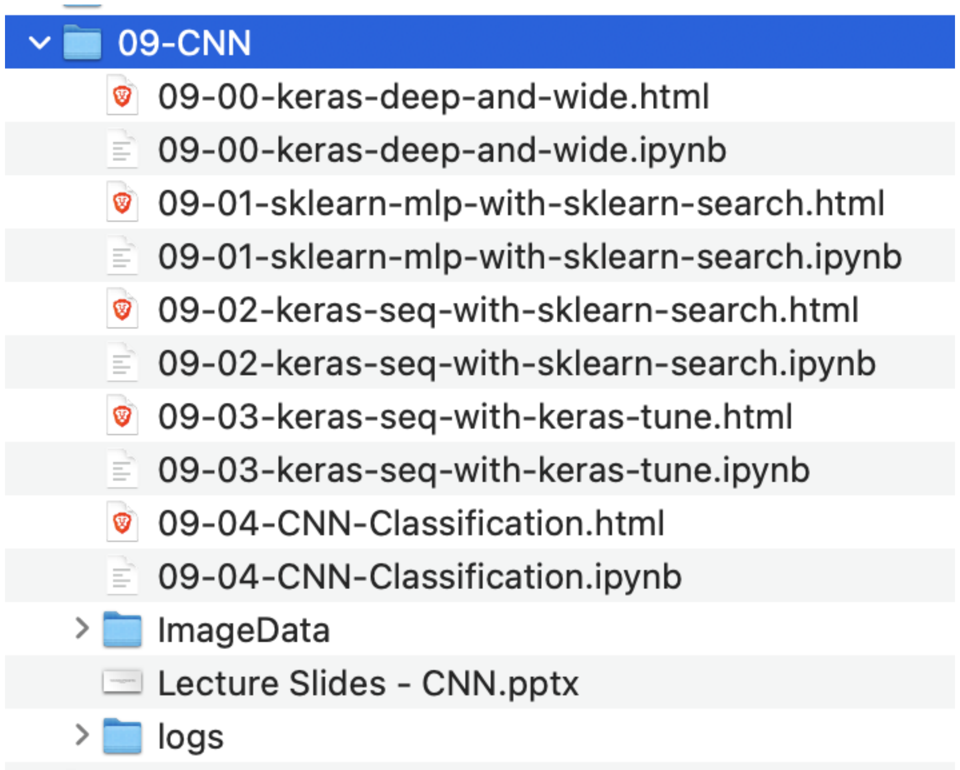
Convolutional Neural Nets

Tim Smith, PhD

Class09

- Review DNN
  - Demonstrate various deep and wide modeling techniques
- Discuss and elaborate on hyperparameter tuning
  - Sklearn model with Sklearn tuning
  - Keras model with Sklearn tuning
  - Keras model with Keras tuning
- Introduce the concept of convolutional neural networks
  - Build an image recognition model using CNN
- Demonstrate debugging models using Tensorflow

For the first part of the lecture:



Download the files from the following link…

https://usf.box.com/s/7bn9cox3sqyyw32lqqps2cb4nmcq0buv

Convolutional Neural Networks

- "Discrete convolution": *a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other (Wikipedia)*
- Input data: **images (usually)**
- Used in
  - Image search
  - Image classification
  - Self-driving cars

Review on Images

- If black and white, there is 1 channel
- If color, there are 3 channels: Red, Green, Blue (RGB)

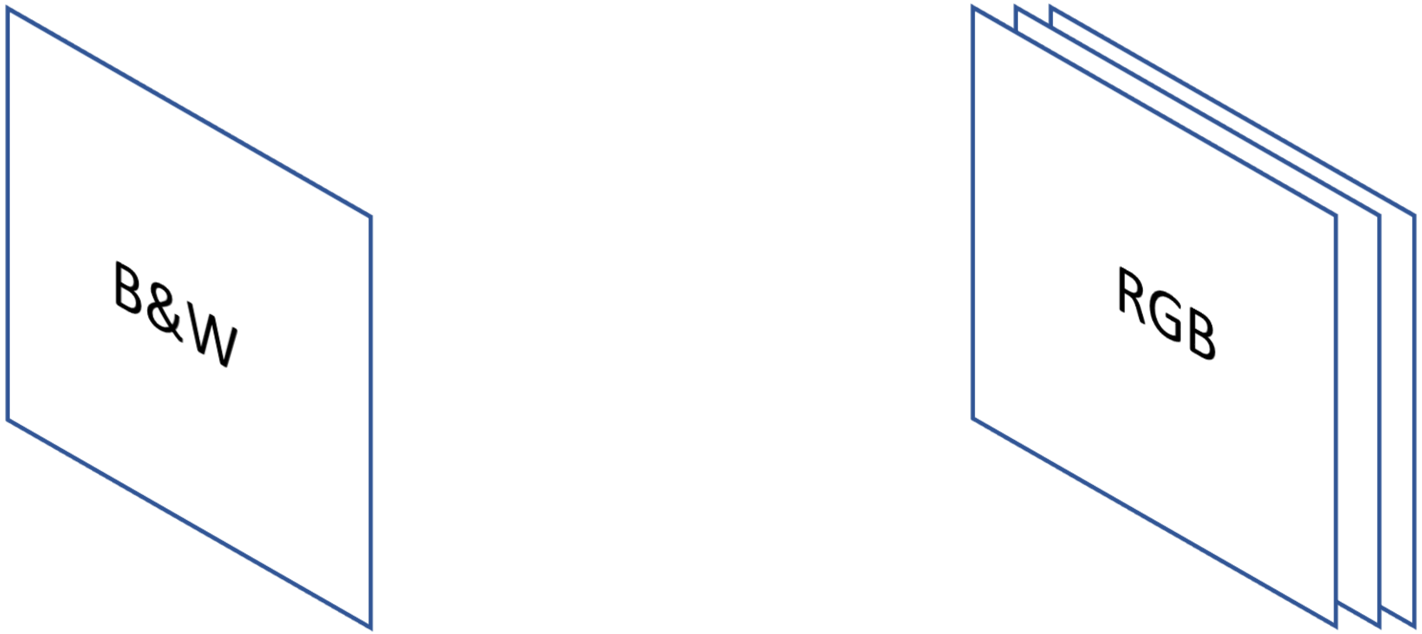
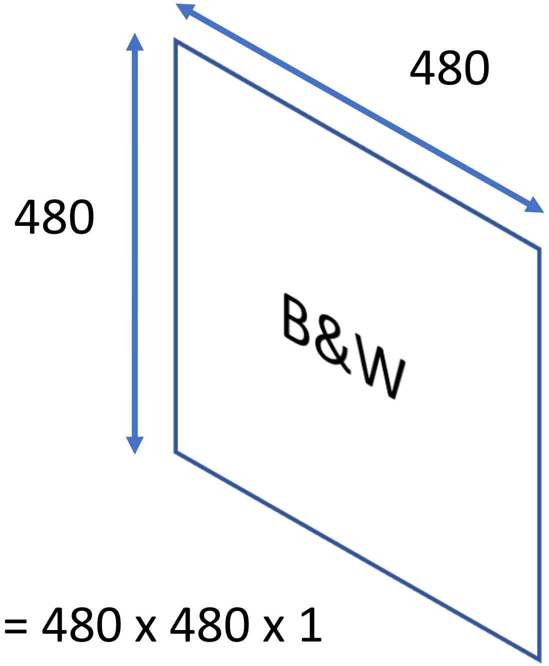
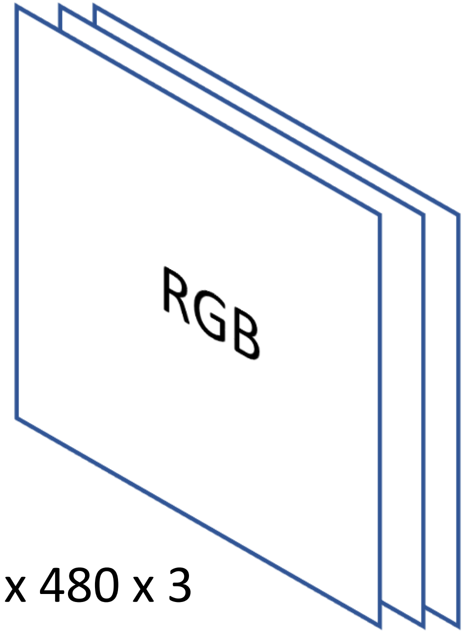


Image Data

- Image size is measured in pixels.
- Ex: 480 x 480 means:



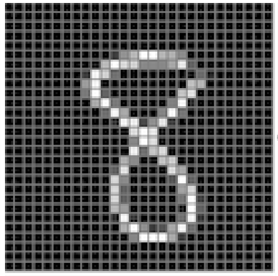
Size = 480 x 480 x 1



Size = 480 x 480 x 3

Image Data

- Each pixel has a value between 0-255



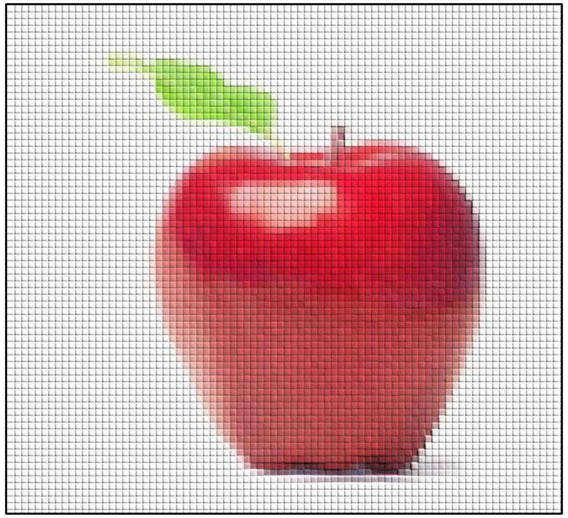
28 x 28
784 pixels

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	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	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	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	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	4	62	146	182	254	254	181	176	139	15	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	34	186	253	217	208	136	136	136	166	232	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	61	242	208	111	3	0	0	0	0	0	18	32	107	43	0	0	0	0	0	0
0	0	0	0	0	0	0	0	156	242	23	0	0	0	0	0	0	0	13	191	181	6	0	0	0	0	0	0
0	0	0	0	0	0	0	0	121	255	98	3	0	0	0	0	0	8	194	225	12	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	169	253	120	3	0	0	0	0	128	247	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	3	111	244	169	19	0	14	131	249	117	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	59	241	235	72	142	229	66	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	25	218	254	231	36	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	133	253	221	33	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	19	237	111	196	217	19	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	138	0	23	193	204	18	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	96	224	0	0	0	25	218	169	3	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	138	0	0	0	0	86	253	99	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	3	162	214	11	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	215	97	0	0	0	0	0	118	253	68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	185	157	0	0	0	0	0	40	254	98	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	50	244	61	0	0	0	0	112	244	58	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	174	251	142	59	83	167	244	111	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	6	133	253	253	253	169	61	3	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	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
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

Image Data

- Let's think of a 1M (megapixel) image
- 1 megapixel = 1,000,000 pixels (1,000 x 1,000)

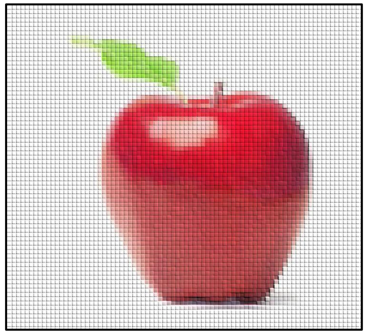
1,000



1,000

Resize

100

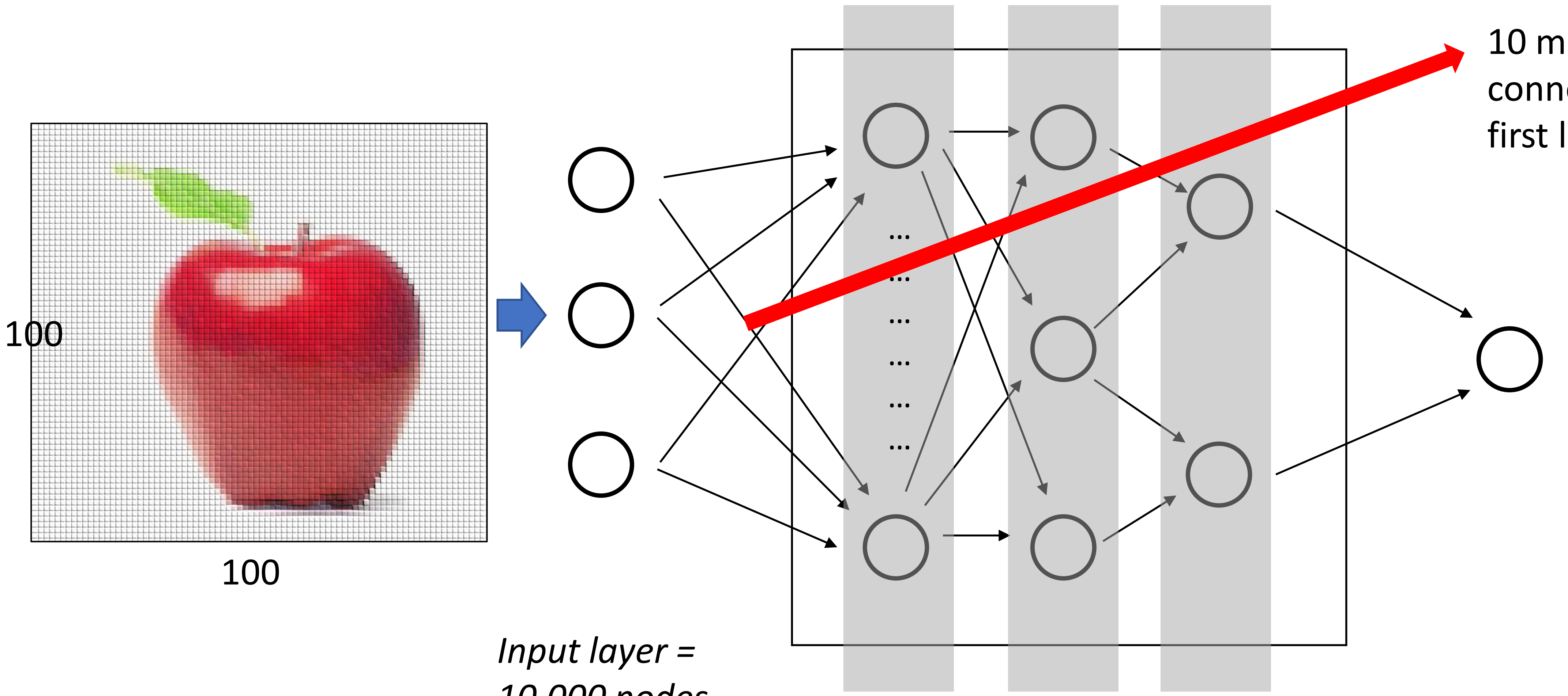


100

10,000 pixels

10K image

Image Data



Input layer = 10,000 nodes

Layer 1 = 1,000 nodes

Problem???

10 million connections in the first layer alone!

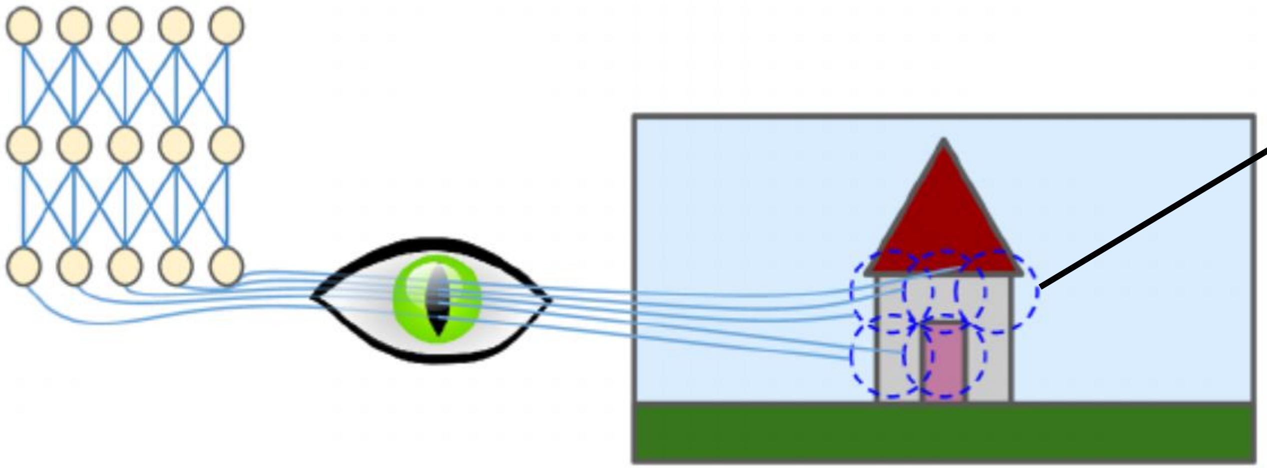
Solution:

- **STEP 1 (convolution):** Extract features from an image:
  - Horizontal lines,
  - Vertical lines,
  - Curvature,
  - Colors,
  - Etc.
- **STEP 2 (pooling):** Reduce the variables (without losing information)

- **GET HELP FROM HUMAN VISUAL CORTEX**

Convolutional Neural Networks

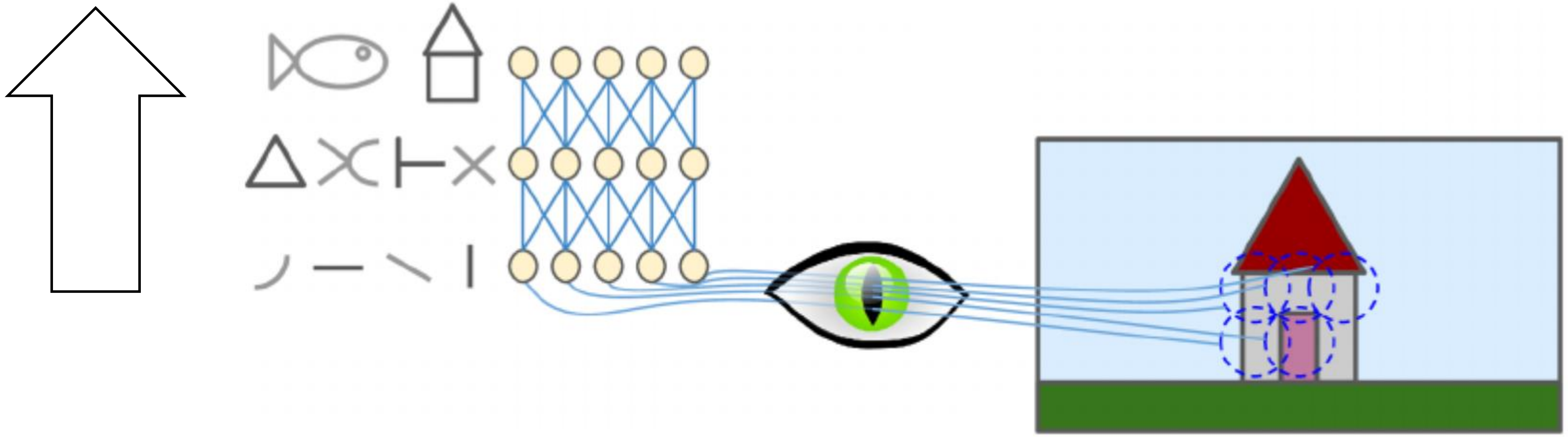
- The visual cortex is made up of neurons
- Each neuron has a specific receptive field (and do not necessarily see the things outside of that field)



Receptive field of a neuron

Convolutional Neural Networks

- Further, some neurons react only to horizontal lines
- Some react to different orientations
- Higher level neurons make sense of the pattern detected at lower levels



Convolutional Layer

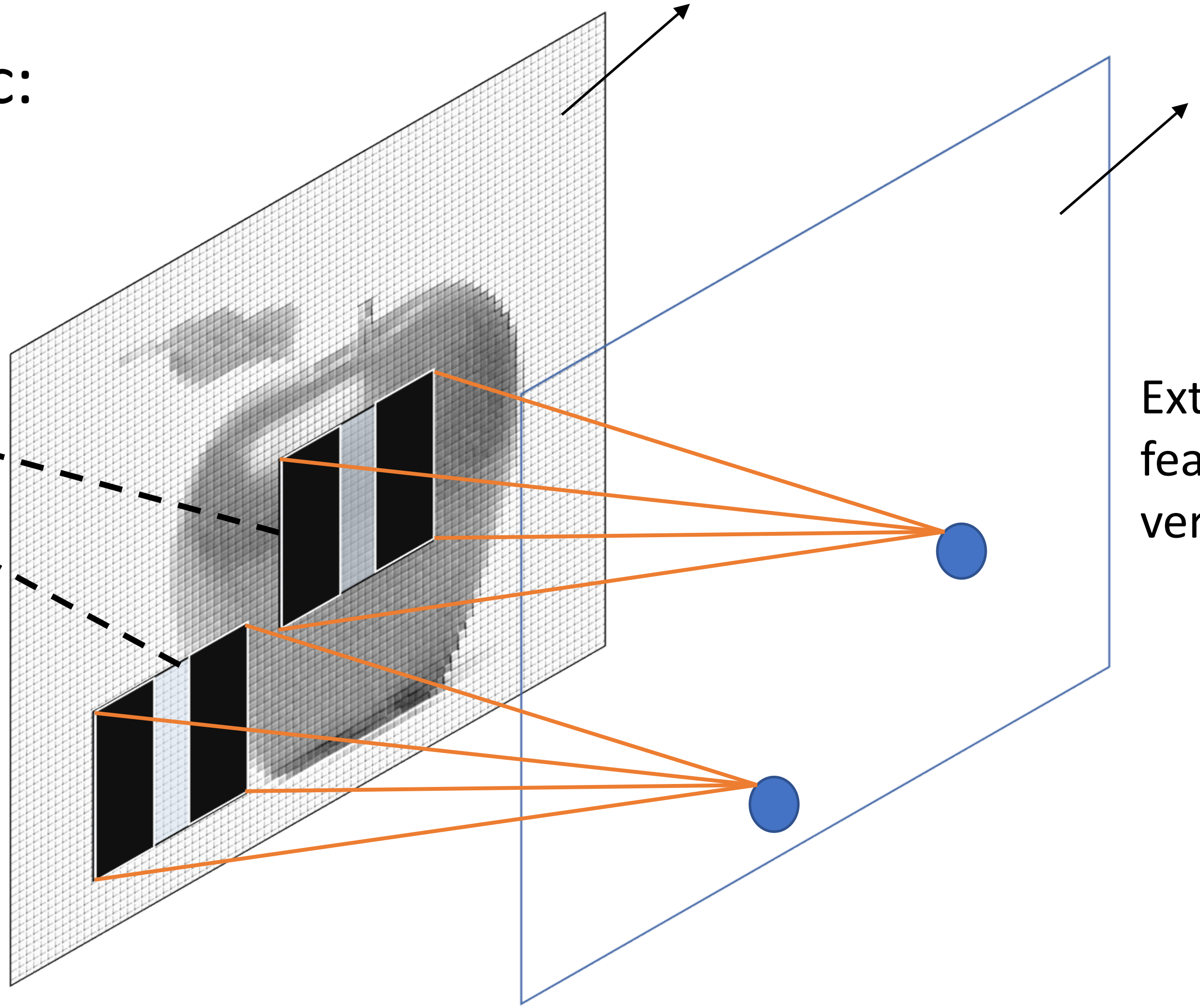
- Apply the same logic:

"Receptive field"
Also called:

- "**Filter**"
- "Kernel"

Grayscale (1 channel)

Feature Map 1

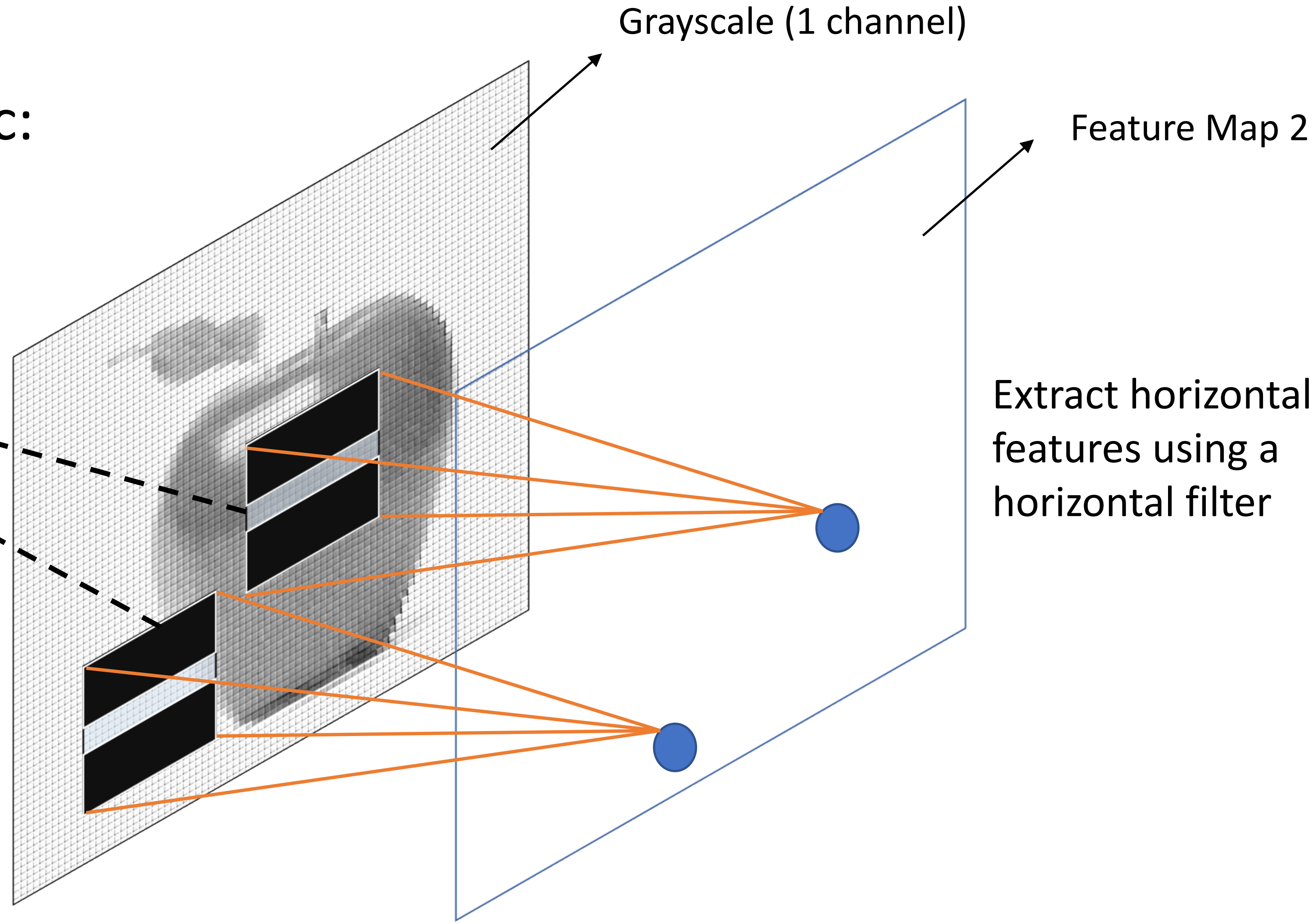


Extract vertical features using a vertical filter

Convolutional Layer

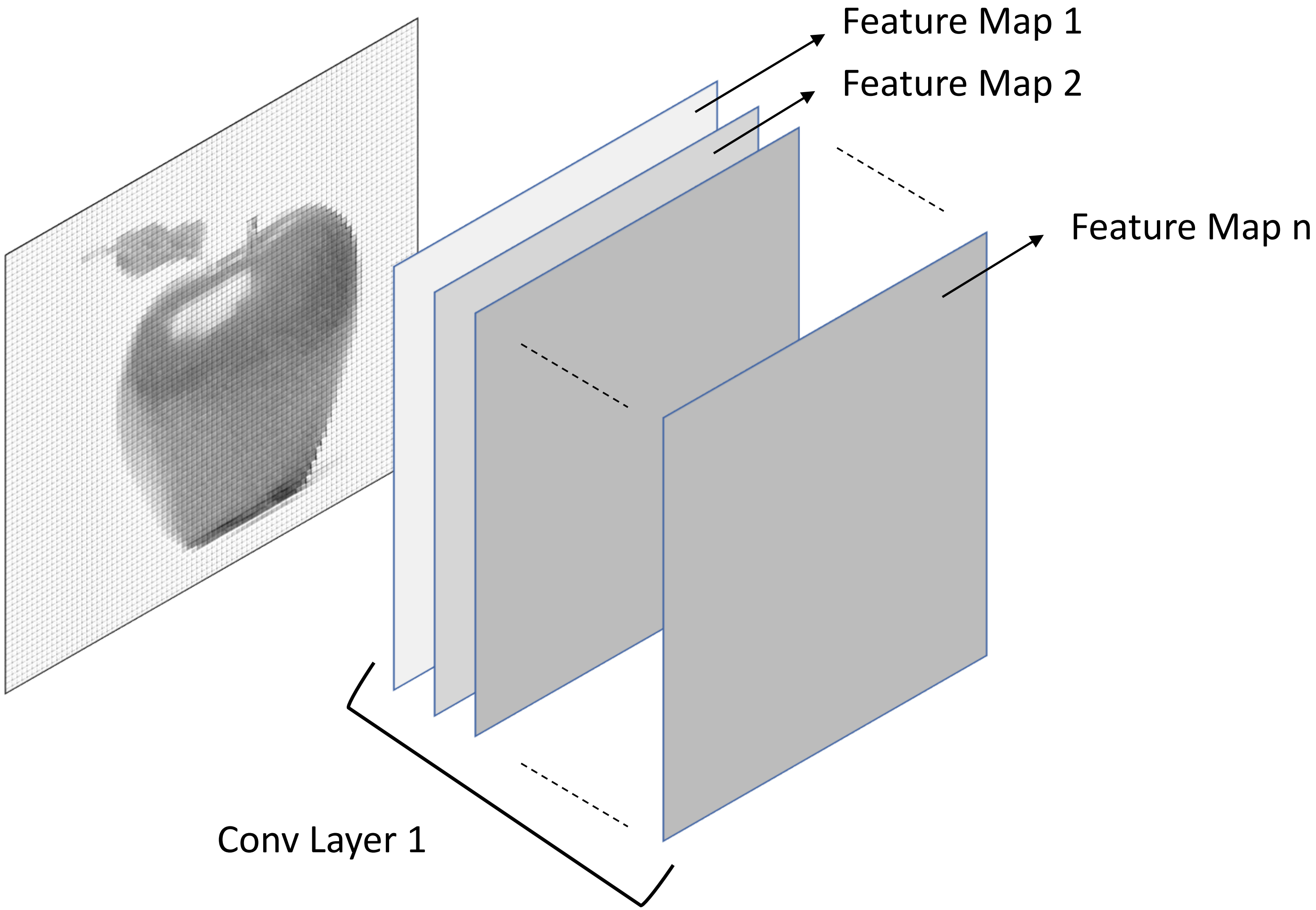
- Apply the same logic:

“Receptive field”
Also called:
- “**Filter**”
- “Kernel”

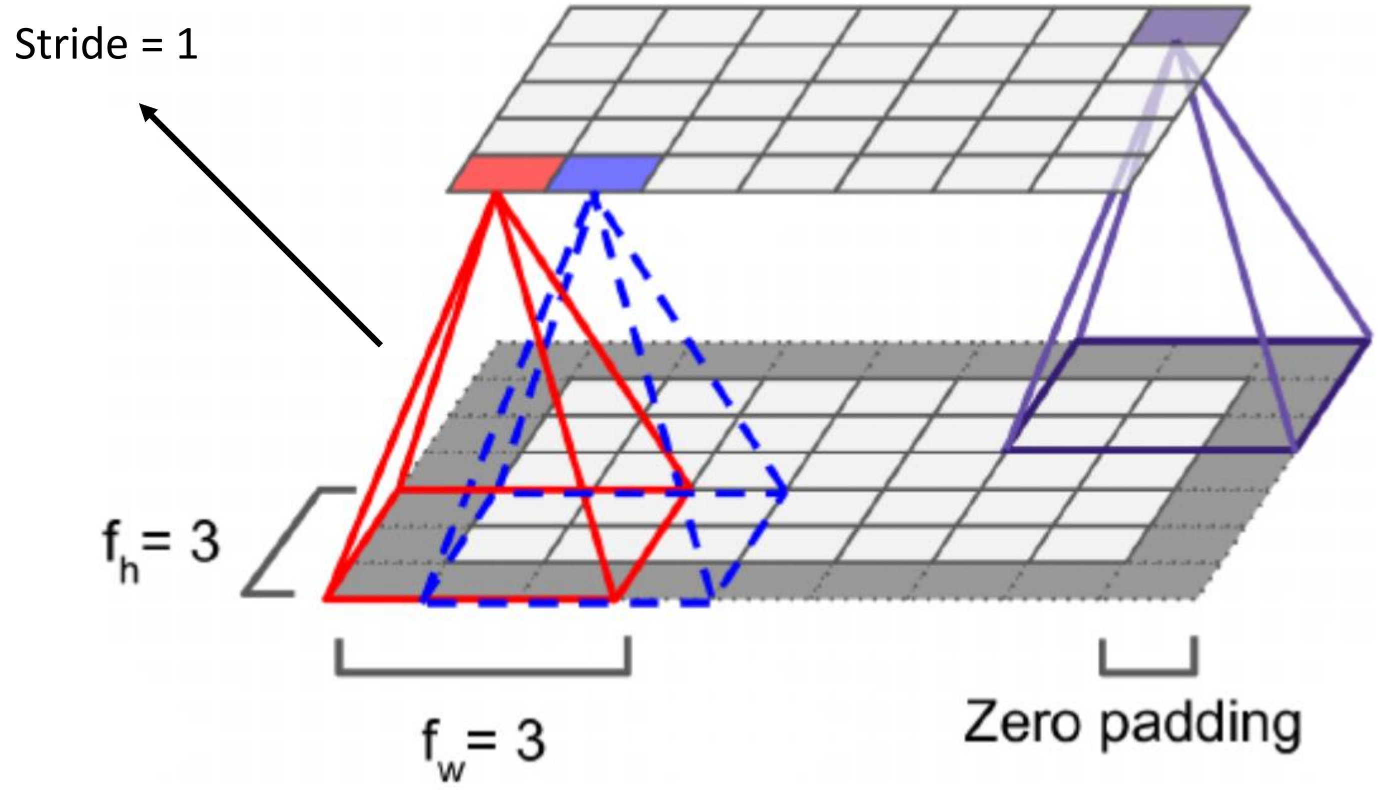


Extract horizontal features using a horizontal filter

Convolutional Layer



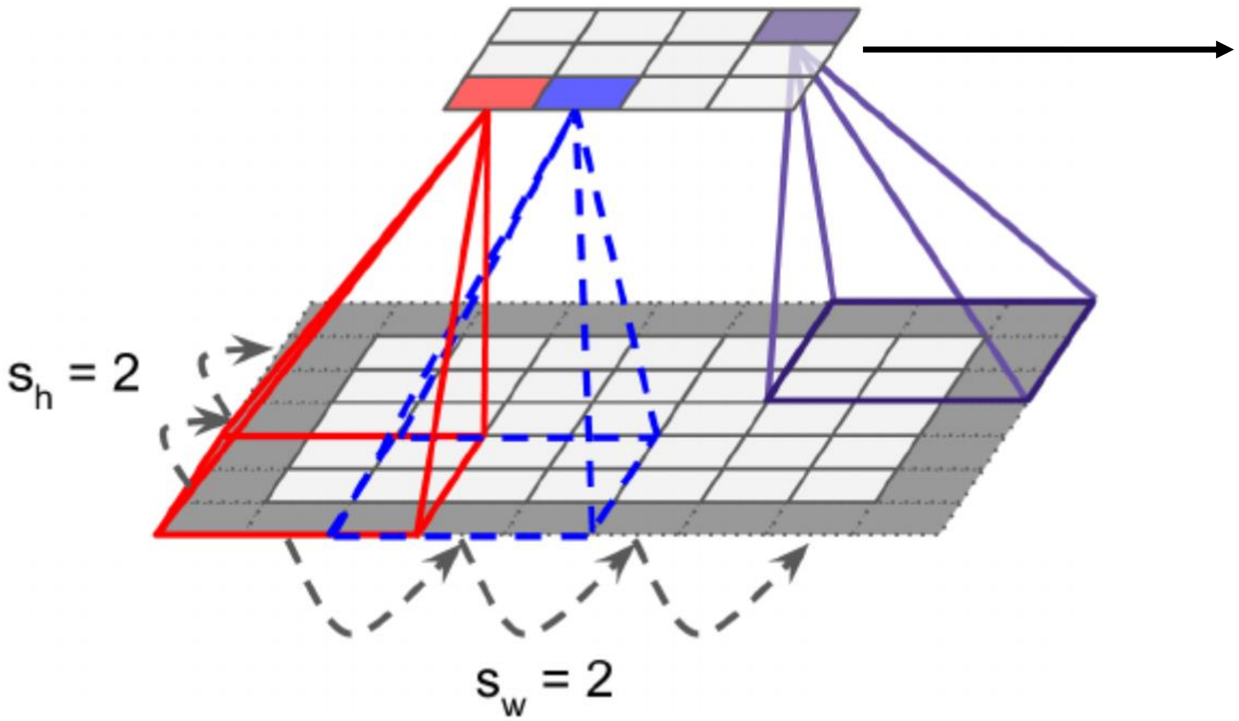
Convolutional Layer



Makes the size of second layer the same as first

Convolutional Layer

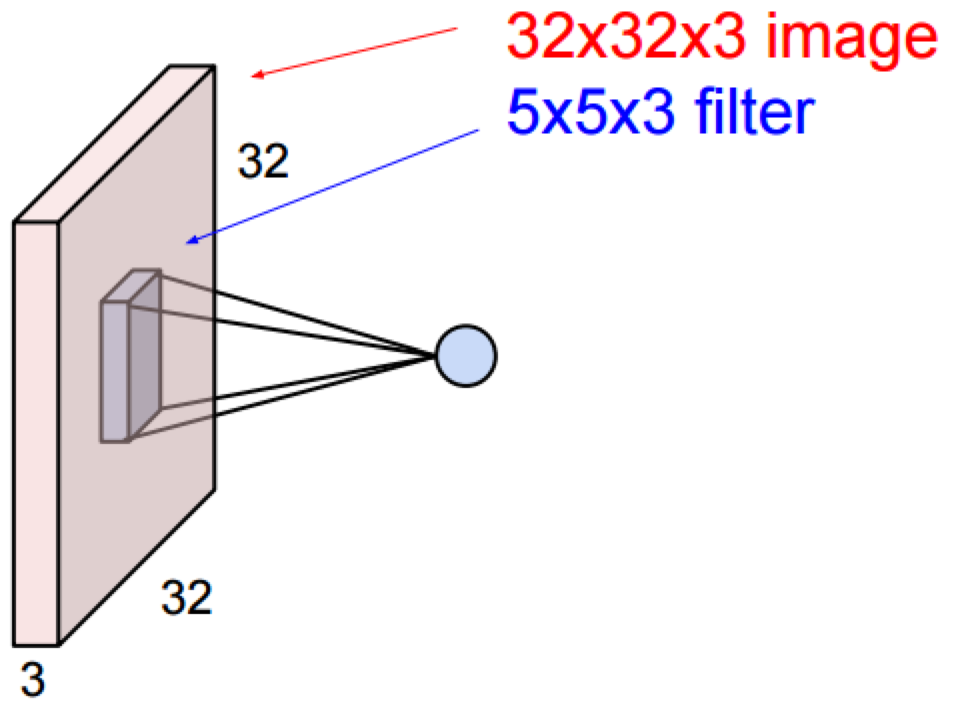
- You can have varying strides (both horizontally, and vertically)



Notice the size got smaller even with zero padding

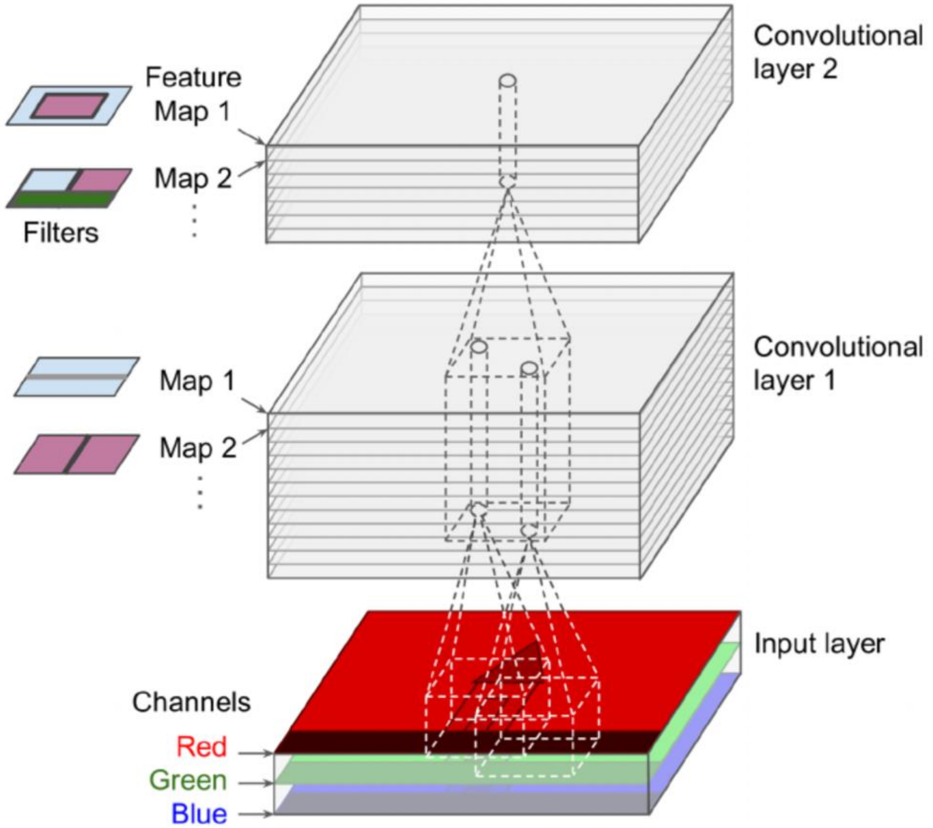
Convolutional Layer

- If you have RGB (3-channel) image, the filter has 3 channels too



Convolutional Layer

- Eventually, you have multiple feature maps (per convolutional layer)



Convolutional Layer

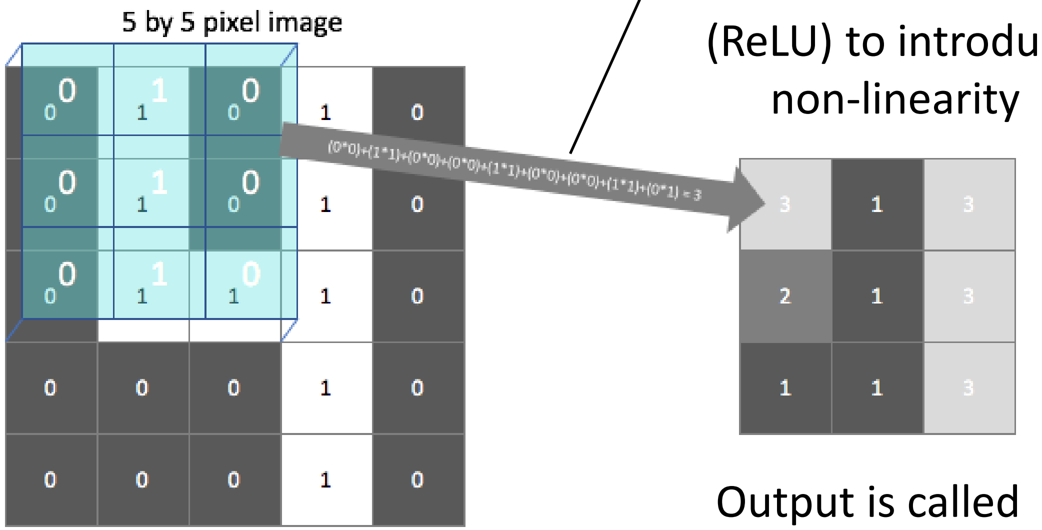
- How to process the values inside the “filter”
- Multiply them with “weights” then add them up
- Example: Apply “vertical” filter (to highlight vertical lines)

5 x 5 image

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

3 x 3 filter

0	1	0
0	1	0
0	1	0



(Actually, this sum is passed through an activation function (ReLU) to introduce non-linearity)

Output is called “feature map”

Convolutional Layer

- Then, apply "horizontal" filter (to highlight horizontal lines)

5 x 5 image

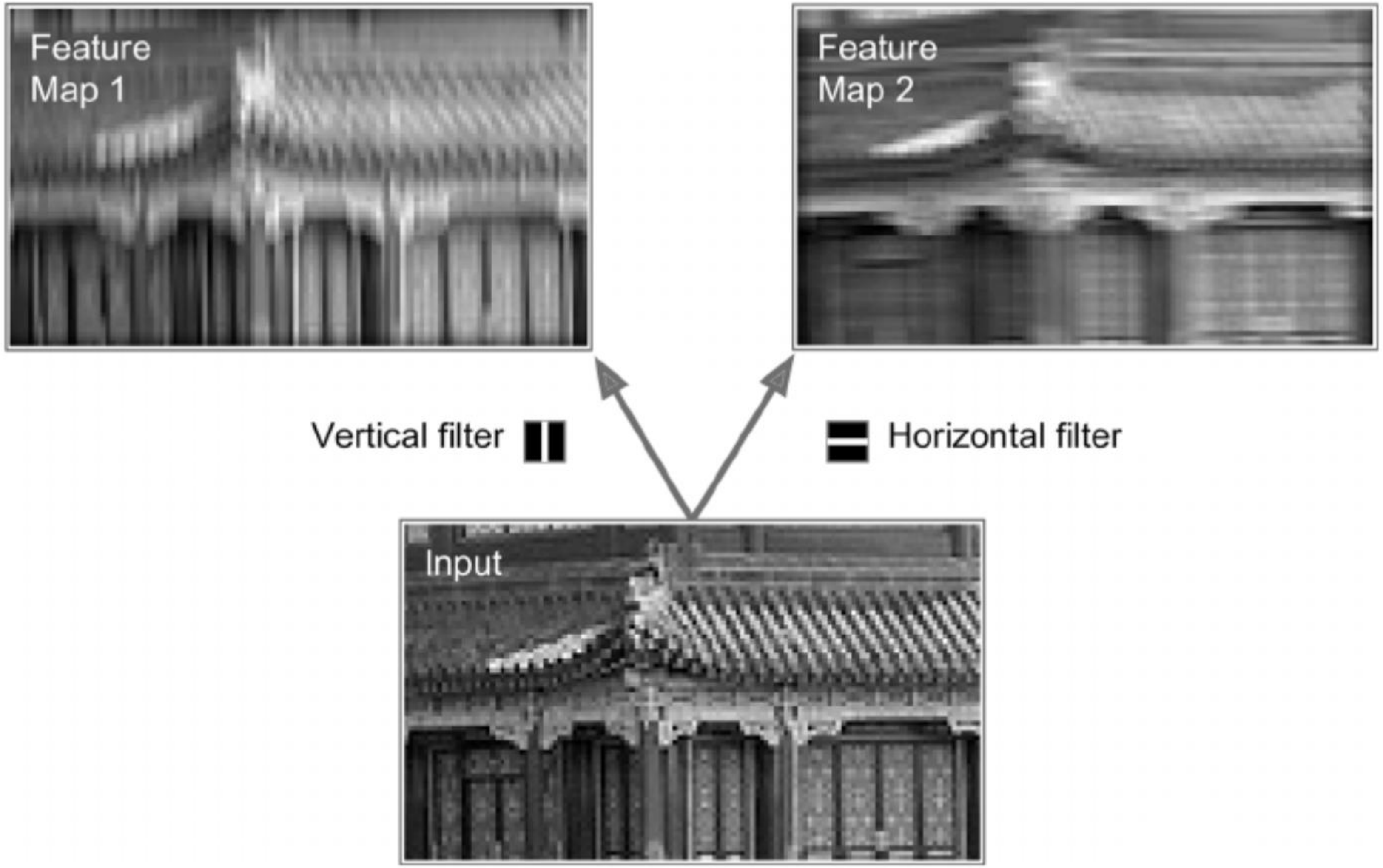
0	1	0	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	1	0
0	0	0	1	0

3 x 3 filter

0	0	0
1	1	1
0	0	0

- There are many more filters…

Convolutional Layer



Memory Requirements

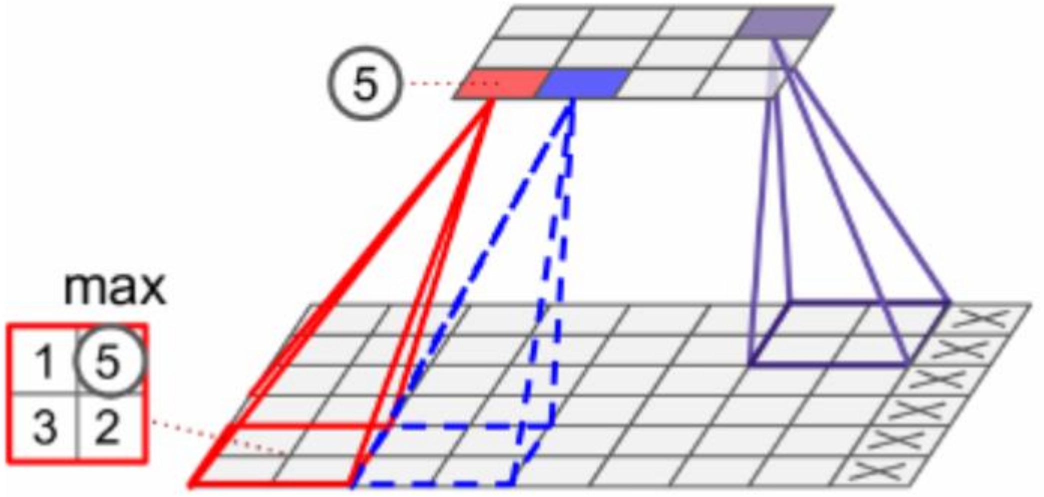
- Imagine a single image with 150 x 100 pixels
  - Filter of 5 x 5 with 1 stride
  - 200 feature maps
  - Padding (add when necessary)
- Calculations:
  - (5 x 5 x 3 + 1) x 200 = 15,200 terms
  - Each feature map has 150 x 100 neurons that need to compute 5 x 5 x 3 inputs. Makes 225 million multiplications
- 1 image needs 12 MB RAM.
- 100 images 1.2 GB RAM.

Pooling Layer

- “Shrinks” the convolutional layer (by sampling)
- An “aggregation” technique!
- Why:
  - Reduce computational load
  - Reduce number of parameters
  - Reduce overfitting

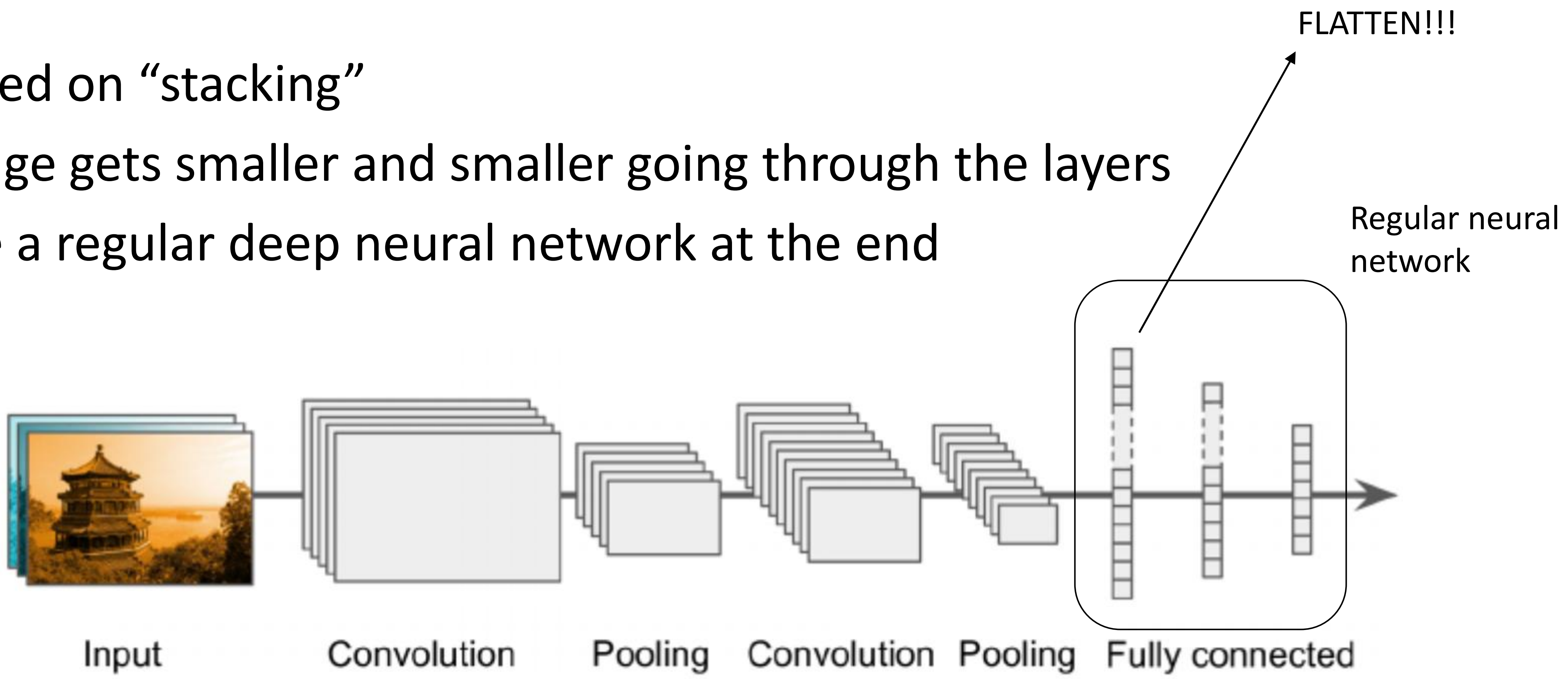
Pooling Layer

- Similar to earlier: define the size, stride, padding type
- It AGGREGATES the values
  - Example: use min, max, average, etc.
- A 2 x 2 kernel with a stride of 2 reduces the image (inputs) by 75%
- You can set it to shrink channels too



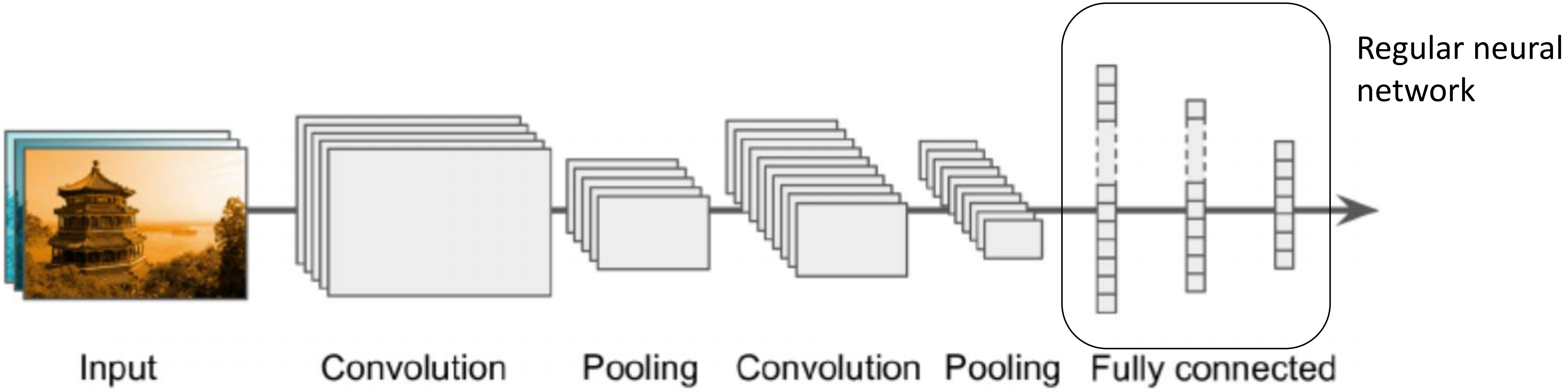
CNN Architectures

- Based on “stacking”
- Image gets smaller and smaller going through the layers
- Use a regular deep neural network at the end



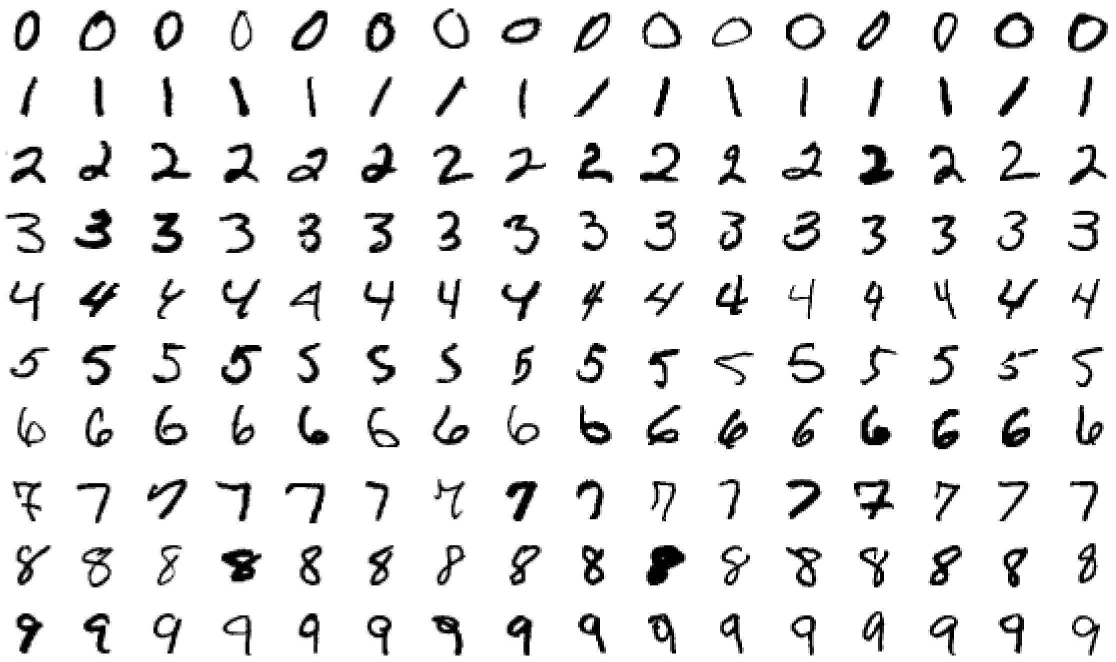
CNN Architectures

- Can also stack two convolutional layers one after another
- Instead of having a 5 x 5 convo layer, have two
  3 x 3 stacked convo layers. Might perform better!
- Double the filter size as you go deeper in CNN



LeNET-5

- Developed in 1998
- Used the MNIST data set (handwritten digits)



- **Labeled data**
- Black and white
- 28x28
- 60,000 images for training
- 10,000 images for testing

LeNET-5

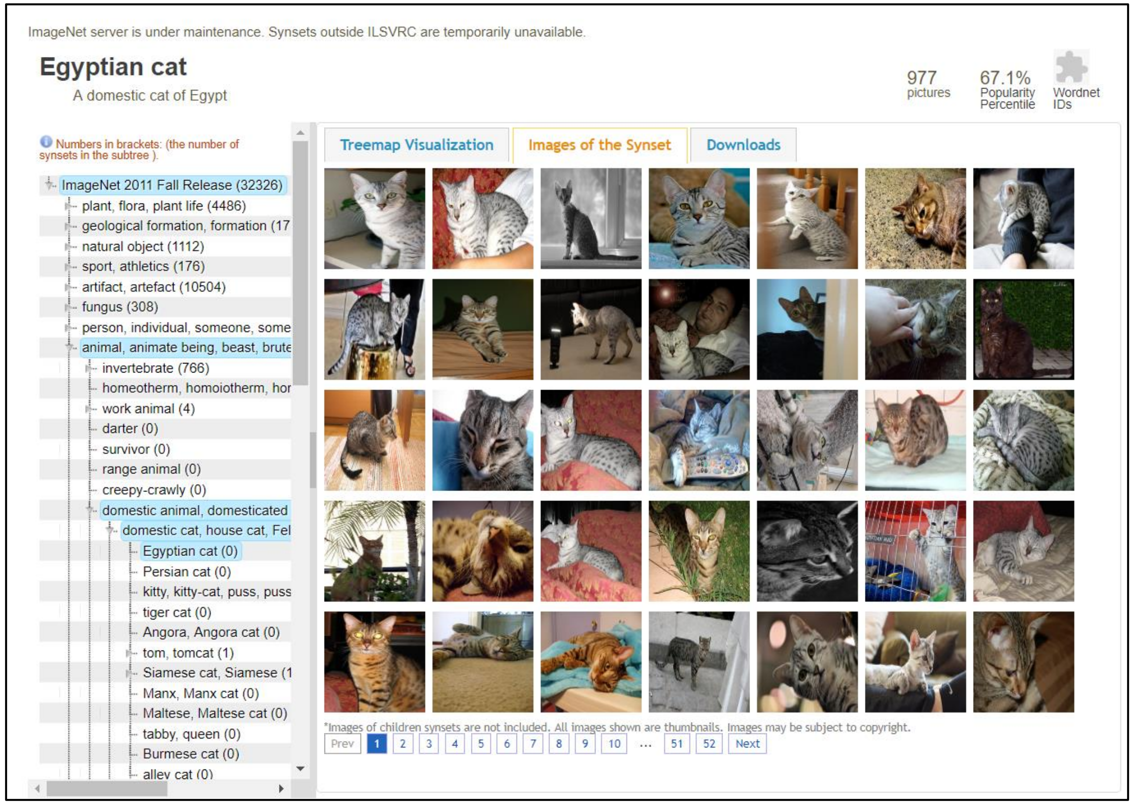
- Architecture:

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

Error: less than 1%

AlexNET

- Developed in 2012
- Achieved 17% error rate in ILSVRC imagenet challenge
  - Labeled data (1,000 classes)
  - 150,000 images in total



AlexNET

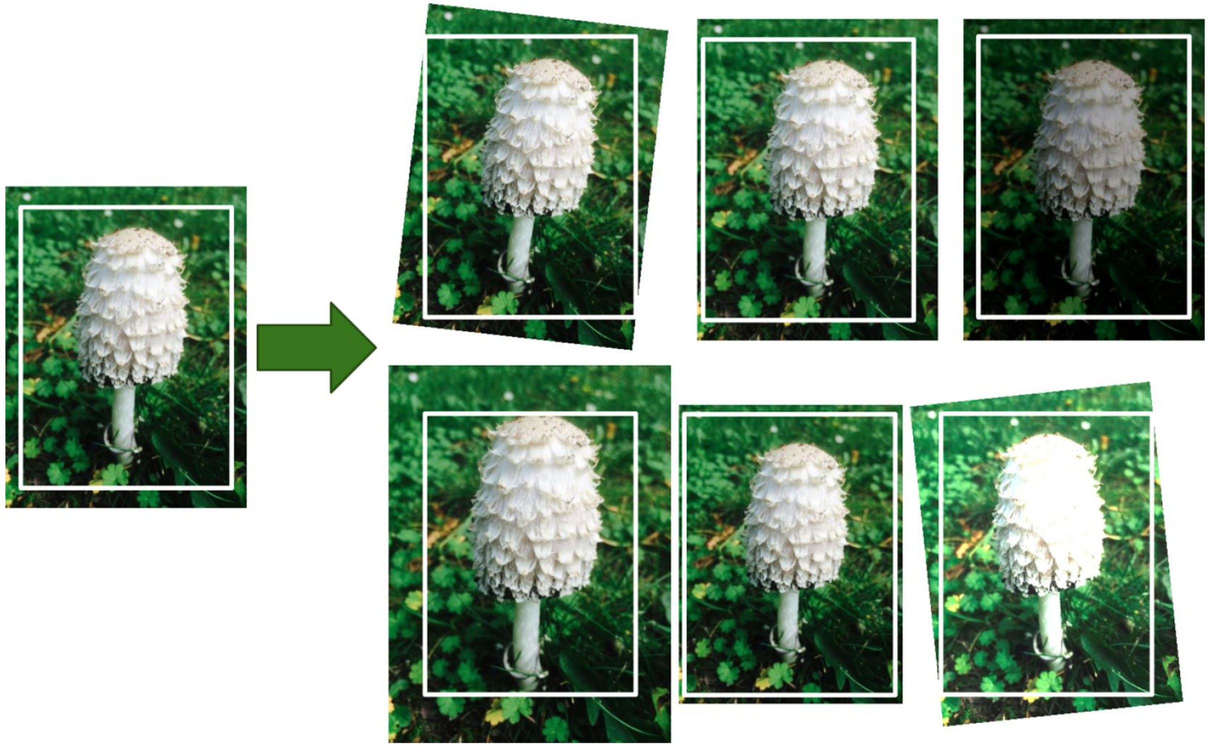
- Architecture:

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	valid	–
C7	Convolution	256	13 × 13	3 × 3	1	same	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	same	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	same	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	valid	–
C3	Convolution	256	27 × 27	5 × 5	1	same	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	valid	–
C1	Convolution	96	55 × 55	11 × 11	4	valid	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

- Used 50% dropout at F9 and 10
- Performed data augmentation
- Changed lighting conditions
- Local response normalization

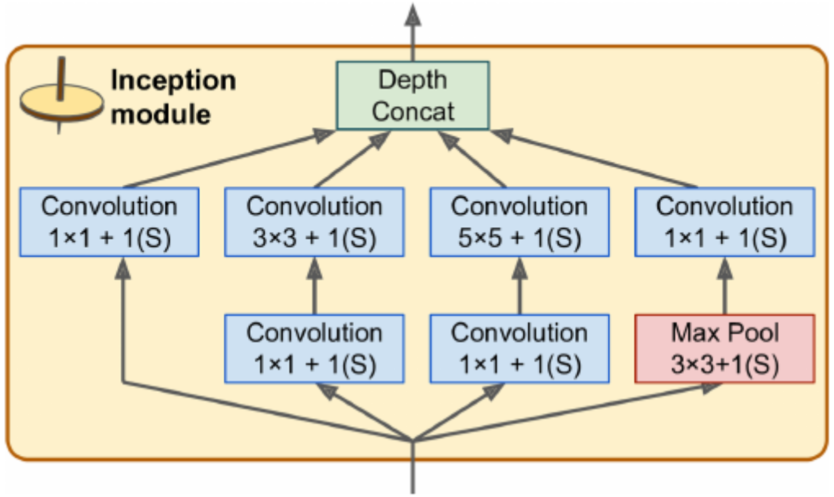
Data Augmentation

- Generate new images from existing training images
- Artificially increase the size
- Reduces overfitting



GoogLeNET

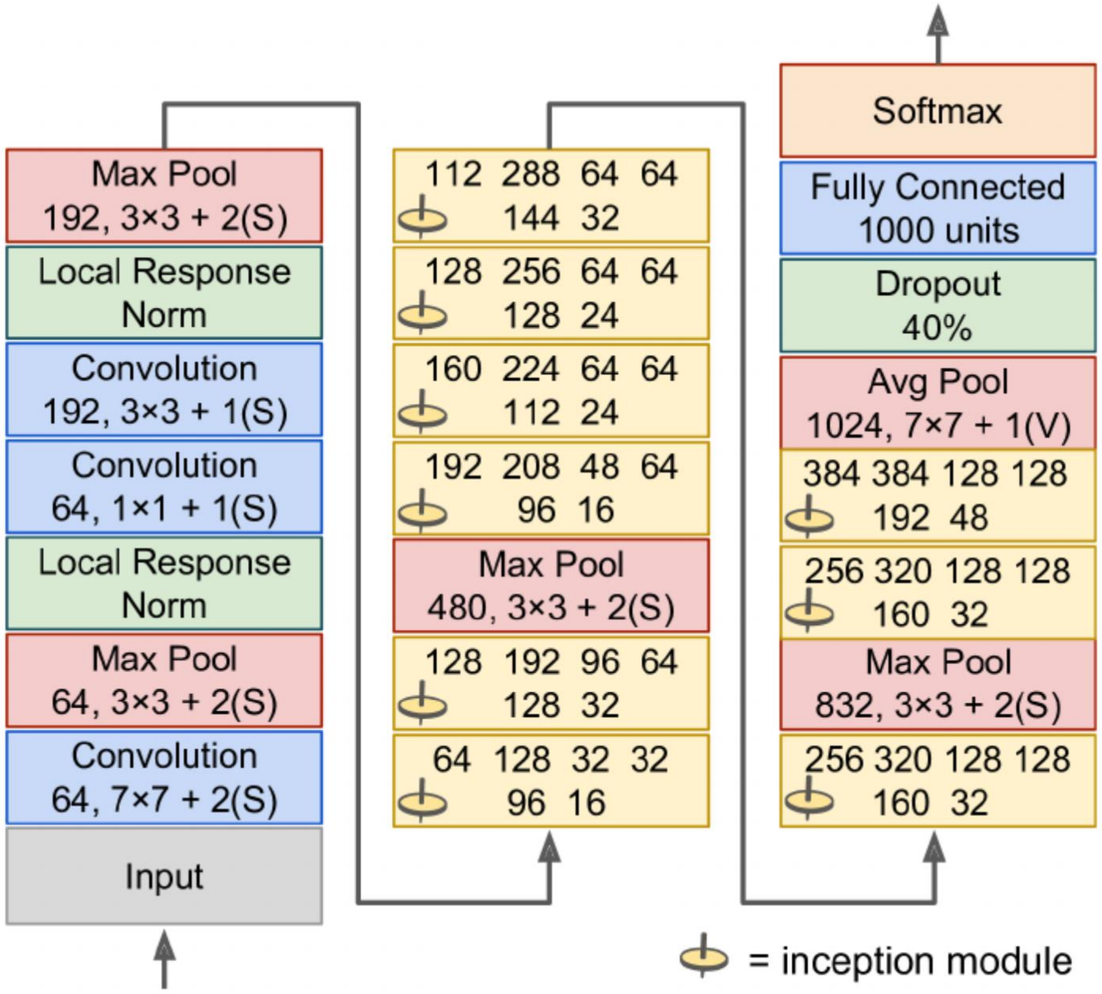
- Achieved an error rate of below 7% (ILSVRC imagenet challenge)
- Deeper network
- Reduced AlexNET's parameters by 10 times (from 60M to 6M)
- Used sub-networks called inception



Why use 1x1 filters?

- Capture patterns along depth dimension
- Reduce dimensionality (bottleneck layers)

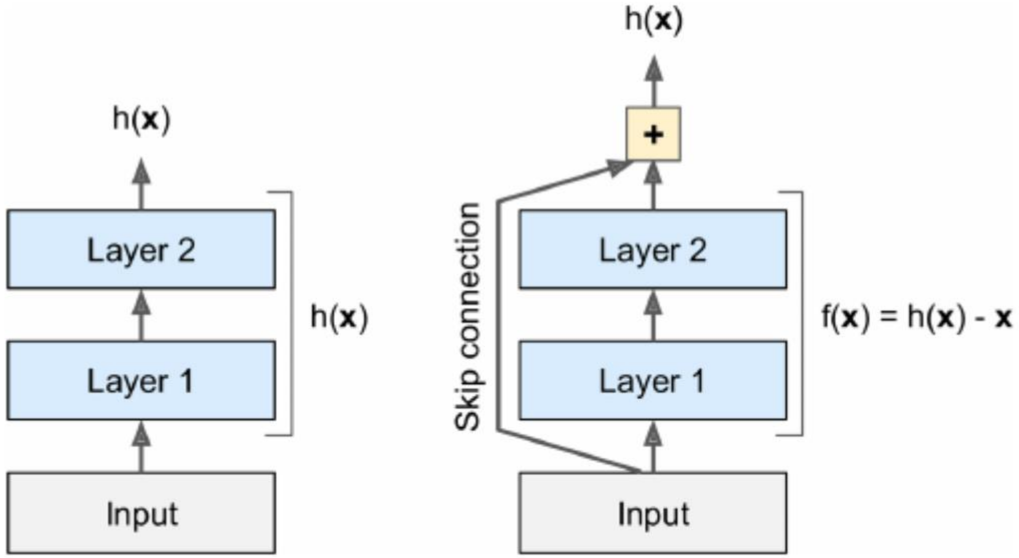
GoogLeNET



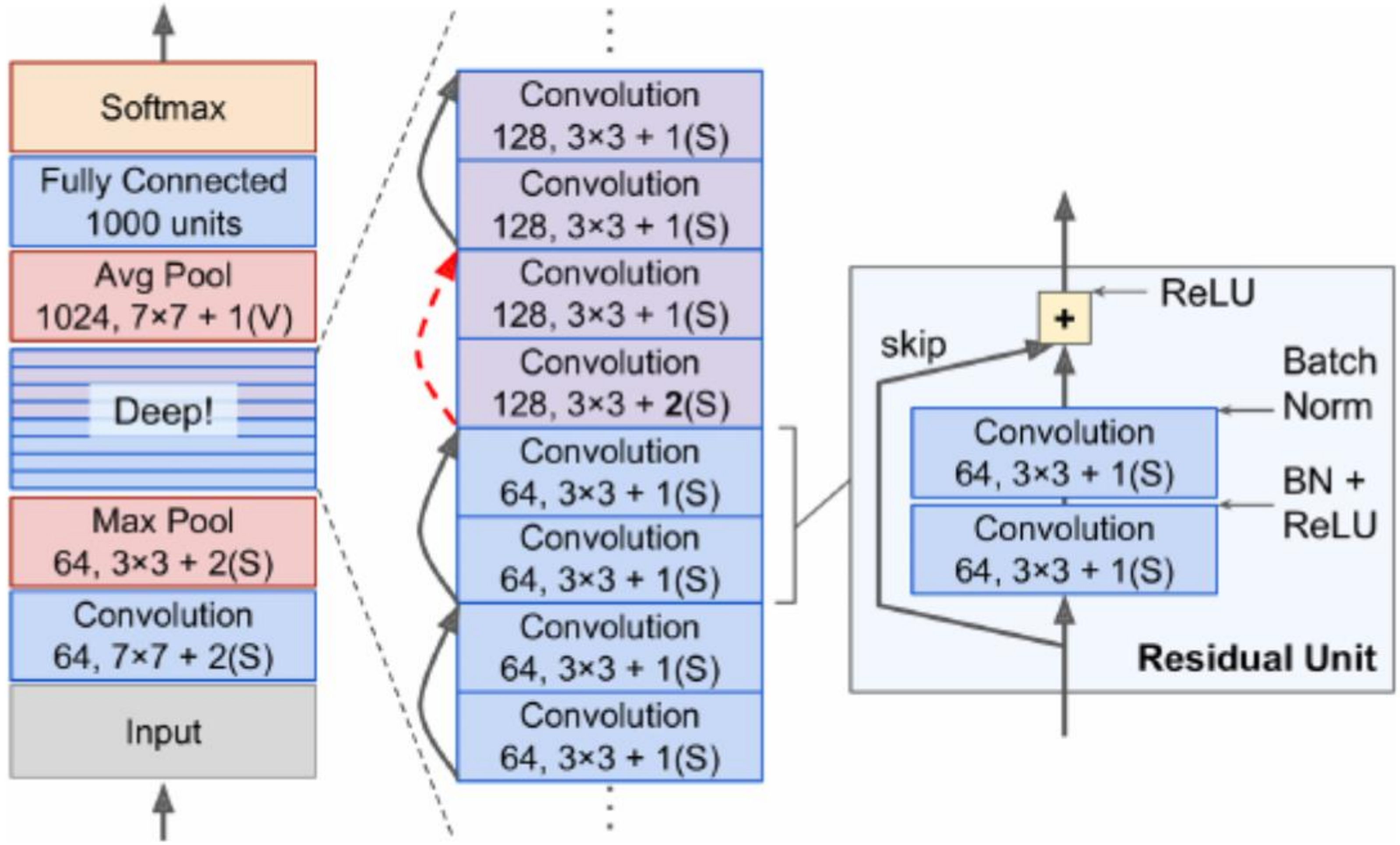
Softmax
Fully Connected 1000 units
Dropout 40%
Avg Pool 1024, 7×7 + 1(V)
384 384 128 128 192 48
256 320 128 128 160 32
Max Pool 832, 3×3 + 2(S)
256 320 128 128 160 32
112 288 64 64 144 32
128 256 64 64 128 24
160 224 64 64 112 24
192 208 48 64 96 16
Max Pool 480, 3×3 + 2(S)
128 192 96 64 128 32
64 128 32 32 96 16
Max Pool 192, 3×3 + 2(S)
Local Response Norm
Convolution 192, 3×3 + 1(S)
Convolution 64, 1×1 + 1(S)
Local Response Norm
Max Pool 64, 3×3 + 2(S)
Convolution 64, 7×7 + 2(S)
Input
= inception module

ResNET

- Reduced the error rate below 3.6%
- A total of 152 layers
- Used skip (shortcut) connections
    - Helps perform residual learning
- Skipping connections helps train faster



ResNET



Softmax
Fully Connected
1000 units
Avg Pool
1024, 7×7 + 1(V)
Deep!
Max Pool
64, 3×3 + 2(S)
Convolution
64, 7×7 + 2(S)
Input
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 1(S)
Convolution
128, 3×3 + 2(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
Convolution
64, 3×3 + 1(S)
ReLU
skip
+
Batch
Norm
Convolution
64, 3×3 + 1(S)
BN +
ReLU
Convolution
64, 3×3 + 1(S)
Residual Unit

Other Models

- VGGNet
- Xception (a variant of GoogLeNet)
- SENet (reduced the error rate to 2.25% in 2017)
- Inception v4
    - Combination of GoogLeNet and Resnet

Data Requirements

- We need **labeled** images (most of the time)

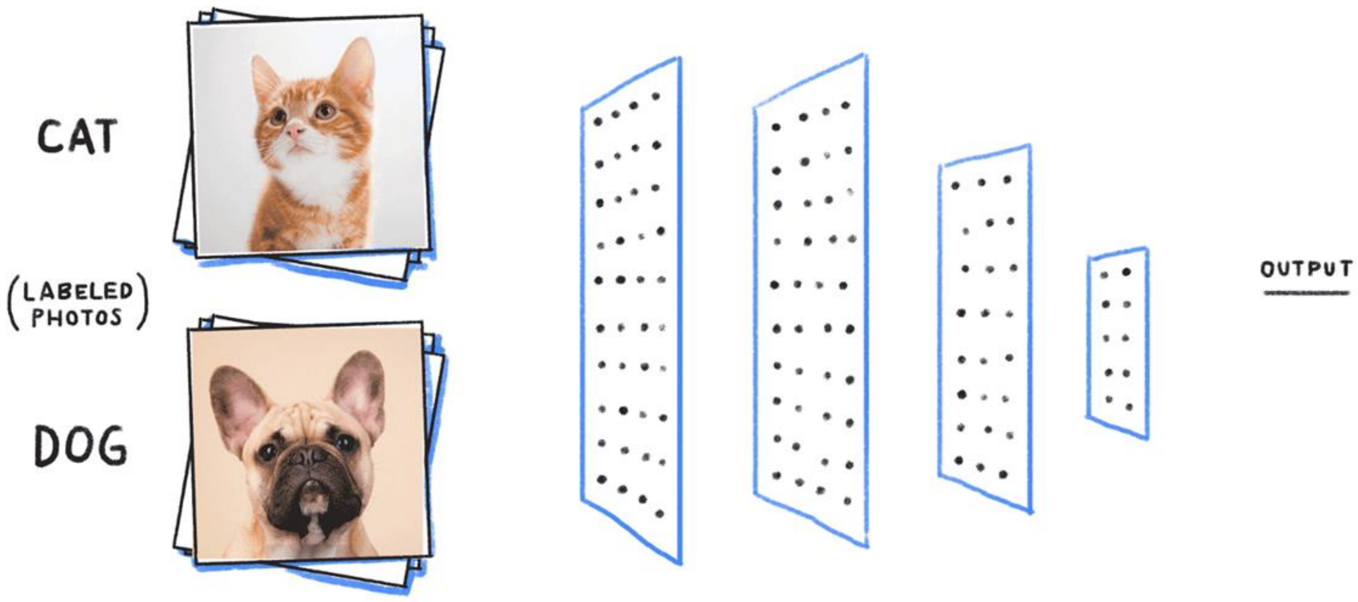
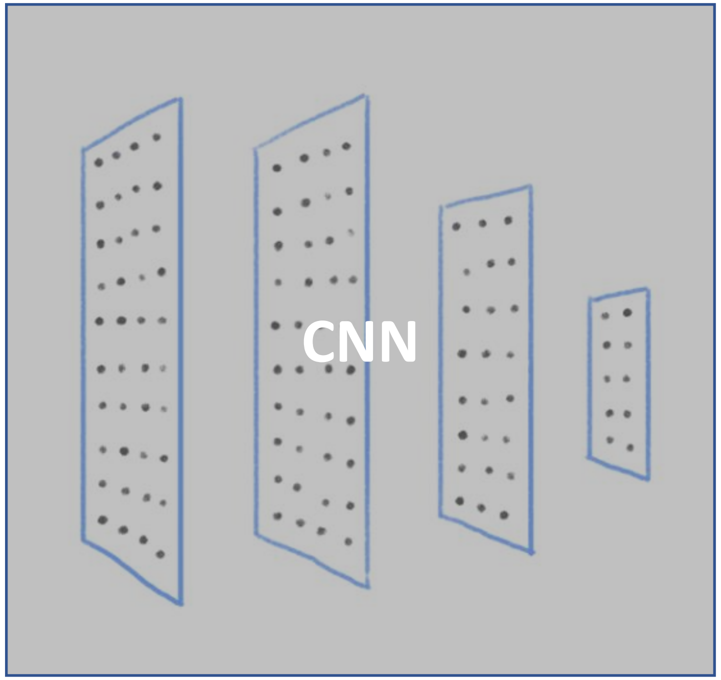


Image classification

New (unseen) image

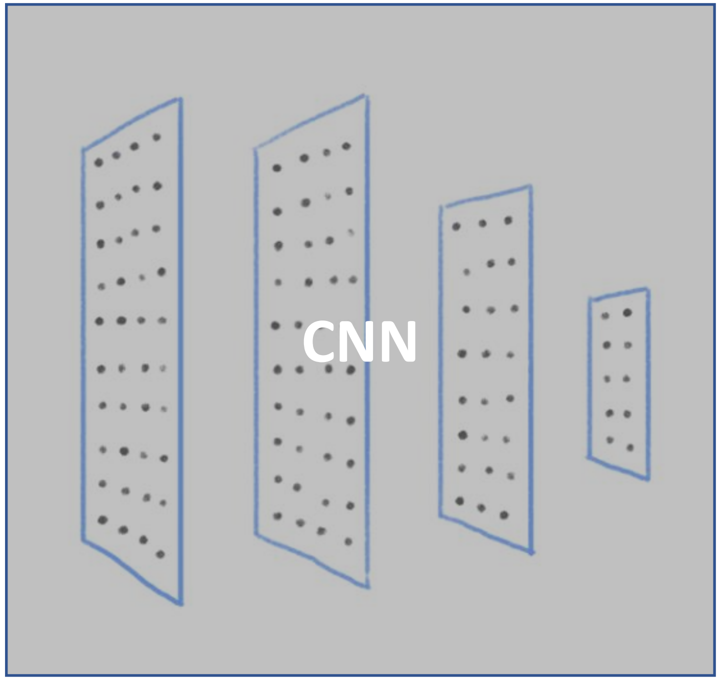


OUTPUT

Cat

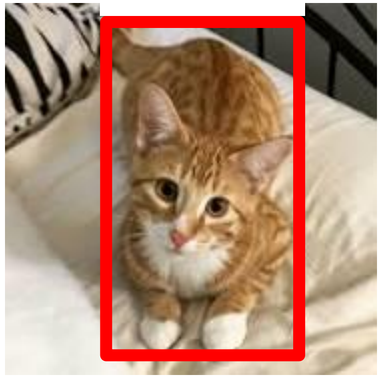
Object detection

New (unseen) image



OUTPUT

Cat 0.95



Object detection

- Example

A popular model:
You Only Look Once (YOLOv3)

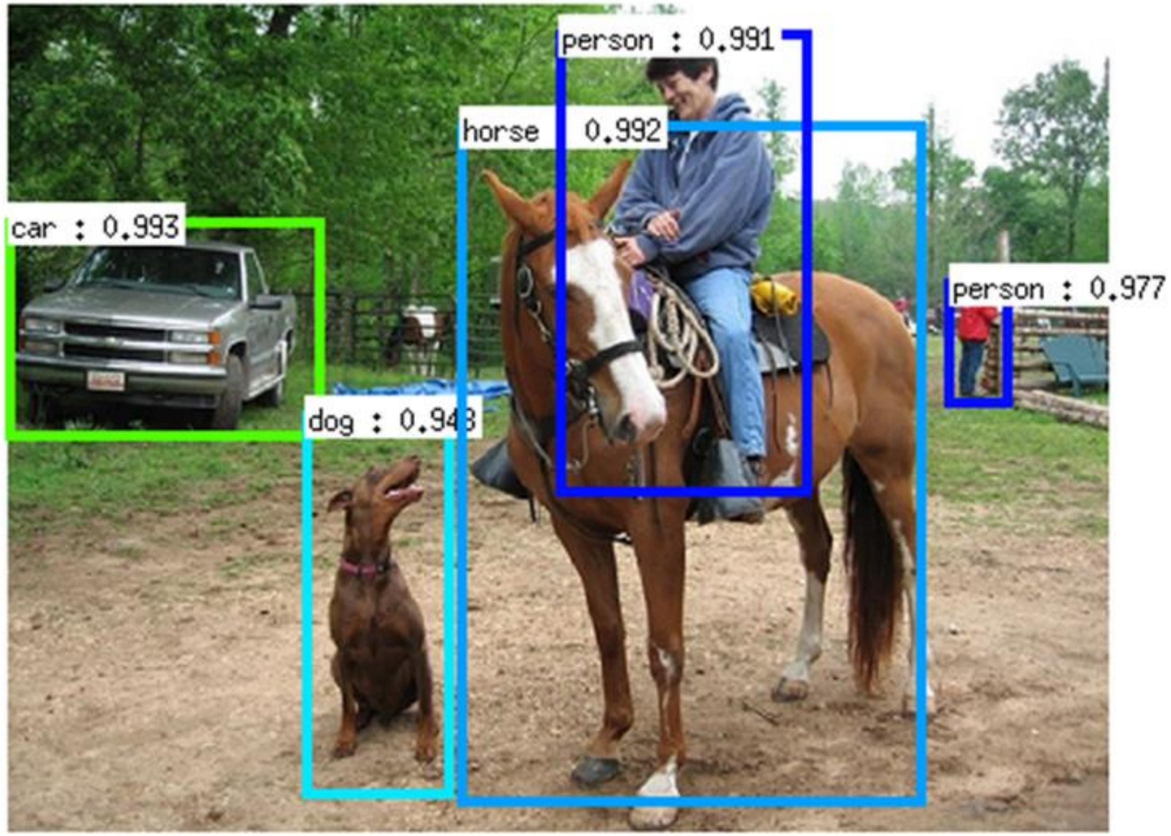
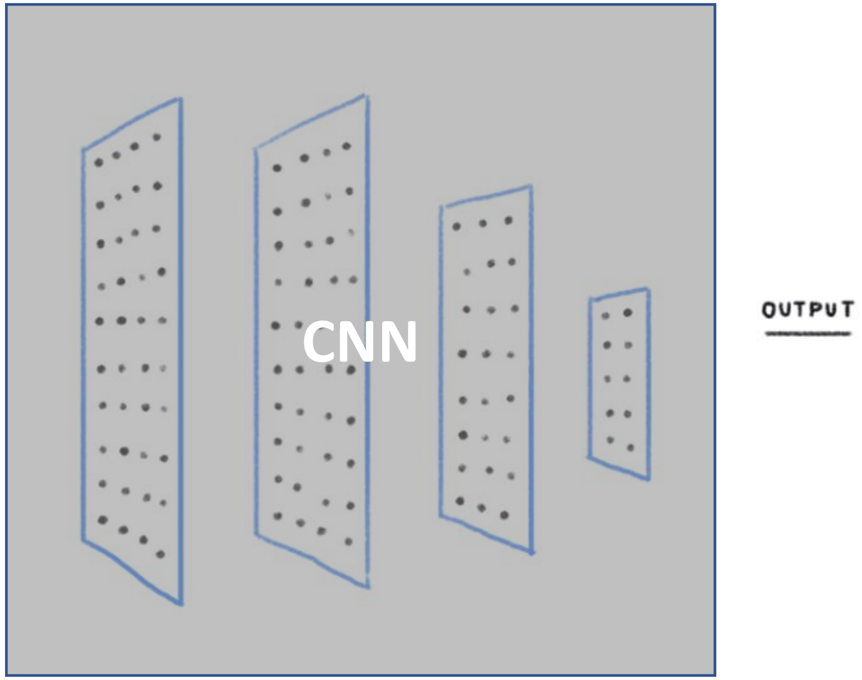


Image segmentation

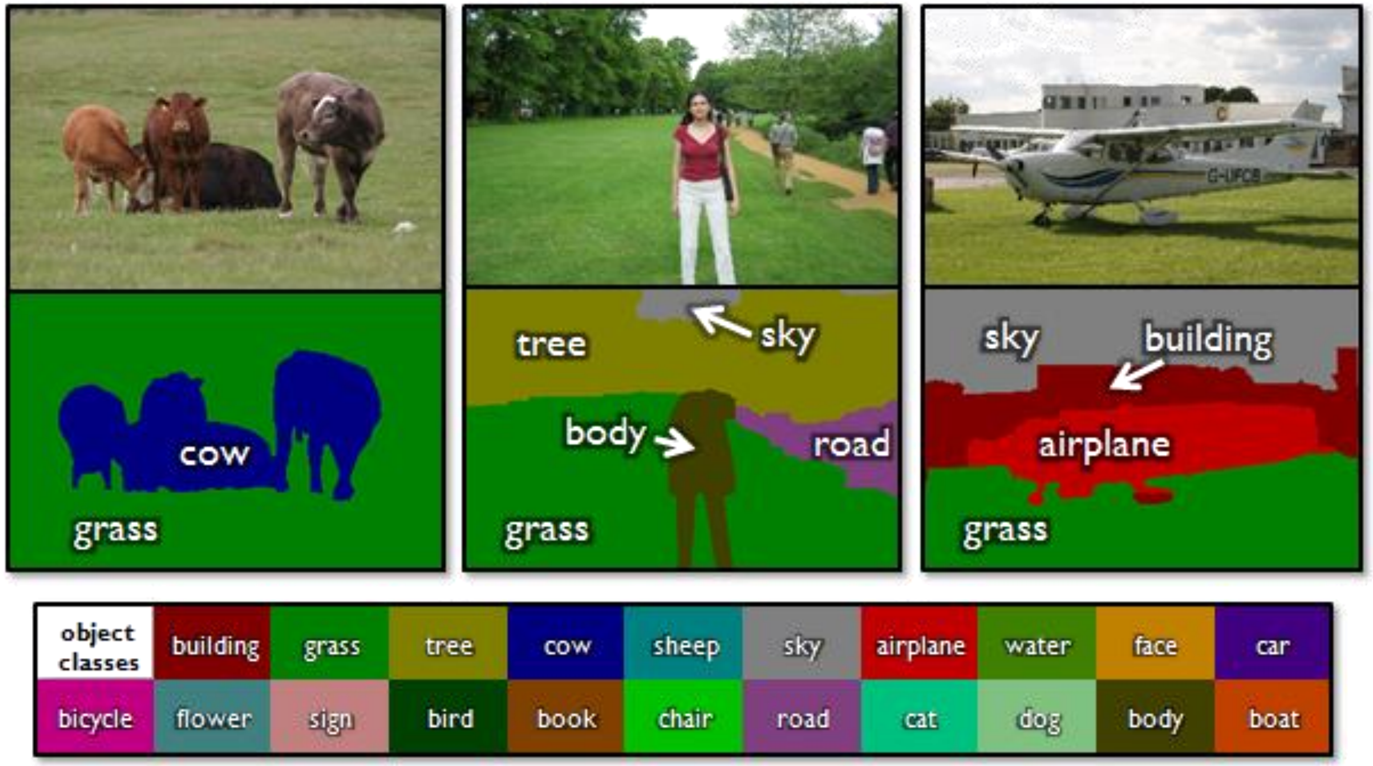
New (unseen) image



Cat

What Kind of Output Does it Generate?

- Image segmentation example



How Are We Going to Use CNNs?

- Image classification