

# Pooling in Convolutional Networks

Sargur Srihari  
[srihari@buffalo.edu](mailto:srihari@buffalo.edu)

# Topics in Convolutional Networks

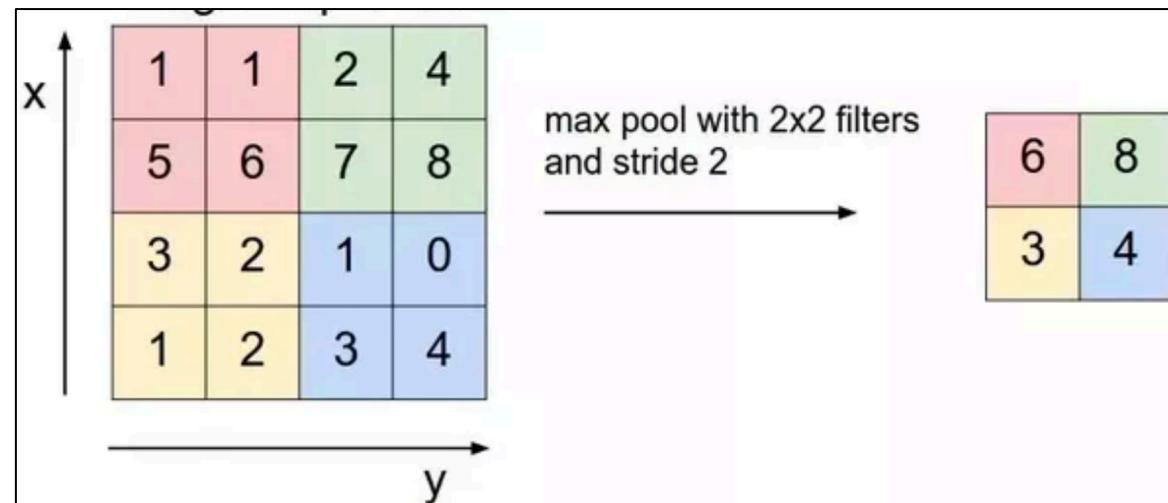
- Overview
- 1. The Convolution Operation
- 2. Motivation
- 3. Pooling
- 4. Convolution and Pooling as an Infinitely Strong Prior
- 5. Variants of the Basic Convolution Function
- 6. Structured Outputs
- 7. Data Types
- 8. Efficient Convolution Algorithms
- 9. Random or Unsupervised Features
- 10. The Neuroscientific Basis for Convolutional Networks
- 11. Convolutional Networks and the History of Deep Learning

## Topics in Pooling

- What is Pooling?
- Three stages of CNNs
- Two terminologies: simple layers, complex layers
- Types of Pooling functions: Max, Average
- Translation invariance
- Rotation invariance
- Pooling with downsampling
- ConvNet Architectures
- Shortcoming of pooling

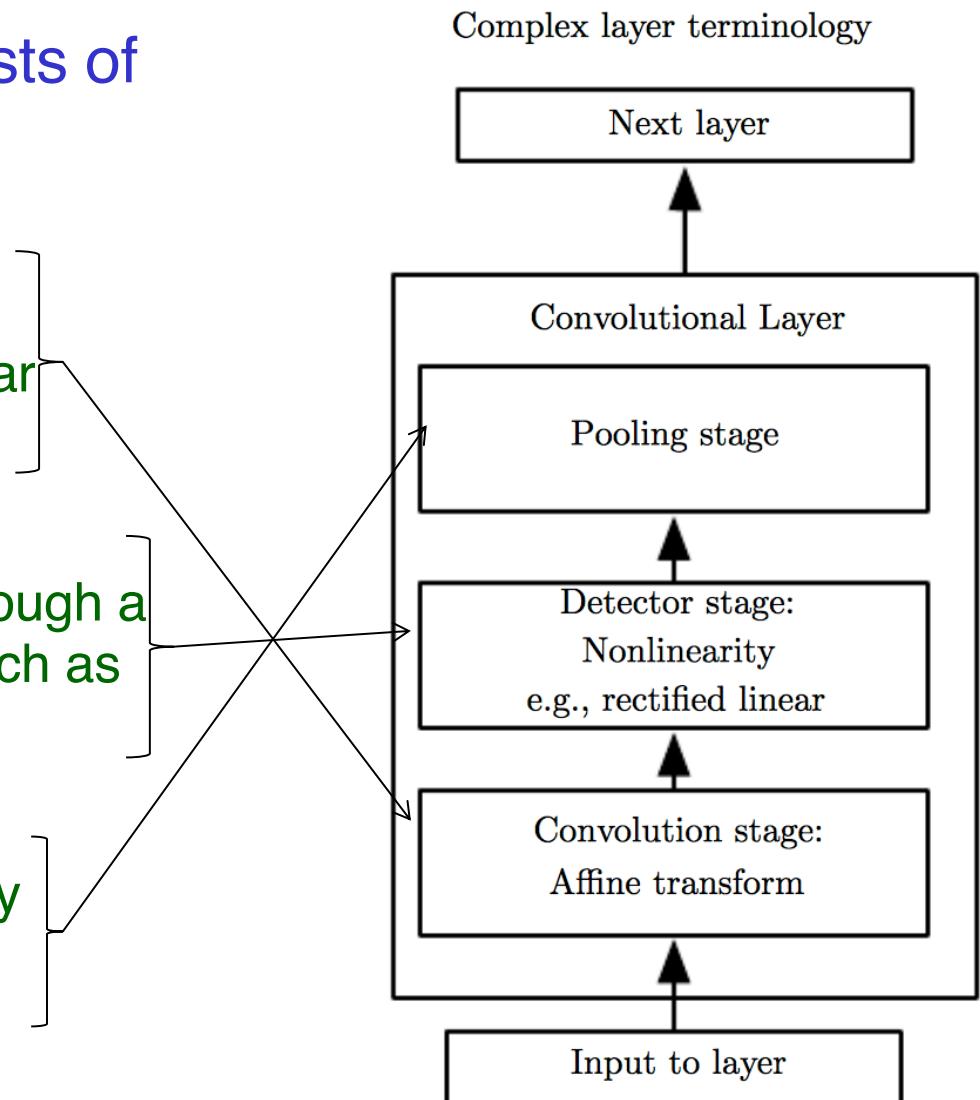
# What is Pooling?

- Pooling in a CNN is a subsampling step
  - It replaces output at a location with a summary statistic of nearby outputs
  - E.g., *Max pooling* reports the maximum output within a rectangular neighborhood



# The pooling stage in a CNN

- Typical layer of a CNN consists of three stages
- Stage 1:
  - perform several convolutions in parallel to produce a set of linear activations
- Stage 2 (Detector):
  - each linear activation is run through a nonlinear activation function such as ReLU
- Stage 3 (Pooling):
  - Use a pooling function to modify output of the layer further



# Pooling Layer in Keras

- MaxPooling 1D

```
keras.layers.MaxPooling1D(pool_size=2, strides=None, padding='valid', data_format='channels_las
```

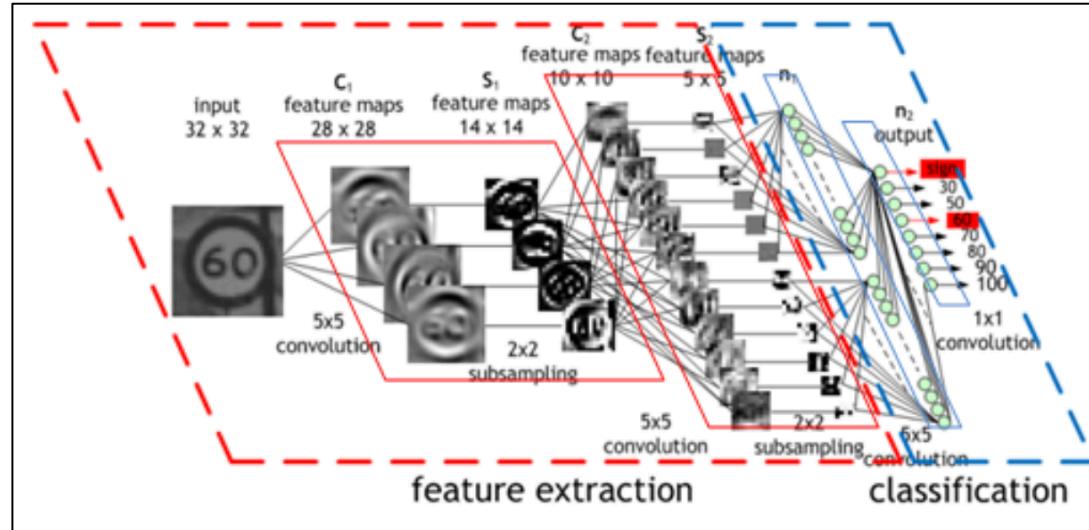
- Arguments

- **pool\_size**: Integer, size of the max pooling windows.
- **strides**: Integer, or None. Factor by which to downscale. E.g. 2 will halve the input. If None, it will default to pool\_size.
- **padding**: One of "valid" or "same" (case-insensitive).
- **data\_format**: A string, one of channels\_last (default) or channels\_first. The ordering of the dimensions in the inputs. channels\_last corresponds to inputs with shape (batch, steps, features) while channels\_first corresponds to inputs with shape (batch, features, steps)

- MaxPooling 2D

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

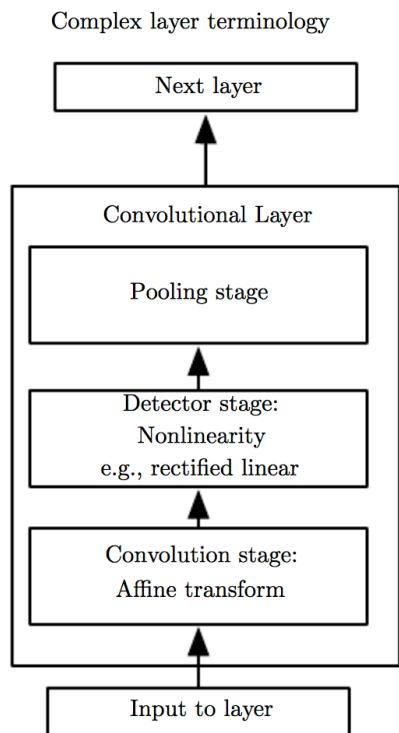
## Typical subsampling in a deep network



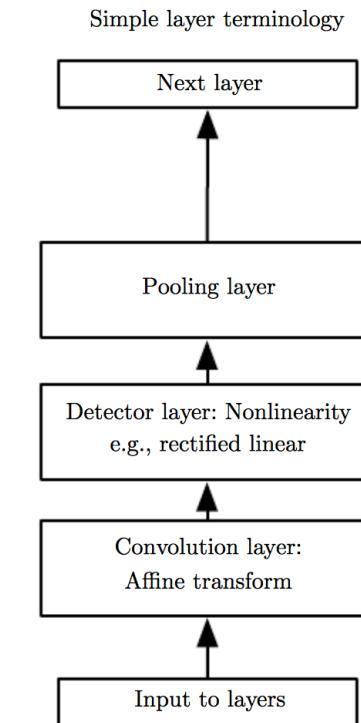
- Input image is filtered by 4  $5 \times 5$  convolutional kernels which create 4 feature maps,
- Feature maps are subsampled by max pooling.
- The next layer applies ten  $5 \times 5$  convolutional kernels to these subsampled images and again we pool the feature maps.
- The final layer is a fully connected layer where all generated features are combined and used in the classifier (essentially logistic regression).

# Two terminologies for a typical CNN layer

1. Net is viewed as a small no. of complex layers, each layer having many stages
  - 1-1 mapping between kernel tensors and network layers

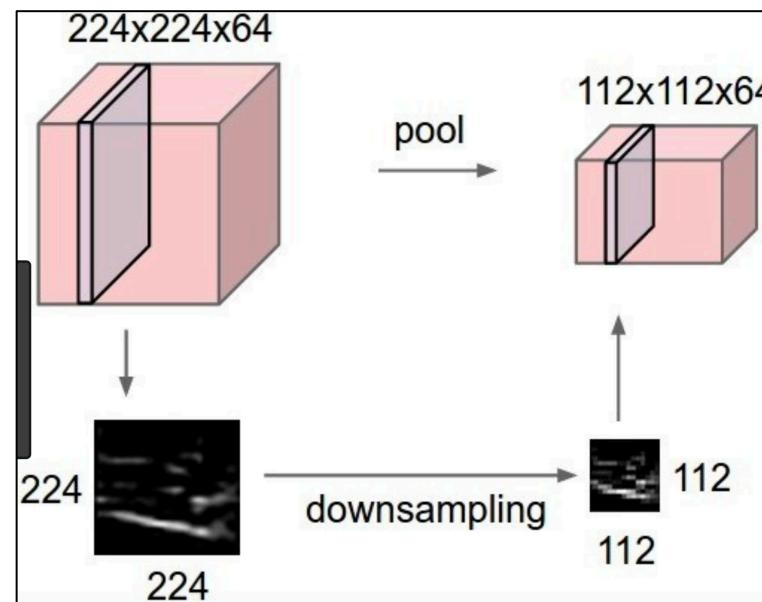


2. Net is viewed as a larger no of simple layers
  - Every processing step is a layer in its own right
  - Not every layer has parameters

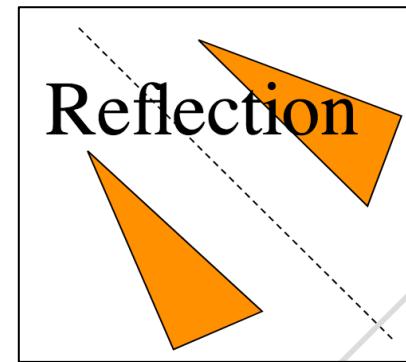
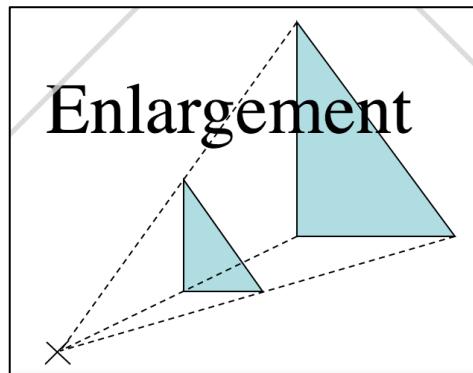
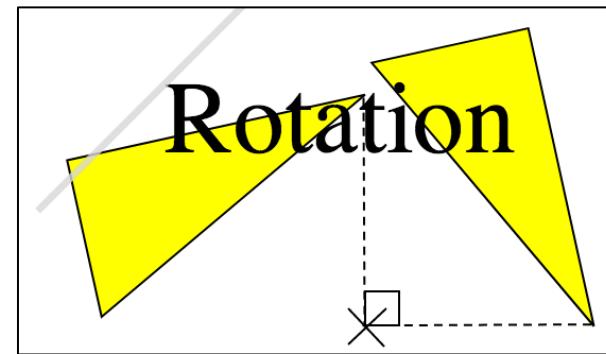
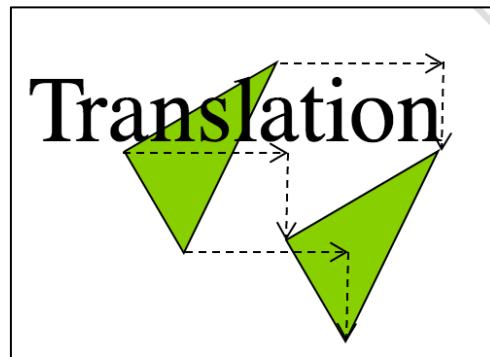


# Why is Pooling performed?

- Pooling is performed for two reasons
  1. Dimensionality reduction
  2. Invariance to transformations of rotation and translation



## Some types of linear transformations



# Types of Pooling functions

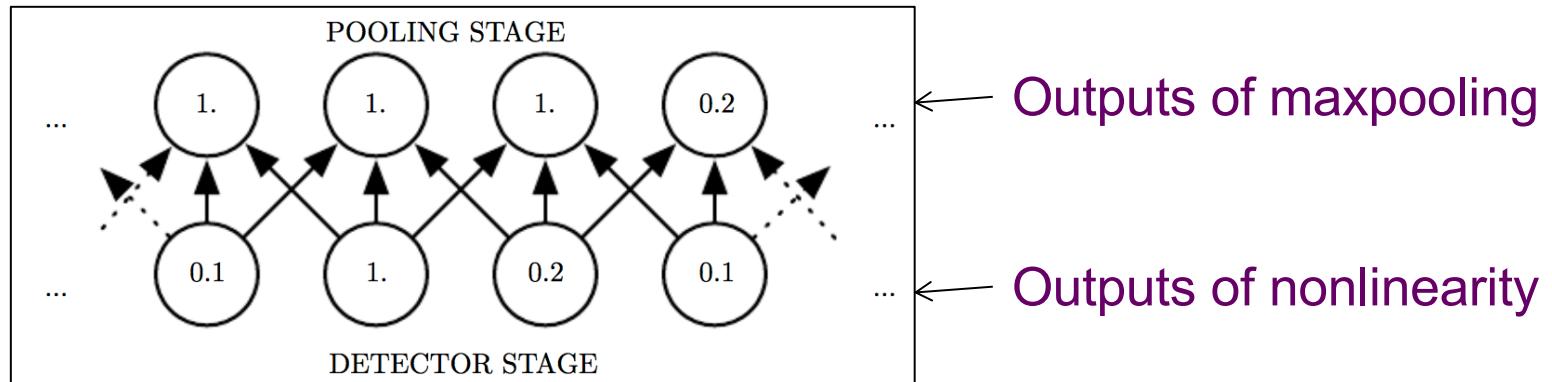
- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby inputs
- Popular pooling functions:
  1. *max pooling* operation reports the maximum output within a rectangular neighborhood
  2. Average of a rectangular neighborhood
  3.  $L^2$  norm of a rectangular neighborhood
  4. Weighted average based on the distance from the central pixel

# Pooling causes translation invariance

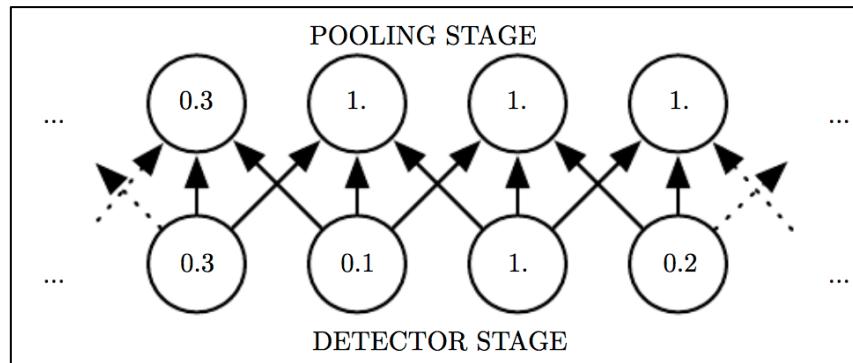
- In all cases, pooling helps make the representation become approximately *invariant* to small translations of the input
  - If we translate the input by a small amount values of most of the outputs does not change (example next slide)
  - Pooling can be viewed as adding a strong prior that the function the layer learns must be invariant to small translations

## Max pooling introduces invariance to translation

- View of middle of output of a convolutional layer



- Same network after the input has been shifted by one pixel



- Every input value has changed, but only half the values of output have changed because maxpooling units are only sensitive to maximum value in neighborhood not exact value

## Importance of Translation Invariance

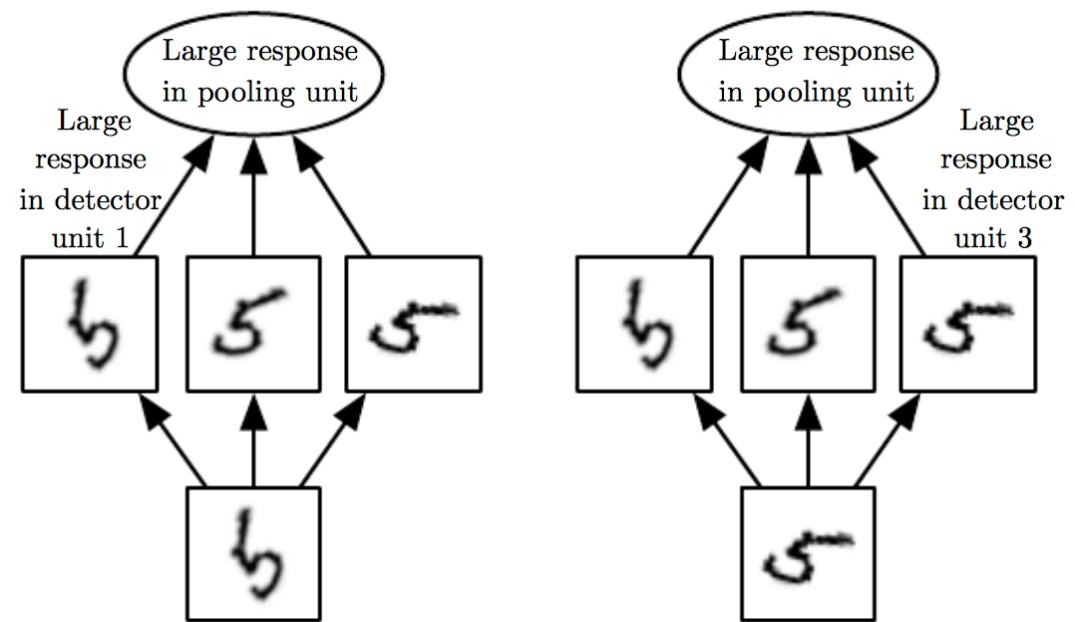
- Invariance to translation is important if we care about whether a feature is present rather than exactly where it is
  - For detecting a face we just need to know that an eye is present in a region, not its exact location
- In other contexts it is more important to preserve location of a feature
  - E.g., to determine a corner we need to know whether two edges are present and test whether they meet

## Learning other invariances

- Pooling over spatial regions produces invariance to translation
- But if we pool over the results of separately parameterized convolutions, the features can learn which transformations to become invariant to

## Learning Invariance to rotation

- A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input



Input tilted left  
gets large response  
from unit tuned to  
left-tilted images

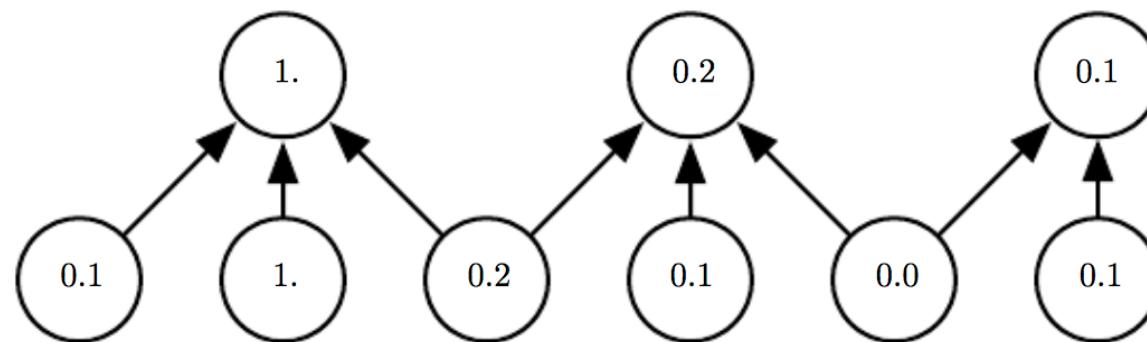
Tilted Right

## Using fewer pooling units than detector units

- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units
  - By reporting summary statistics for pooling regions spaced  $k$  pixels apart rather than one pixel apart
  - This improves computational efficiency because the next layer has  $k$  times fewer inputs to process
- An example is given next

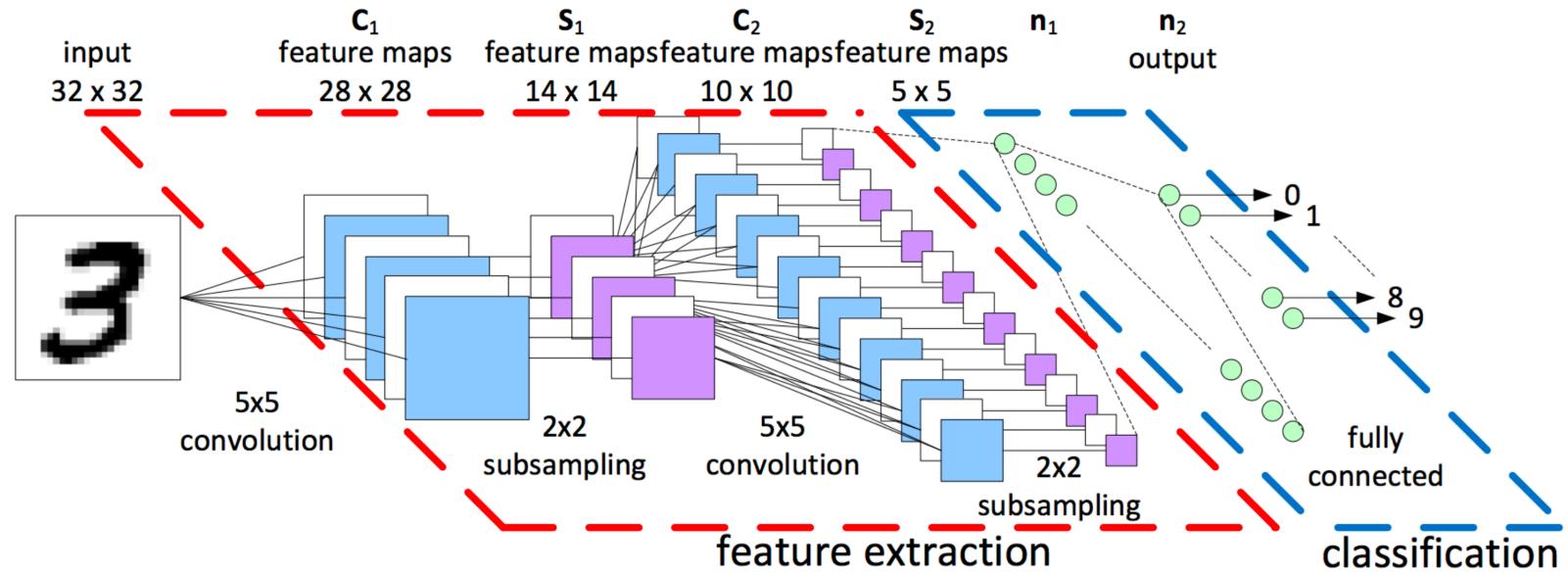
## Pooling with down-sampling

- Max-pooling with a pool width of three and a stride between pools of two



- This reduces representation size by a factor of two
  - Which reduces computational burden of next layer
  - Rightmost pooling region has a smaller size but must be included if we don't want to ignore some of the detector units

## Subsampling as Average pooling

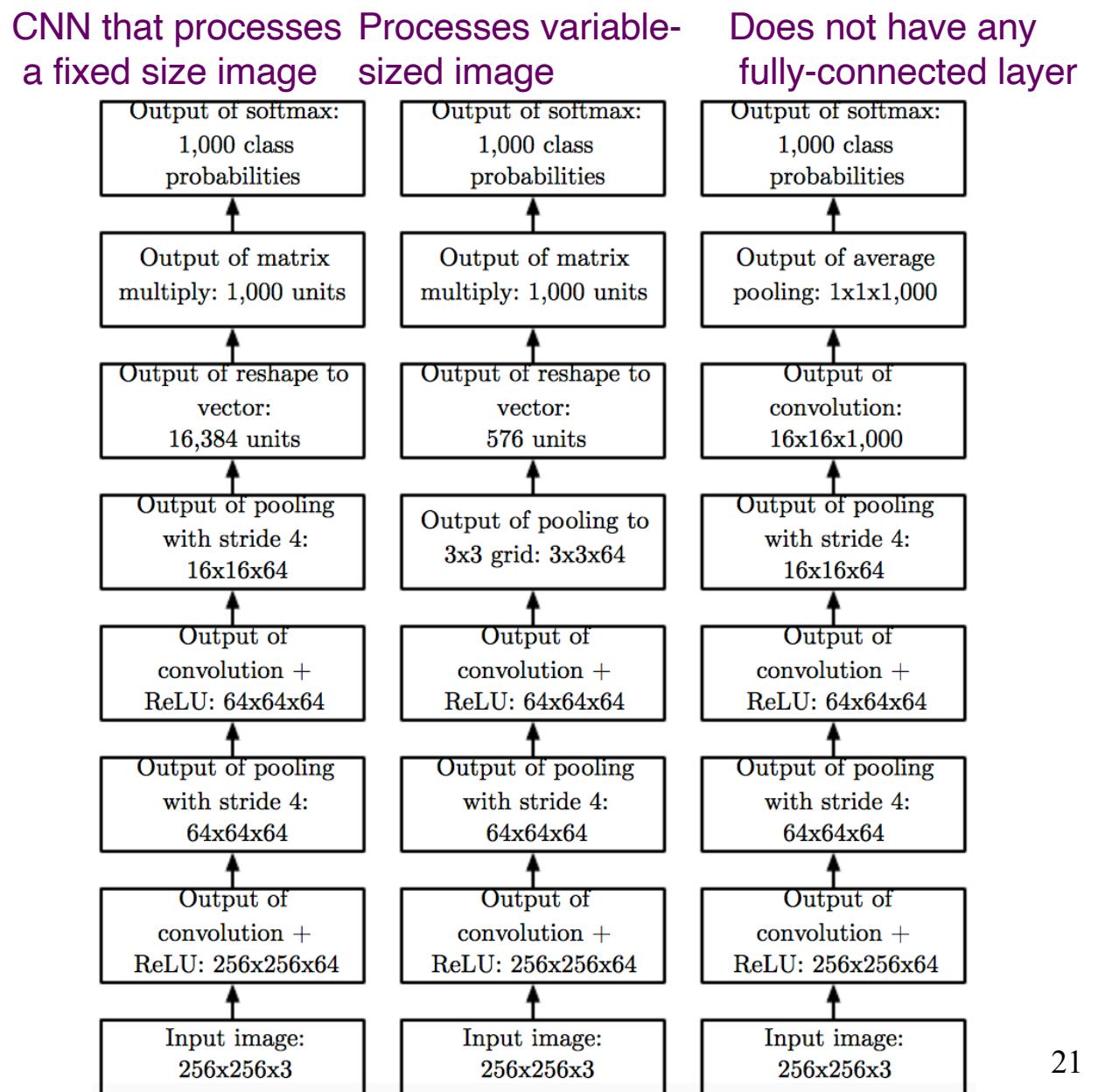


# Theoretical Guidance on Pooling

- Which kind of pooling one should use in different situations
- It is possible to dynamically pool features together
  - By running a clustering algorithm on locations of interesting features
    - Yields a different set of pooling regions for each image
    - Another approach: learn a single pooling structure
- Pooling can complicate architectures that use top-down information
  - E.g., Boltzmann machines and autoencoders

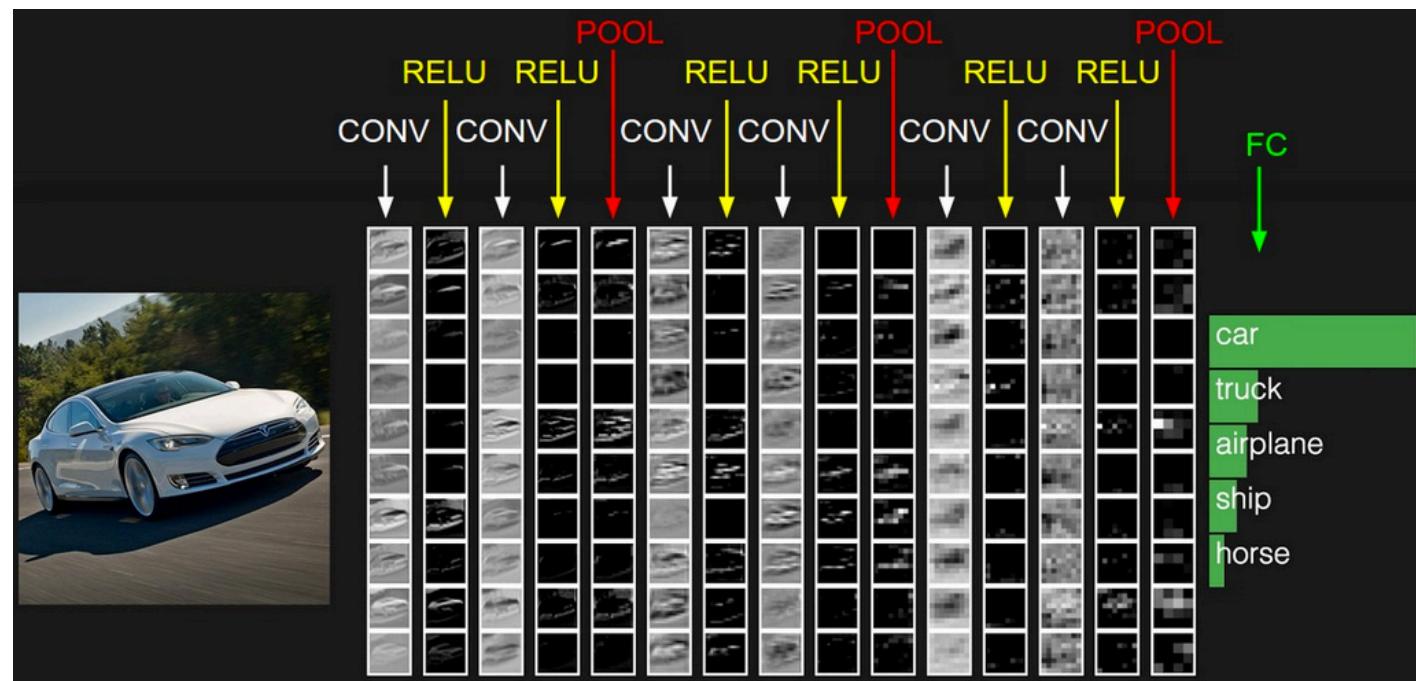
# Examples of Architectures for Classification with CNNs

- Real networks have branching structures.
- Chain structures shown for simplicity



# A ConvNet architecture

INPUT: 32x32x3 holds raw pixel values: an image of width 32, height 32 and 3 color channels RGB  
 CONV layer will compute the output of neurons connected to local regions in the input  
 Each computing a dot product between their weights and a small region they are connected to  
 In the input volume. This may result in a volume such as 32x32x12 if we used 12 filters  
 POOL layer will perform a down-sampling operation along spatial dimensions (width, height)  
 resulting in a volume such as 12x16x12



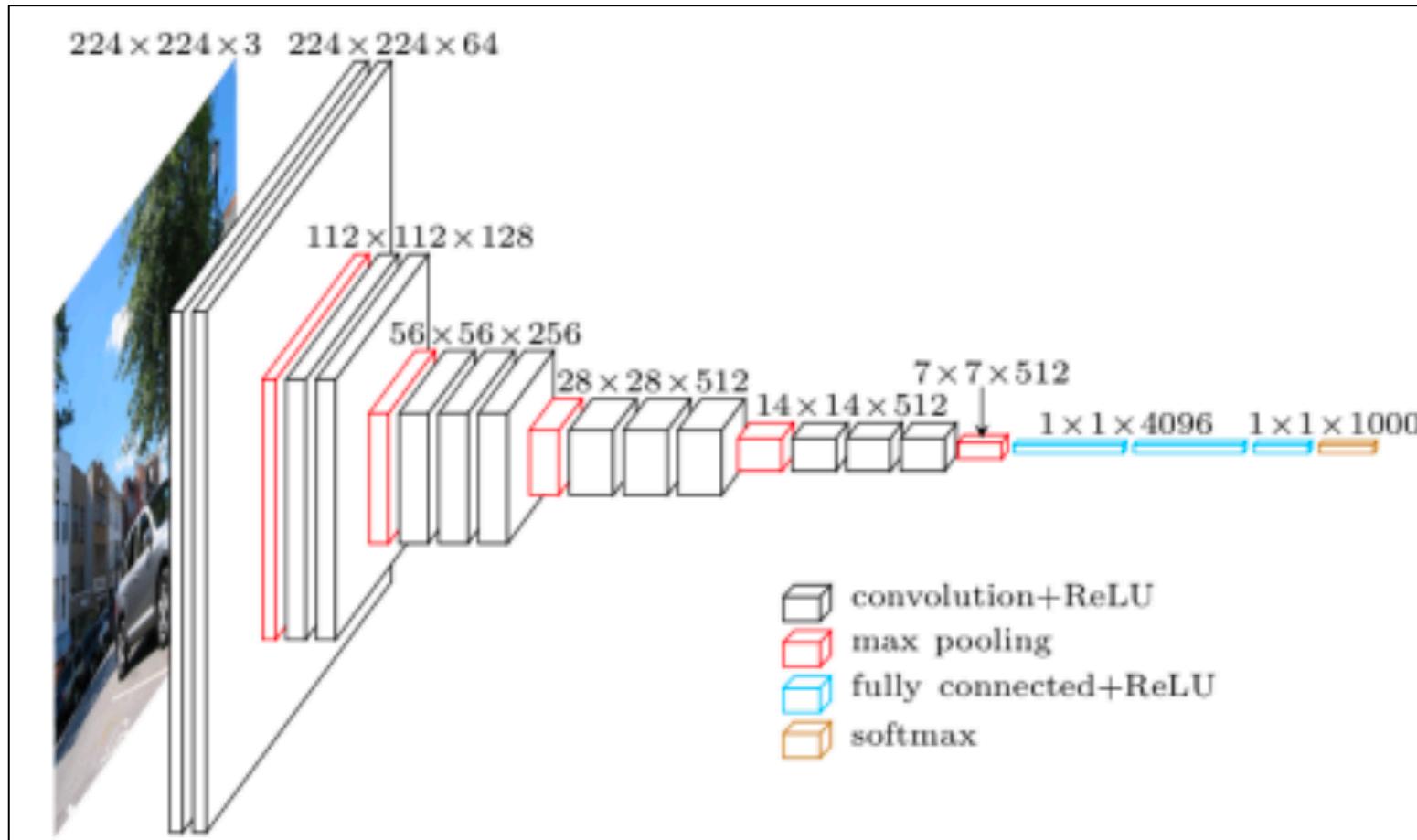
Activations of an example ConvNet architecture.

Initial volume stores raw image pixels (left) and the last volume stores class scores (right)  
 Each volume of activations along the processing path is shown as a column.  
 Since it is difficult to visualize 3D volumes, each volume's slices are laid out in rows

## VGG Net

- VGG is a convolutional neural network model
  - K. Simonyan and A. Zisserman, University of Oxford
  - “*Very Deep Convolutional Networks for Large-Scale Image Recognition*”
- The model achieves 93% top-5 test accuracy in ImageNet
  - which is a dataset of over 14 million images belonging to 1000 classes.

# VGG 16



Source: <https://www.cs.toronto.edu/~frossard/post/vgg16/>

# VGG-16 pre-trained model for Keras

`vgg-16_keras.py`

```

1  from keras.models import Sequential
2  from keras.layers.core import Flatten, Dense, Dropout
3  from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D
4  from keras.optimizers import SGD
5  import cv2, numpy as np
6
7  def VGG_16(weights_path=None):
8      model = Sequential()
9      model.add(ZeroPadding2D((1,1),input_shape=(3,224,224)))
10     model.add(Convolution2D(64, 3, 3, activation='relu'))
11     model.add(ZeroPadding2D((1,1)))
12     model.add(Convolution2D(64, 3, 3, activation='relu'))
13     model.add(MaxPooling2D((2,2), strides=(2,2)))
14
15     model.add(ZeroPadding2D((1,1)))
16     model.add(Convolution2D(128, 3, 3, activation='relu'))
17     model.add(ZeroPadding2D((1,1)))
18     model.add(Convolution2D(128, 3, 3, activation='relu'))
19     model.add(MaxPooling2D((2,2), strides=(2,2)))
20
21     model.add(ZeroPadding2D((1,1)))
22     model.add(Convolution2D(256, 3, 3, activation='relu'))
23     model.add(ZeroPadding2D((1,1)))
24     model.add(Convolution2D(256, 3, 3, activation='relu'))
25     model.add(ZeroPadding2D((1,1)))
26     model.add(Convolution2D(256, 3, 3, activation='relu'))
27     model.add(MaxPooling2D((2,2), strides=(2,2)))
28
29     model.add(ZeroPadding2D((1,1)))
30     model.add(Convolution2D(512, 3, 3, activation='relu'))
31     model.add(ZeroPadding2D((1,1)))

```

```

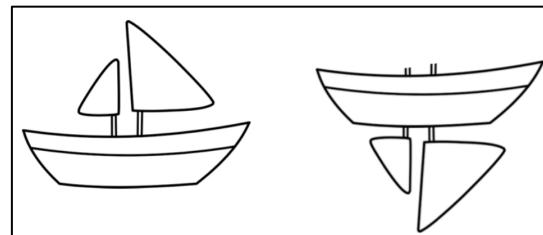
33     model.add(ZeroPadding2D((1,1)))
34     model.add(Convolution2D(512, 3, 3, activation='relu'))
35     model.add(MaxPooling2D((2,2), strides=(2,2)))
36
37     model.add(ZeroPadding2D((1,1)))
38     model.add(Convolution2D(512, 3, 3, activation='relu'))
39     model.add(ZeroPadding2D((1,1)))
40     model.add(Convolution2D(512, 3, 3, activation='relu'))
41     model.add(ZeroPadding2D((1,1)))
42     model.add(Convolution2D(512, 3, 3, activation='relu'))
43     model.add(MaxPooling2D((2,2), strides=(2,2)))
44
45     model.add(Flatten())
46     model.add(Dense(4096, activation='relu'))
47     model.add(Dropout(0.5))
48     model.add(Dense(4096, activation='relu'))
49     model.add(Dropout(0.5))
50     model.add(Dense(1000, activation='softmax'))
51
52     if weights_path:
53         model.load_weights(weights_path)
54
55     return model
56
57 if __name__ == "__main__":
58     im = cv2.resize(cv2.imread('cat.jpg'), (224, 224)).astype(np.float32)
59     im[:, :, 0] -= 103.939
60     im[:, :, 1] -= 116.779
61     im[:, :, 2] -= 123.68
62     im = im.transpose((2, 0, 1))
63     im = np.expand_dims(im, axis=0)
64
65     # Test pretrained model
66     model = VGG_16('vgg16_weights.h5')
67     sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
68     model.compile(optimizer=sgd, loss='categorical_crossentropy')
69     out = model.predict(im)
70     print np.argmax(out)

```

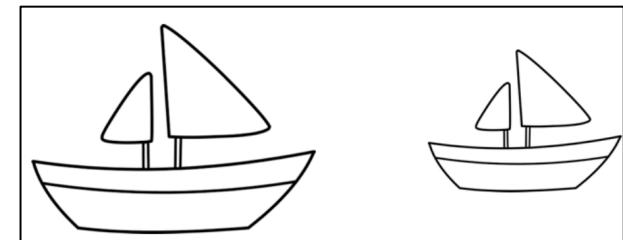
<https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3>

# Pooling, Invariance, Equivariance

- Pooling is supposed to obtain positional, orientational, proportional or rotational invariance.
  - But it is a very crude approach
    - In reality it removes all sorts of positional invariance
    - Leads to detecting right image in Fig. 1 as a correct ship



1. Disfiguration Transformation



2. Proportional Transformation

- Equivariance makes network understand the rotation or proportion change and adapt itself accordingly so that the spatial positioning inside an image is not lost
  - A solution is capsule nets

Source: <https://hackernoon.com/what-is-a-capsnet-or-capsule-network-2bfbe48769cc>

# Definition of Equivariance

- Generalizes the concept of invariance
- Invariance is a property which remains unchanged when transformations of a certain type are applied to the objects
  - Area and perimeter of a triangle are invariants
    - Translating/rotating a triangle does not change its area or perimeter
  - Triangle centers such as the centroid, circumcenter, are not invariant
    - Because moving a triangle will cause its centers to move
- Instead, these centers are equivariant
  - applying any Euclidean congruence (a combination of a translation and rotation) to a triangle, and then constructing its center, produces the same point as constructing the center first, and then applying the same congruence to the center.

$$f(g \bullet x) = g \bullet f(x)$$

O=Triangle;  $t$ =translation  
 $\text{Area}(O)=\text{Area}(t(O))$   
 $\text{Center}(O) \neq \text{Center}(t(O))$   
 $g(x) = \text{center}(x)$   
 $f(x) = \text{translation}(x)$

