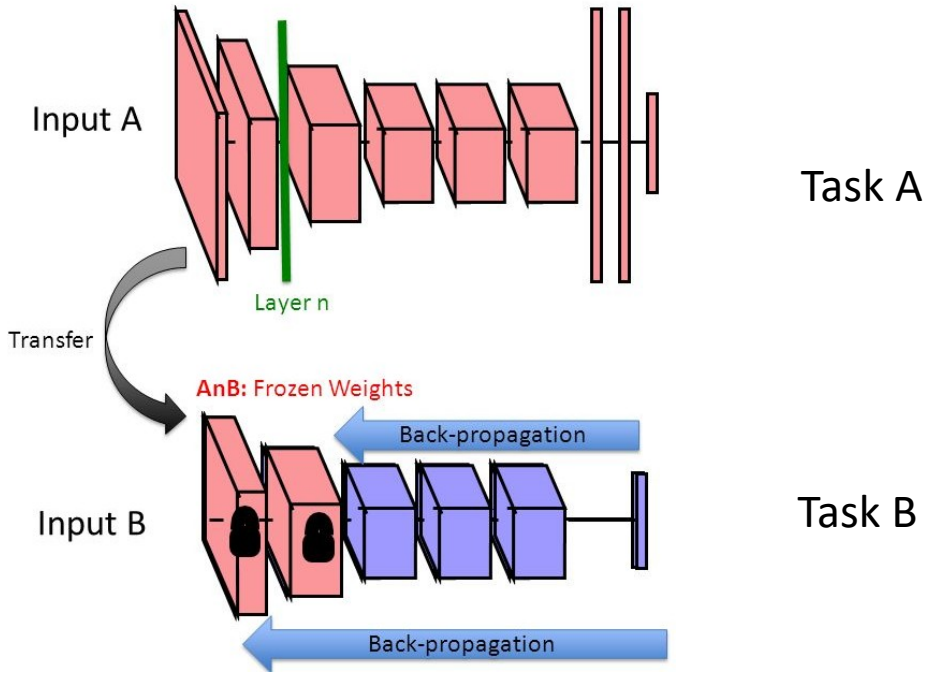
```
preds = model.predict(x)
preds_decoded = decode_predictions(preds, top=3)
```

	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

Transfer Learning

- For image classification or other applications, training from scratch takes tremendous resources
- Instead, can refine the VGG or other well trained networks for Task A (e.g. imagenet)
- Build a new network for Task B
  - Use early layers from first network
  - Freeze those parameters
  - Only train small number of parameters at end
- Greatly reduces number of parameters for Task B
- Can be trained on ~1000 images



Typical convolutional neural network architecture:

From https://cs231n.github.io/convolutional-networks/

“The most common form of a ConvNet architecture stacks a few CONV-RELU layers, follows them with POOL layers, and repeats this pattern until the image has been merged spatially to a small size. At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores. In other words, the most common ConvNet architecture follows the pattern:

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC”
```

*Sometimes people combine conv+relu into a single layer and refer to it as a convolutional layer

On your own watch

https://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html