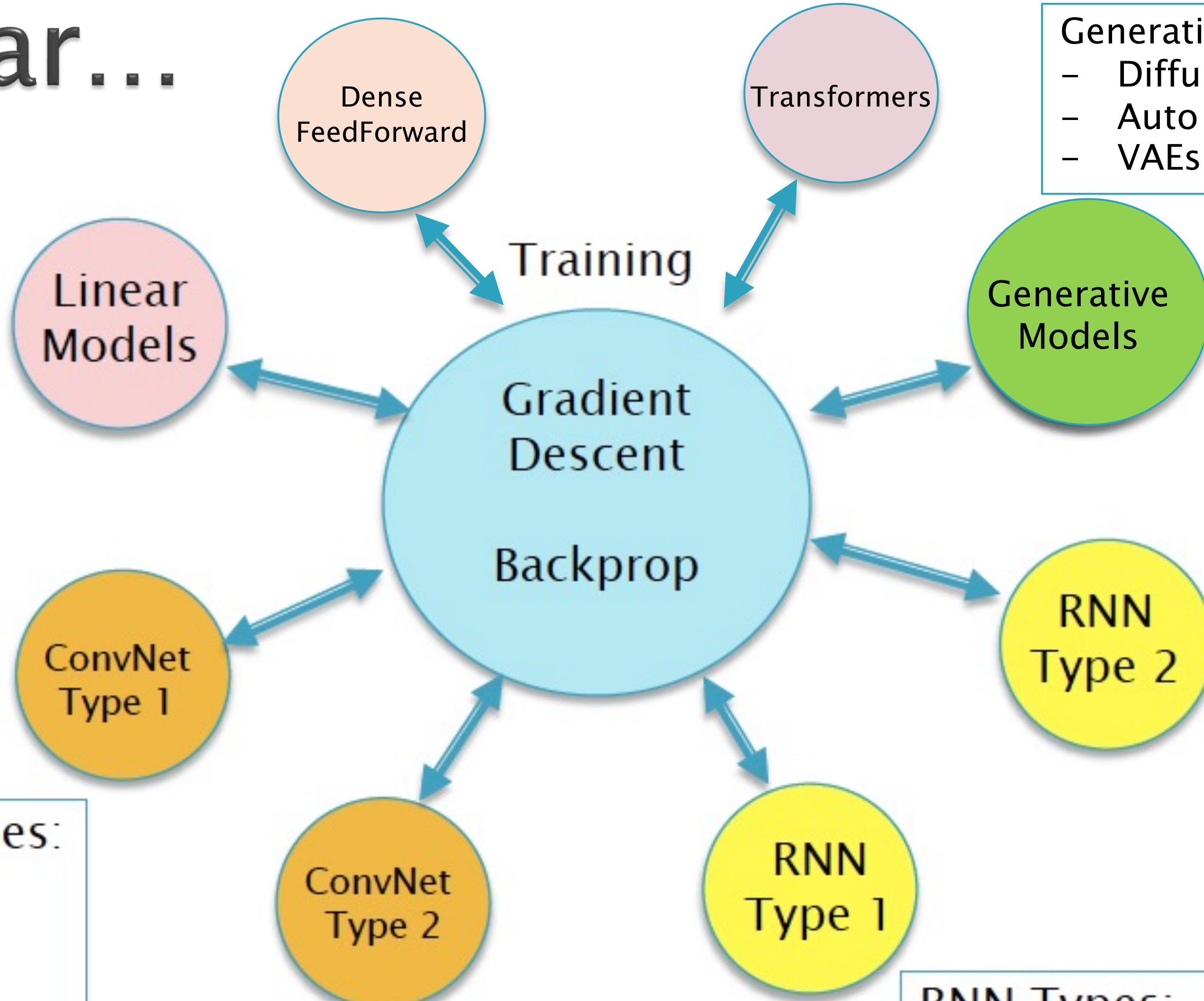
Convolutional Neural Networks: Part 1

Lecture 10

Subir Varma

So Far...



Generative Models:
- Diffusion Models
- Auto Regressive
- VAEs

ConvNet Types:
AlexNet
ResNet
InceptionNet

Combo RNNs+ConvNets

RNN Types:
Encoder Decoder RNNs
LSTMs
GRUs

A Keras Program

```
import keras
keras.__version__
```

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Import Dataset
(already in Tensor form)

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Data Reshaping
+
Data Normalization

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

Label Conversion from Sparse to Categorical (1-Hot Encoded)

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

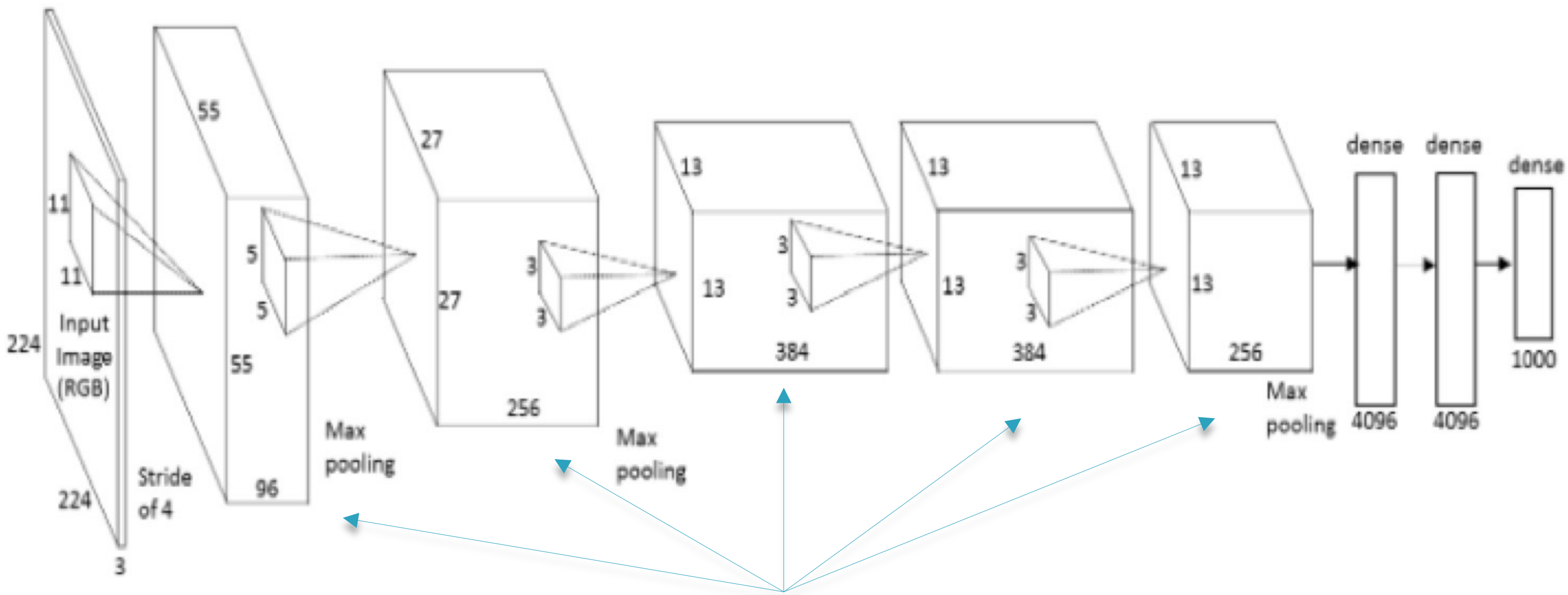
Define the Network

```
network.compile(optimizer='sgd',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Compile the Model

CNNs

- Can process images in their native 3D format
- Require much less parameters
- Have built in priors about the structure of images



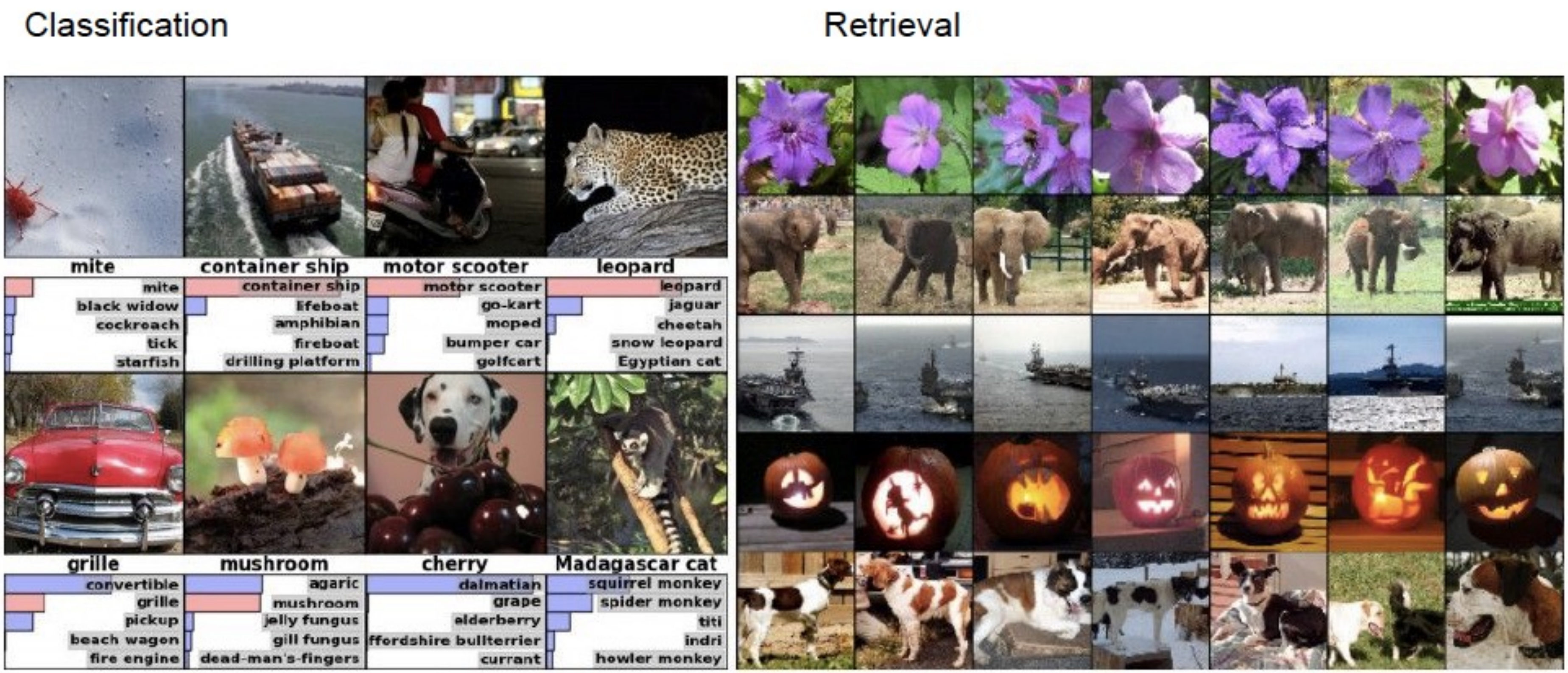
Input

Convolutional Layers

Fully Connected Layers

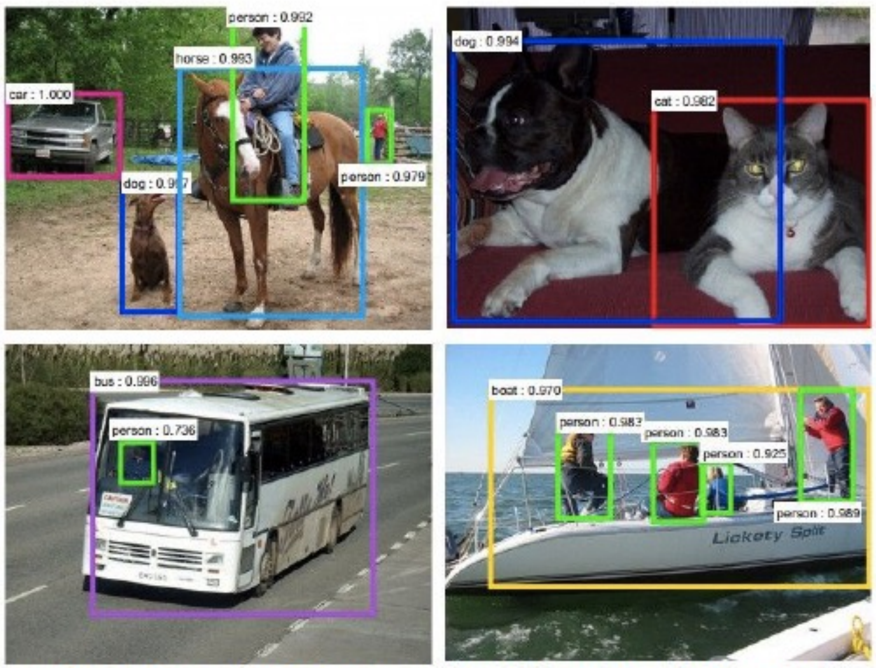
Applications

Google Photos, Google Image Search, YouTube, Video Filters in Camera Apps, Self Driving Cars, robotics, Medical Diagnosis, Game Playing Systems



Applications

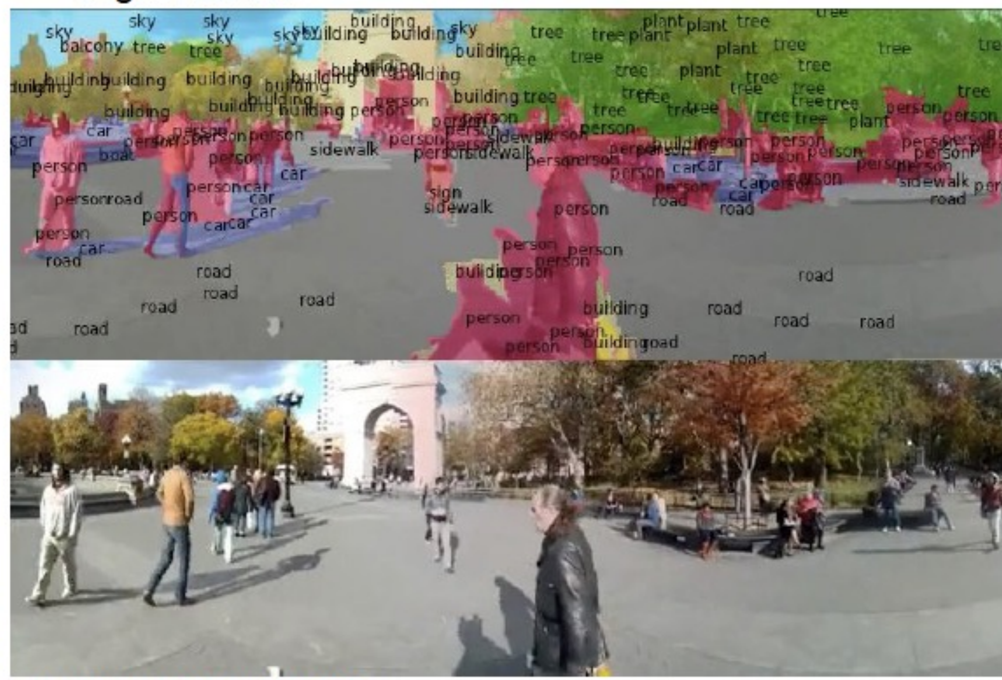
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Applications: Self Driving Cars

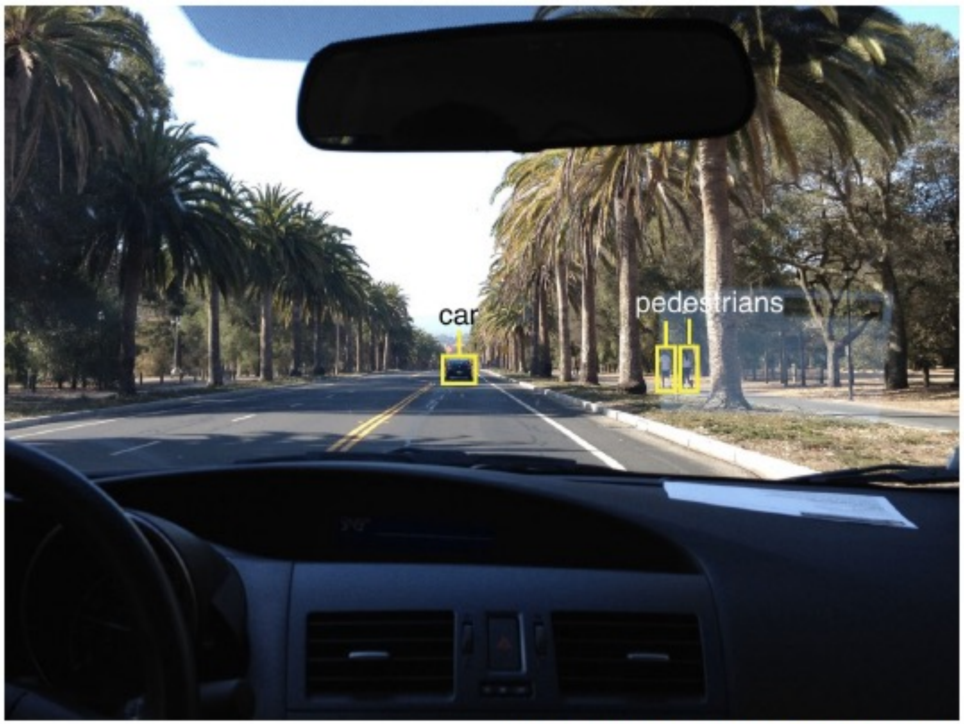


Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Applications: Image Captioning

No errors

Minor errors

Somewhat related

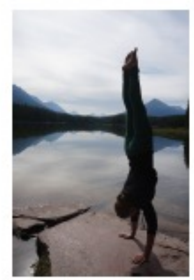
A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor



A woman standing on a beach holding a surfboard

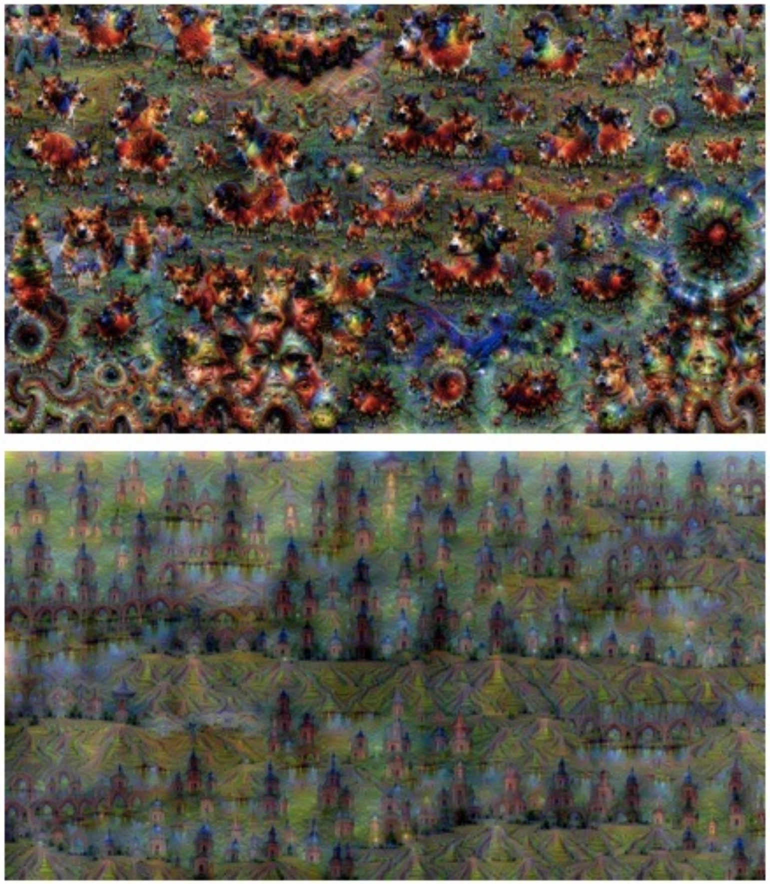
Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

Captions generated by Justin Johnson using Neuraltalk2

Applications: Image Generation

Deep Dream



Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.

Neural Style Transfer

Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Generating Images from Captions

A stop sign is flying in blue skies.

A herd of elephants flying in the blue skies.

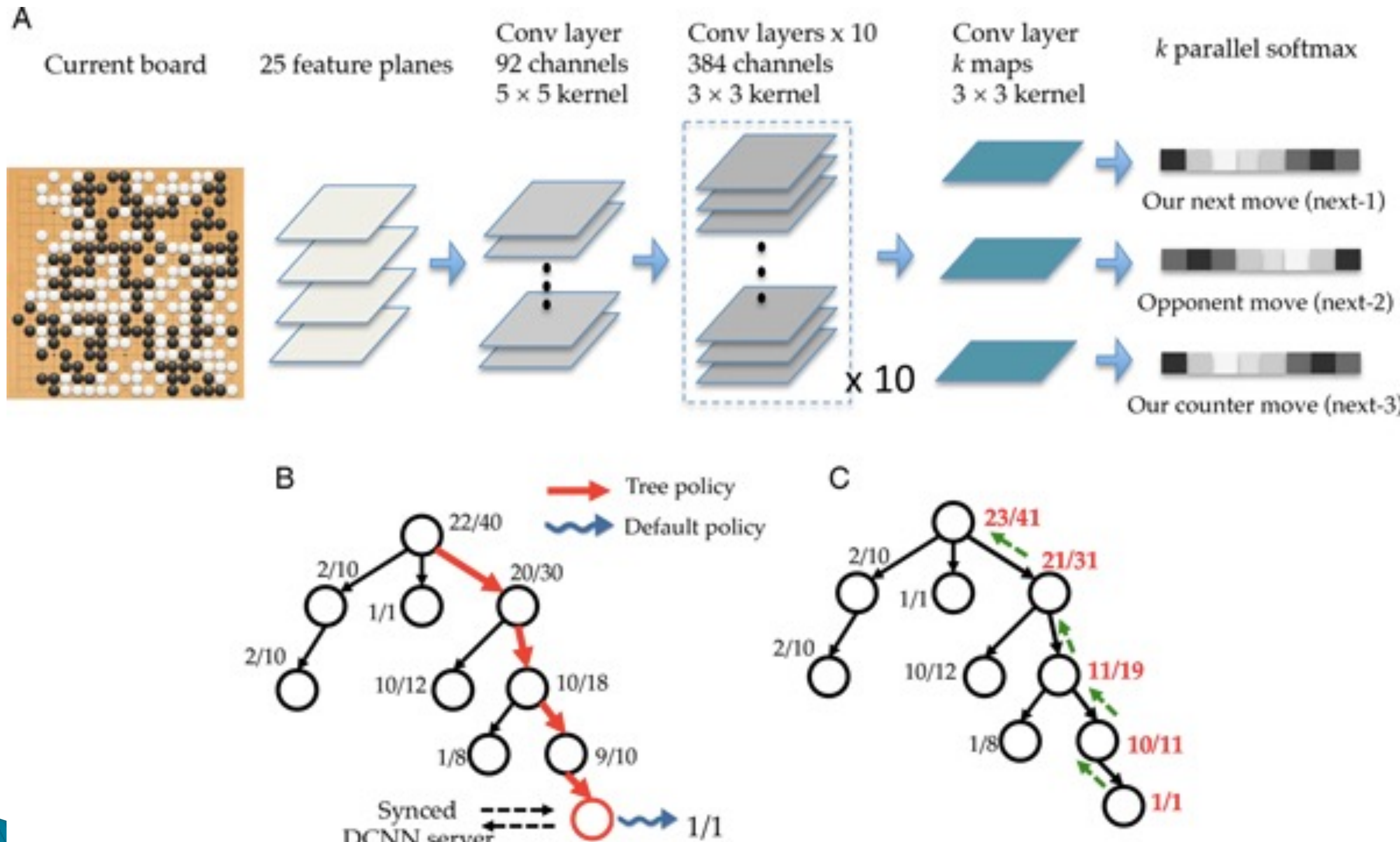
A toilet seat sits open in the grass field.

A person skiing on sand clad vast desert.

Figure 1: Examples of generated images based on captions that describe novel scene compositions that are highly unlikely to occur in real life. The captions describe a common object doing unusual things or set in a strange location.

arXiv:1511.02793v2 [cs.LG] 29 Feb 2016

Playing Go using CNNs



CNN Architecture

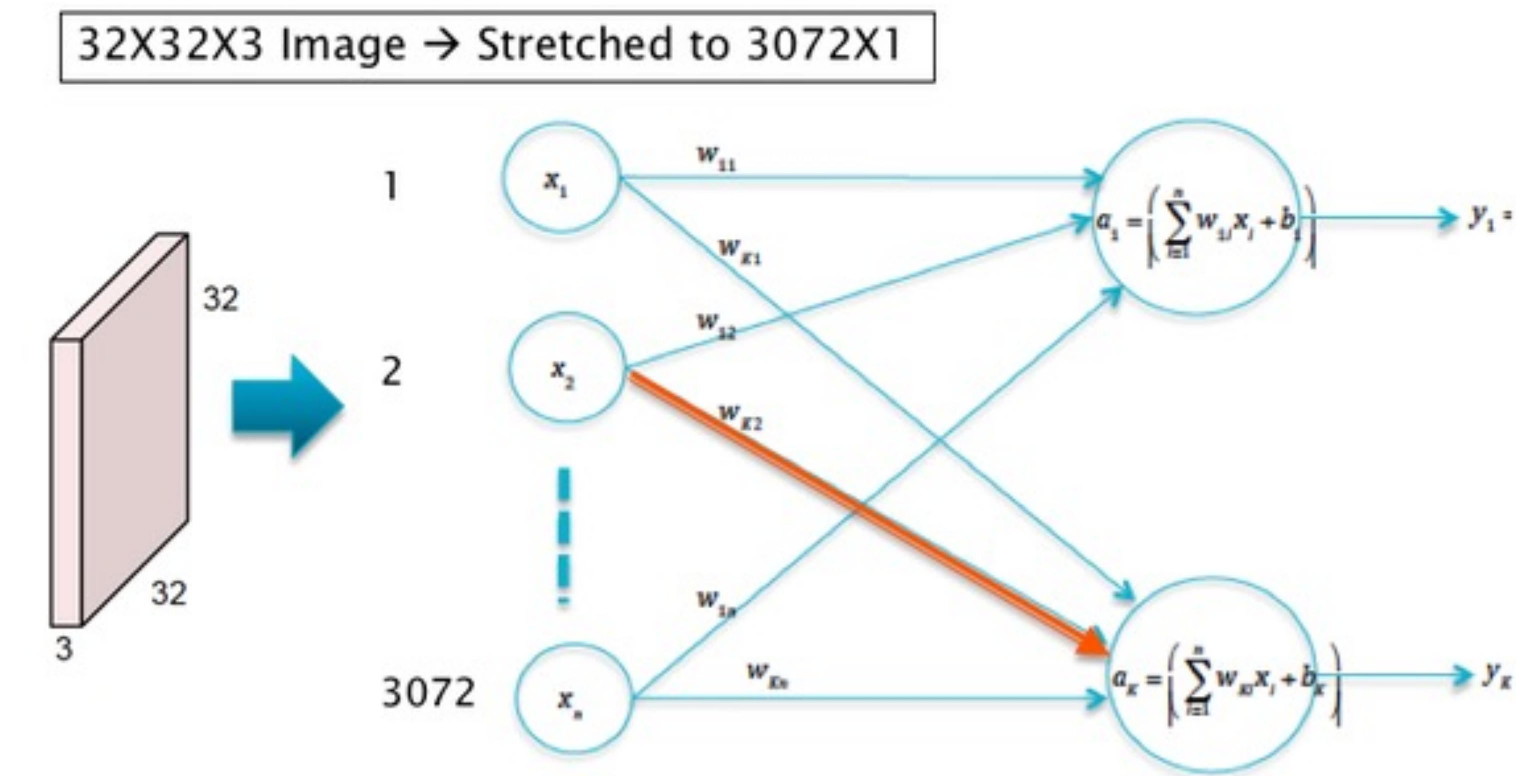
Why are Dense FeedForward Networks not Optimal for Images

- Consider a typical image consisting of 200×200×3 pixels, which corresponds to 3 layers of 200×200 numbers, one for each color Red, Green and Blue.
  Hence the input consists of 120,000 numbers

- Given a typical dense feedforward network with 100 nodes in the first hidden layer, this corresponds to 12 million weight parameters needed to describe just this layer.

The Parameter Explosion Problem

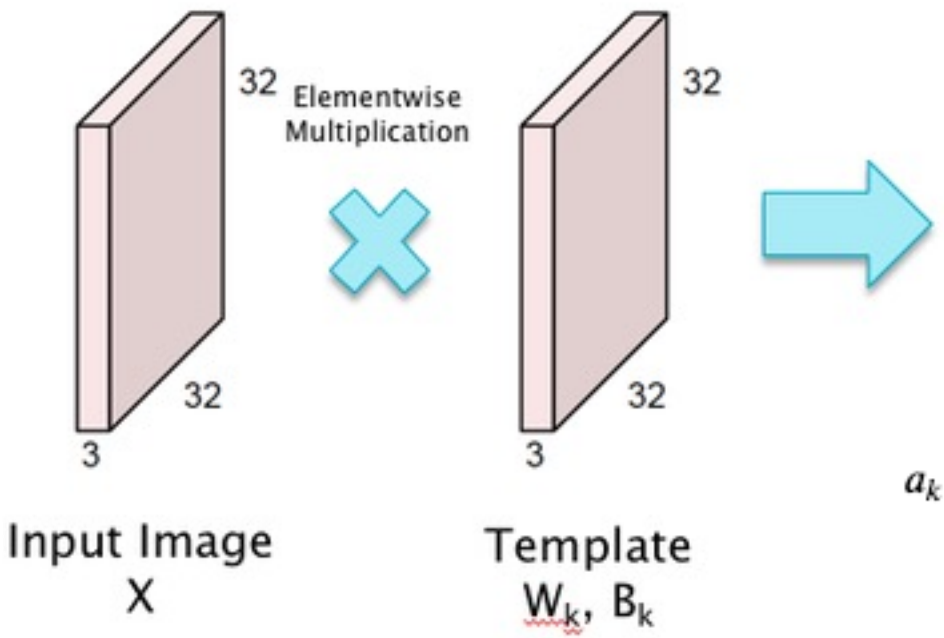
K-ary Linear Model with CIFAR-10 Input



Flattening causes loss of structural information from the image

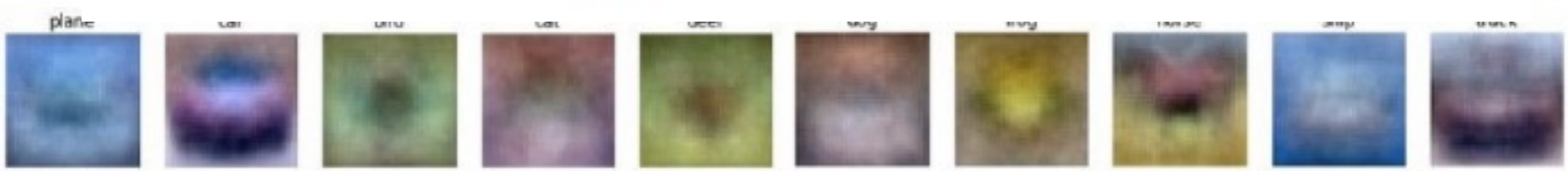
Interpretation of Weights as a Filter – Template Matching

10 Templates



Choose Category k for which $X^*W_k + B_k$ is maximum

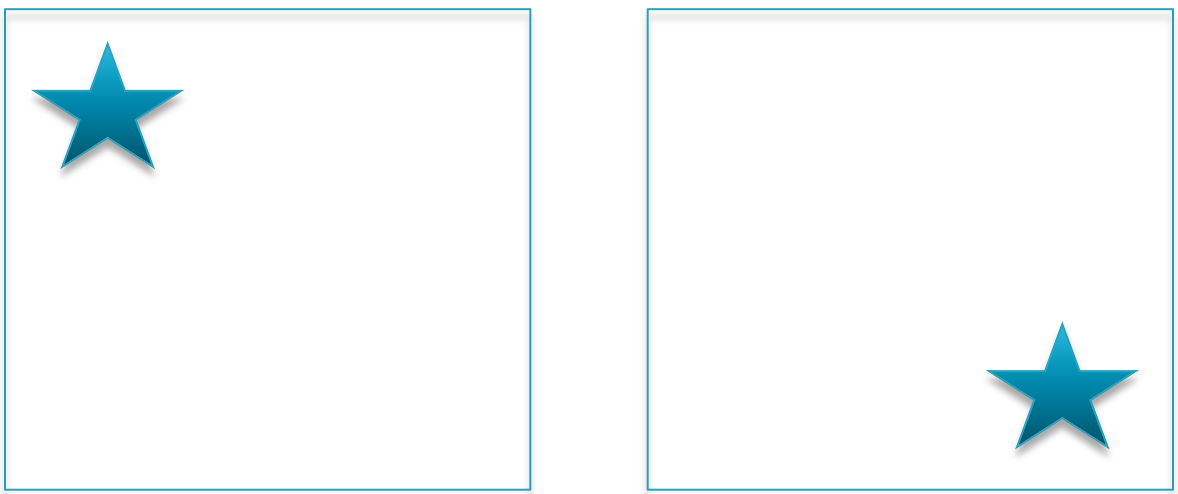
$$a_k = \sum_{i=1}^{3072} w_{ki} x_i + b_k, \quad 1 \leq k \leq 10$$



Issues with the One Filter Model

- Trying to detect the whole object with a single filter
- Too many parameters
- Lack of translational invariance

CNNs Solve All These Problems

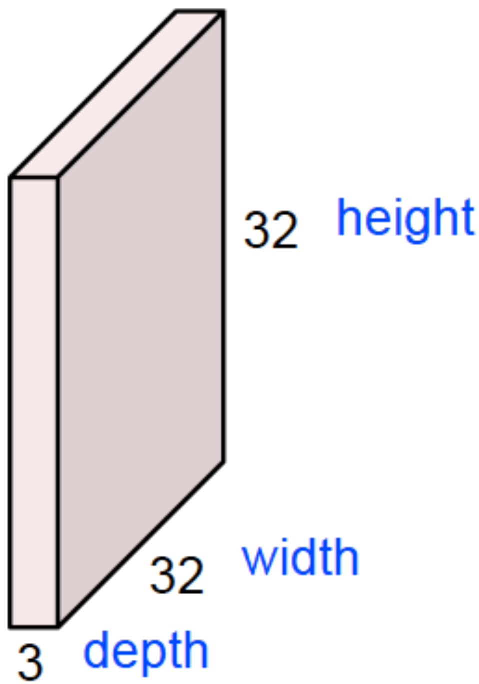


Need two different filters to detect these two objects

Build in the prior that a pattern remains the same irrespective of where it is located

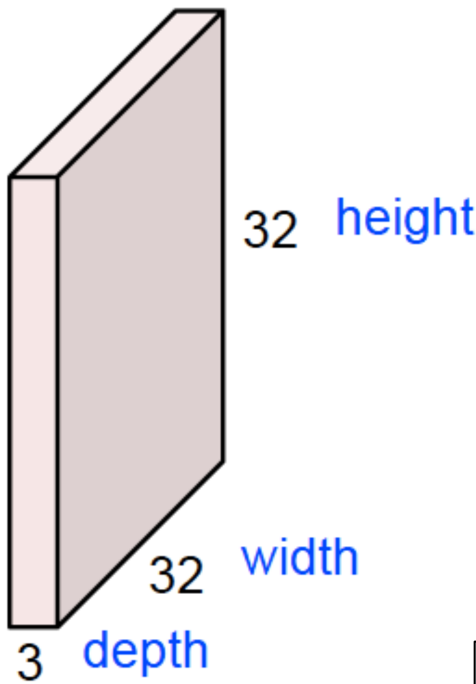
Step 1: Preserve the Spatial Structure of the Input Image

32x32x3 image -> preserve spatial structure

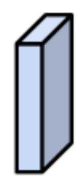


Step 2: Use Smaller Filter

32x32x3 image -> preserve spatial structure



5x5x3 filter

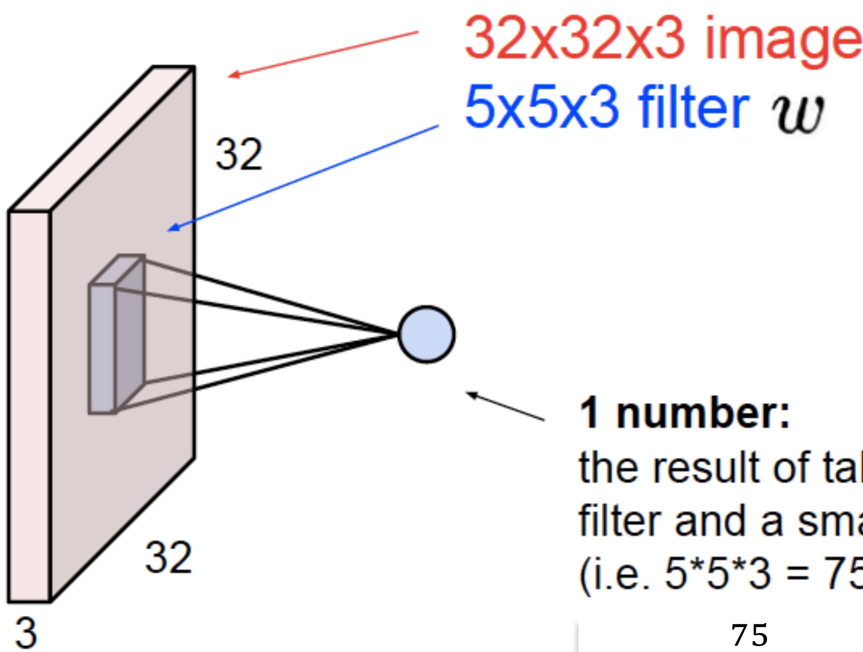


$$a = \sum_{i=1}^{75} w_i x_i + b$$

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

Local Filtering

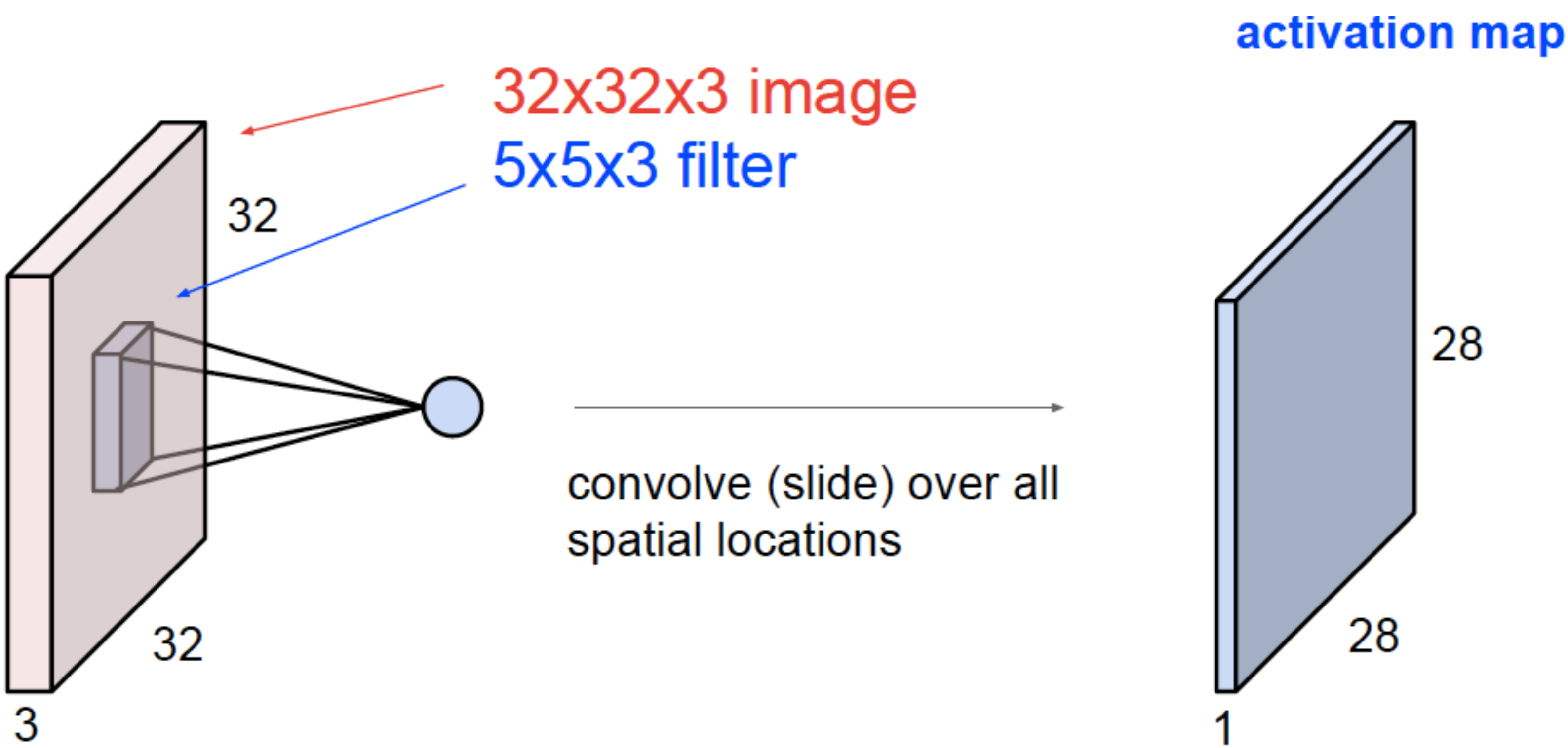
Step 3: Take Dot Product of Filter with a 3-D Chunk of the Input



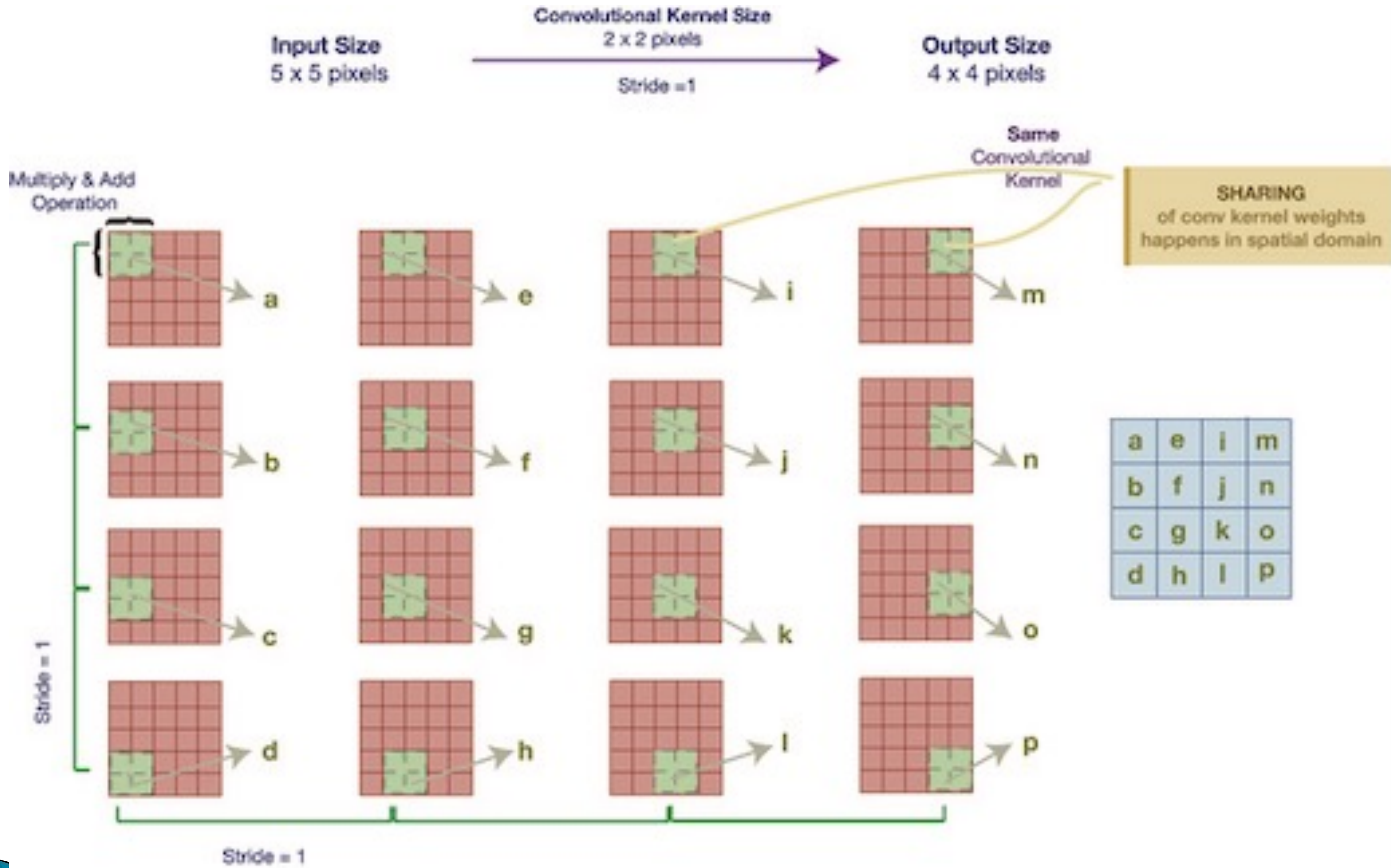
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$a = \sum_{i=1}^{75} w_i x_i + b$$

Step 4: Slide Filter all Over Image (Convolution Operation)



Stride =1



Input Size
5 x 5 pixels
Convolutional Kernel Size
2 x 2 pixels
Stride =1
Output Size
4 x 4 pixels
Same
Convolutional
Kernel
SHARING
of conv kernel weights
happens in spatial domain
Multiply & Add
Operation
a
e
i
m
b
f
j
n
c
g
k
o
d
h
l
p
Stride = 1
Stride = 1

Benefits

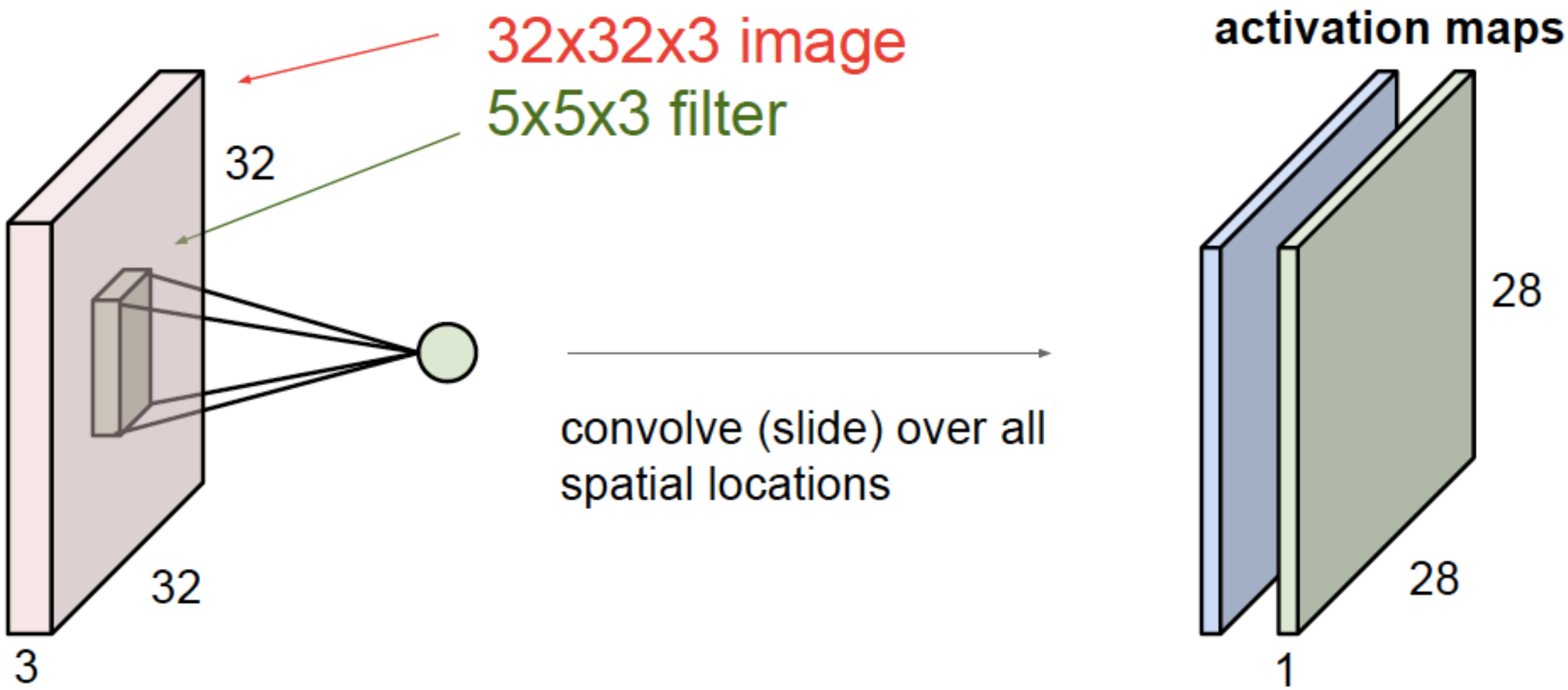
- Translational Invariance
  - Since the same Filter is used at all locations in the image, CNNs are able to detect a pattern irrespective of where it occurs in the image
- Reduction in Number of Parameters
  - Instead of 32*32*3+1 = 3073 parameters, need only

    5*5*3+1 = 76 parameters!!

Results in Higher Model Capacity

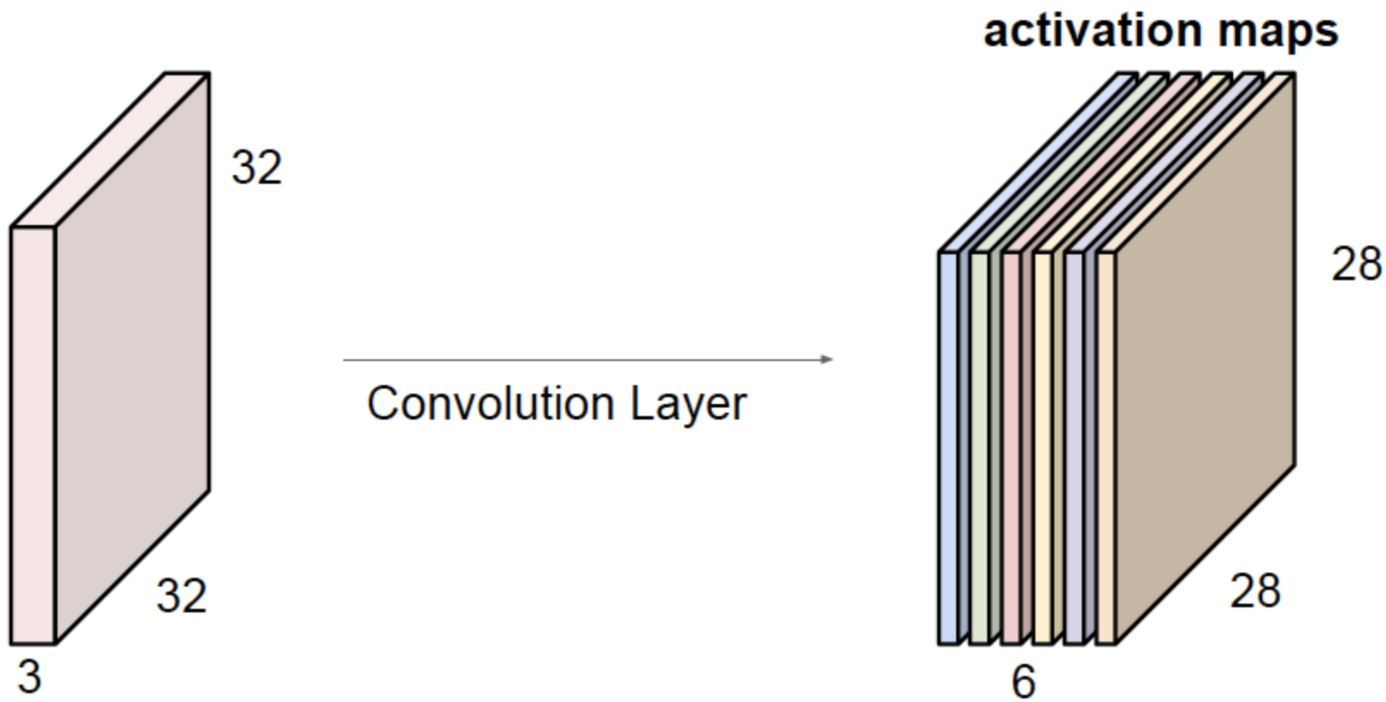
Multiple Activation Maps

To Detect Multiple Shapes!



Construction of Multiple Layers

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



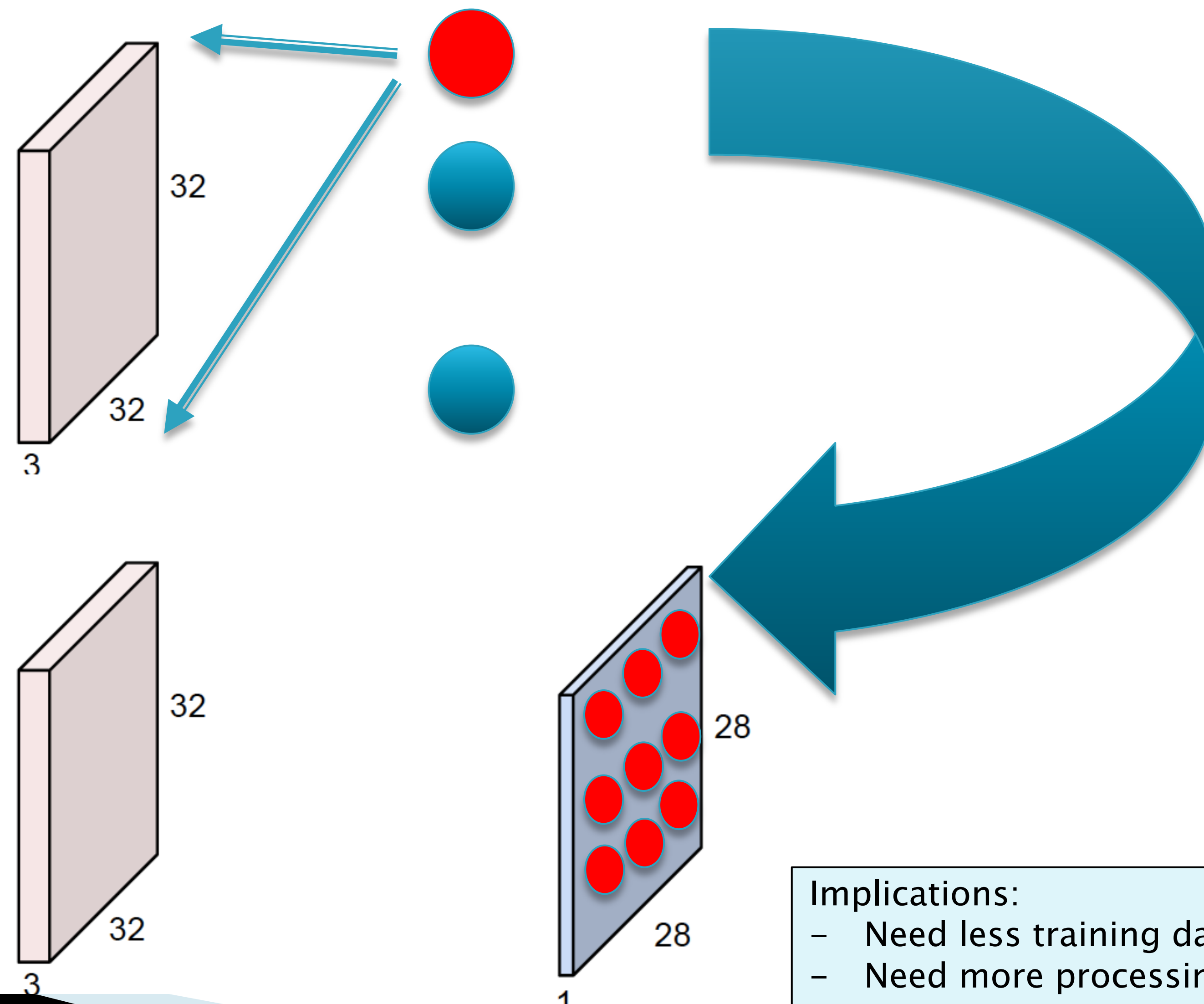
We stack these up to get a "new image" of size 28x28x6!

How many Activation Maps needed?

Node Expansion

Results in more nodes but less parameters!

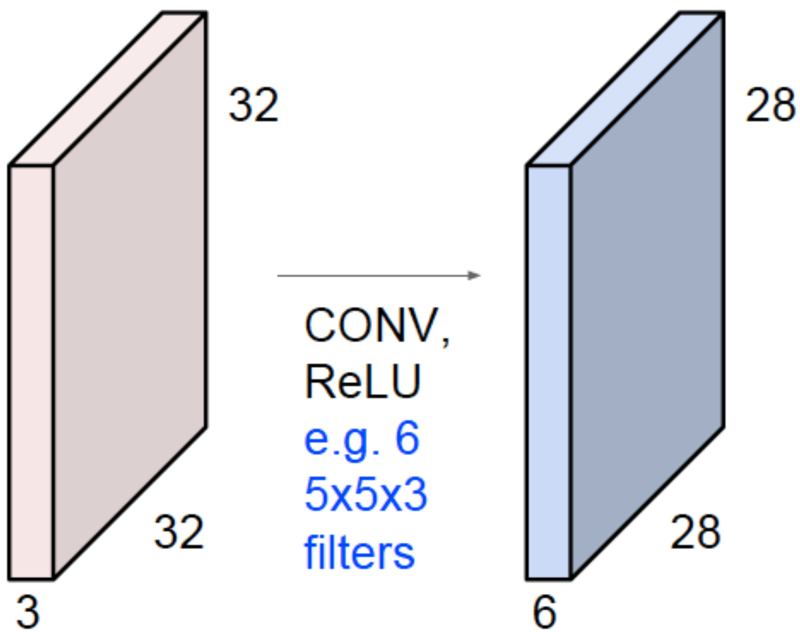
A Node in the Dense FF Architecture turns into an Activation Map in a ConvNet



Implications:
- Need less training data
- Need more processing

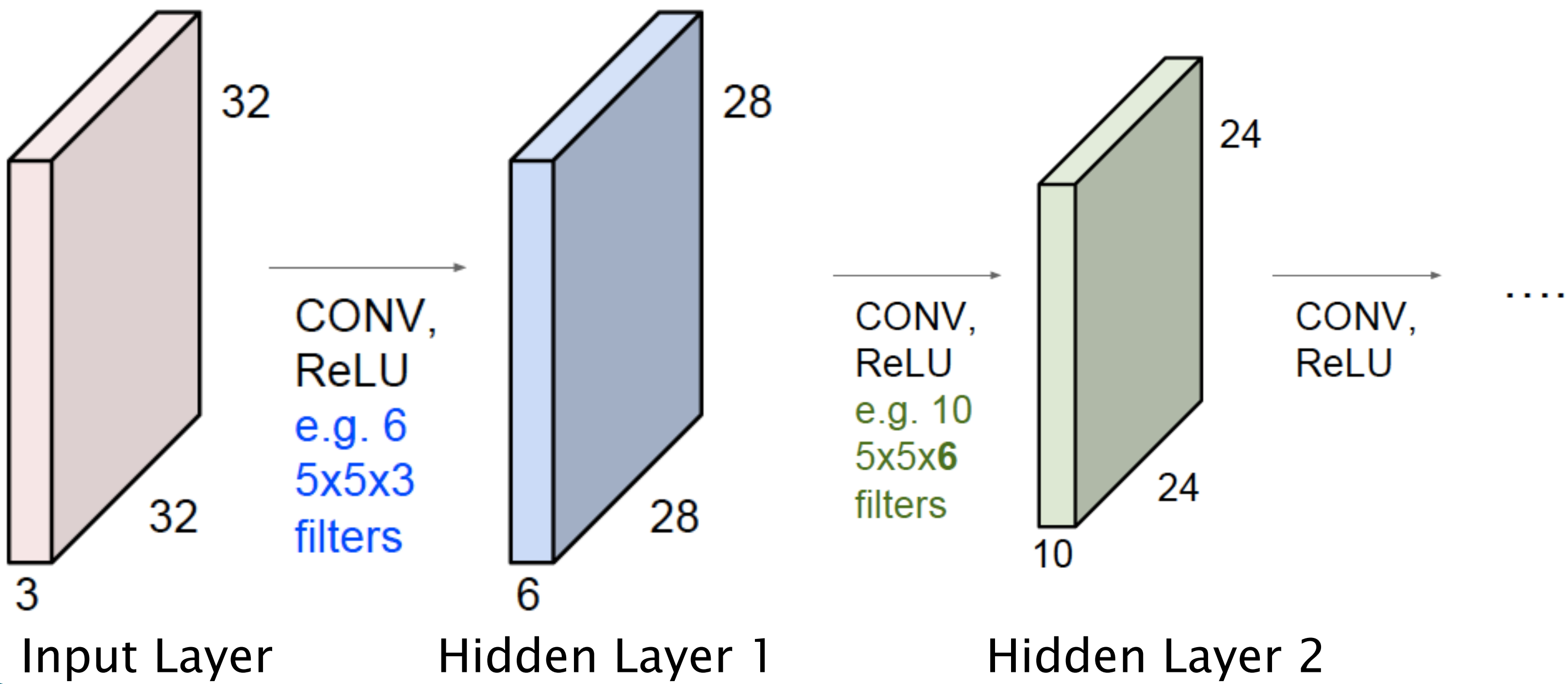
Two Convolutional Layers

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

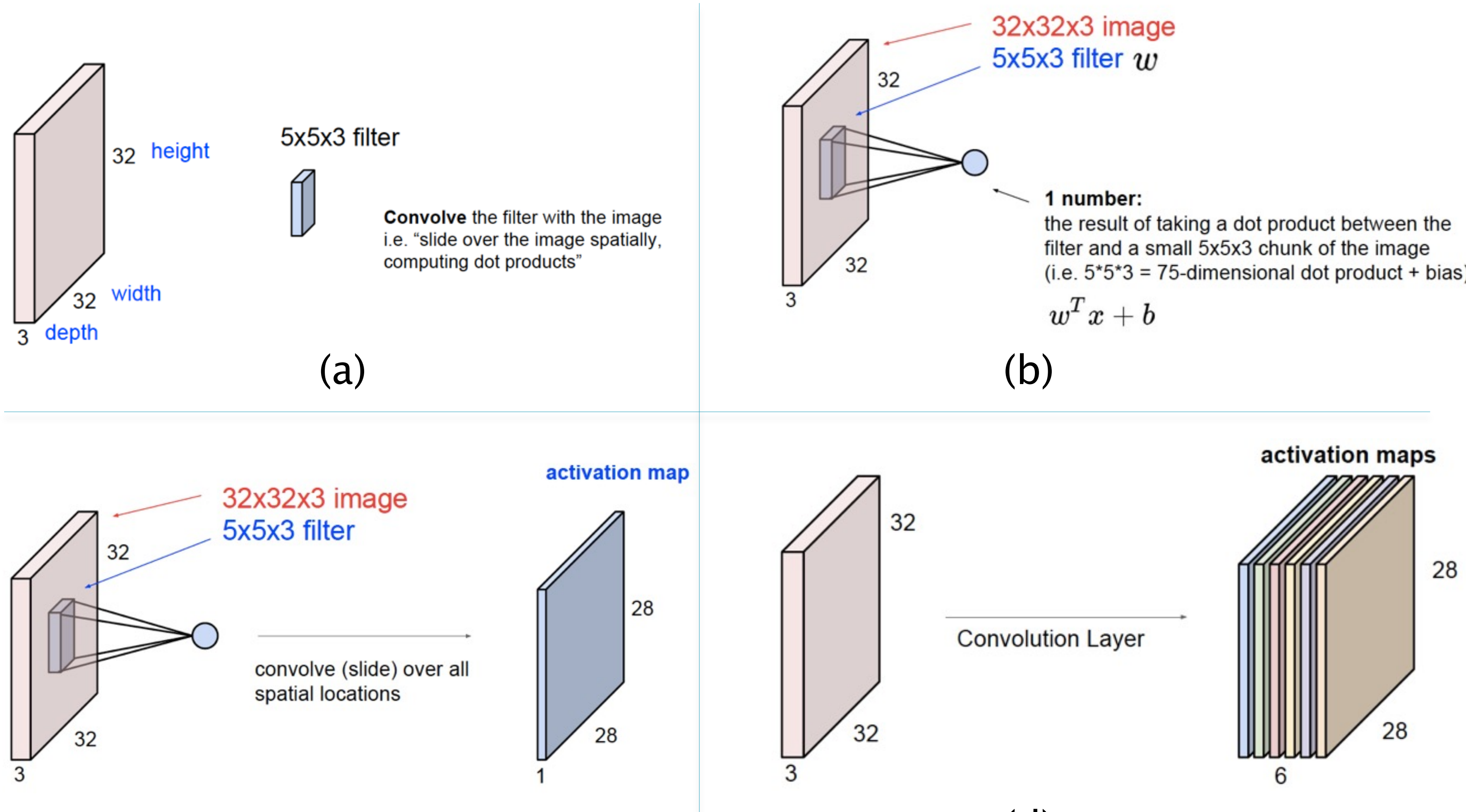


Three Convolutional Layers

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

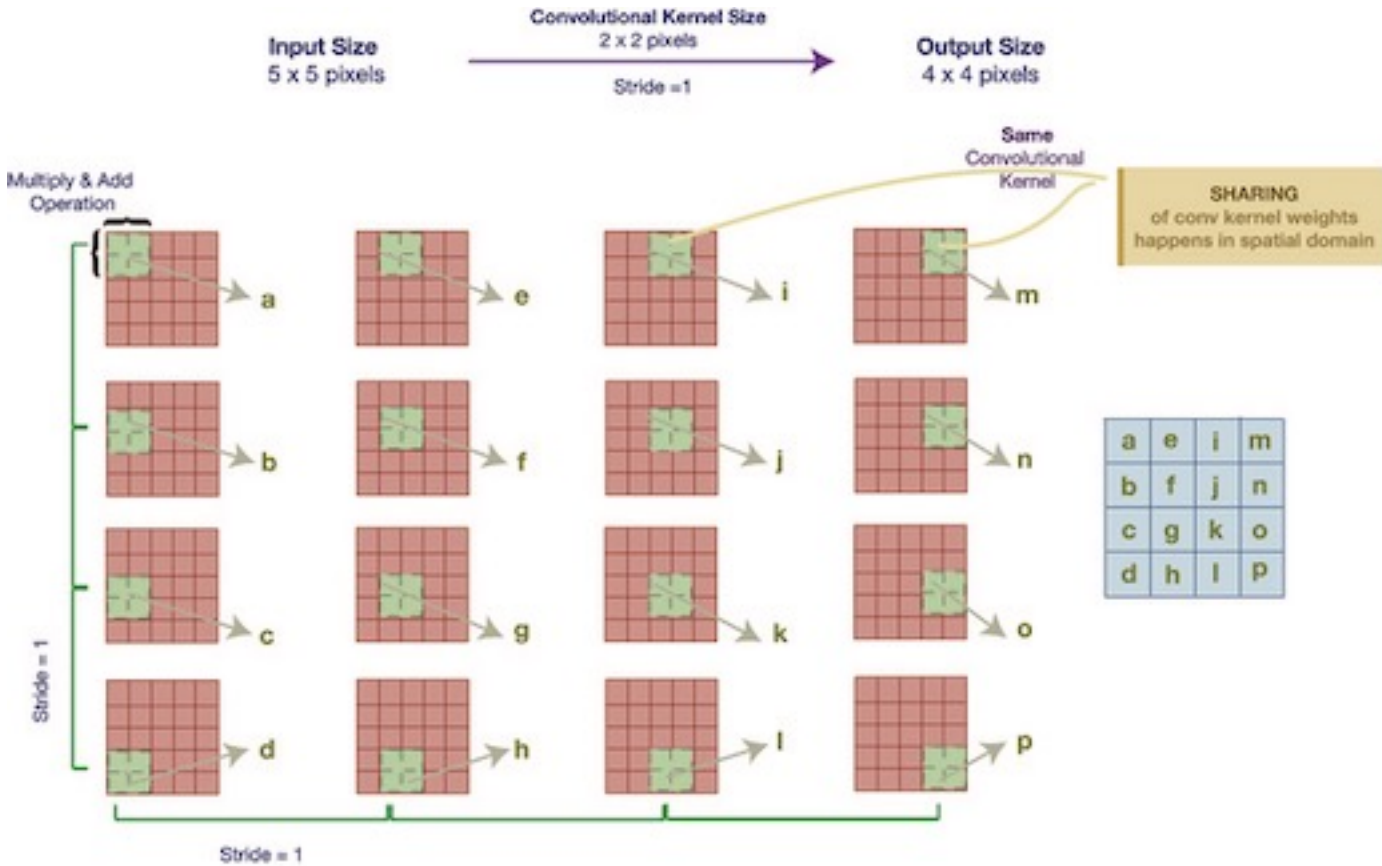


Summary



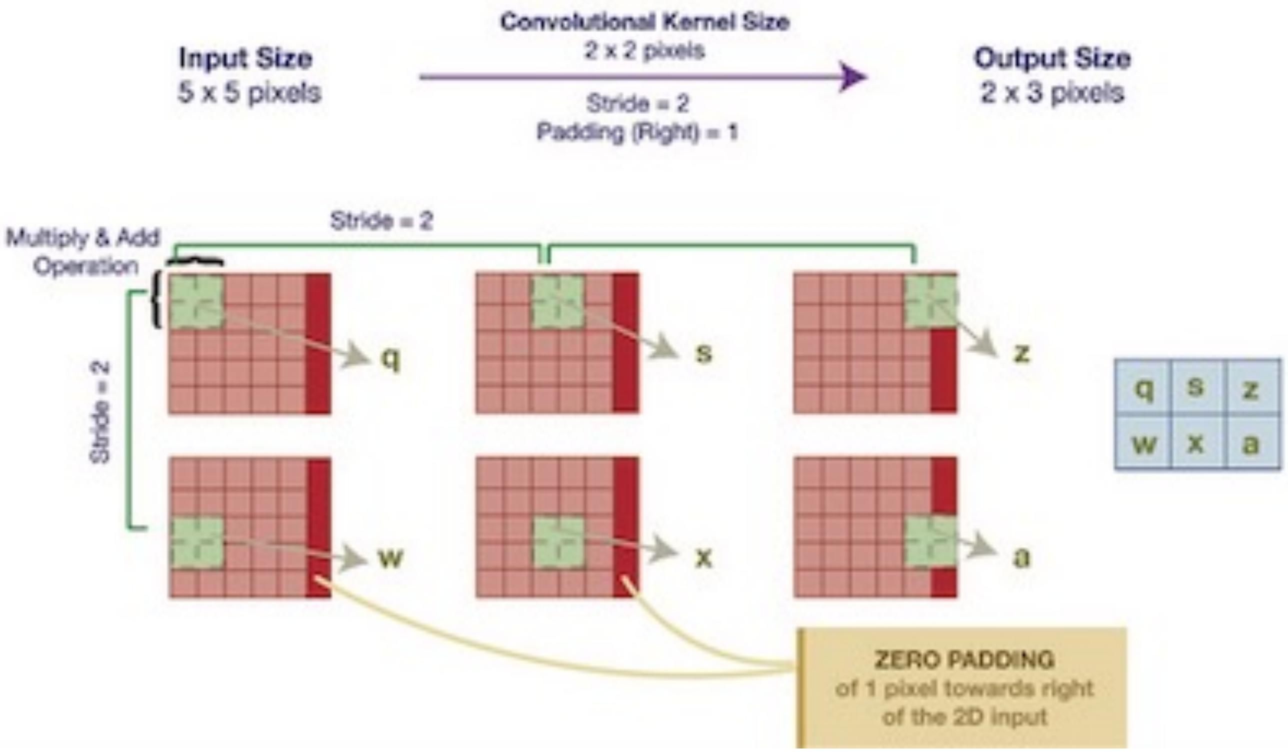
(a) (b) (c) (d)

Stride =1



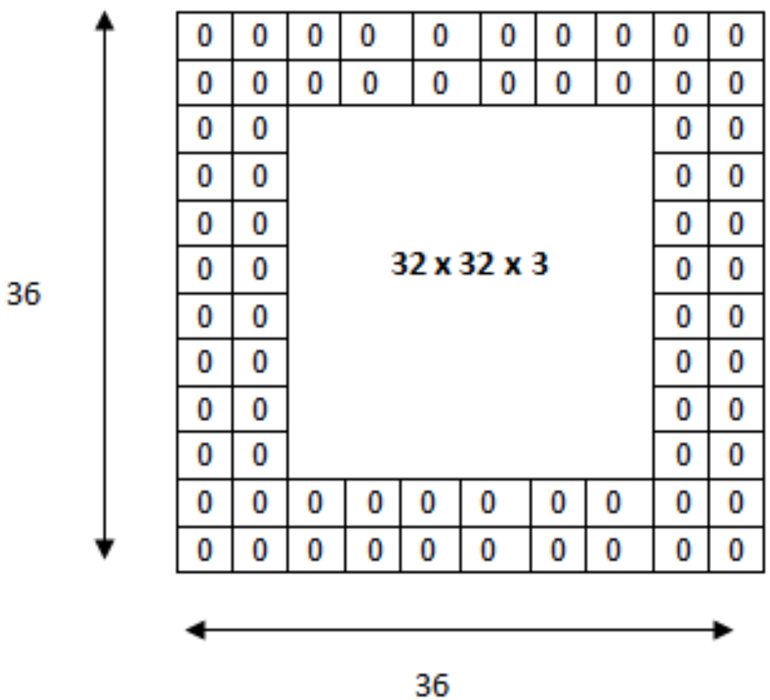
Convolutional Kernel Size
2 x 2 pixels
Stride =1
Input Size
5 x 5 pixels
Output Size
4 x 4 pixels
Same
Convolutional
Kernel
SHARING
of conv kernel weights
happens in spatial domain
Multiply & Add
Operation
a
e
i
m
b
f
j
n
c
g
k
o
d
h
l
p
Stride = 1
Stride = 1

Stride = 2



Input Size
5 x 5 pixels
Convolutional Kernel Size
2 x 2 pixels
Stride = 2
Padding (Right) = 1
Output Size
2 x 3 pixels
Multiply & Add
Operation
Stride = 2
Stride = 2
q
s
z
w
x
a
q s z
w x a
ZERO PADDING
of 1 pixel towards right
of the 2D input

Zero Padding

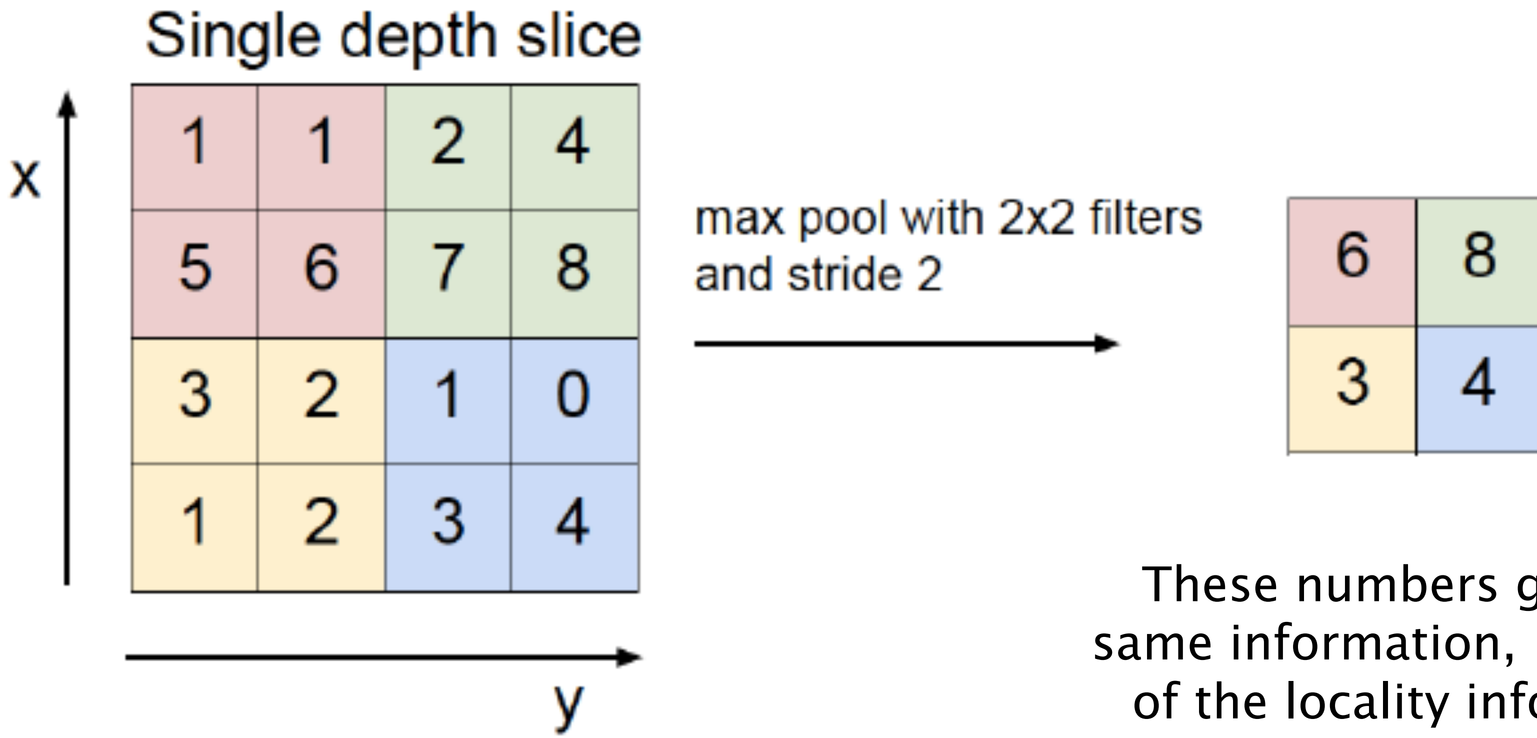


Zero Padding =1 → 34 X 34 X 3

Zero Padding = 2 → 36 X 36 X 3

Pooling

Max Pooling

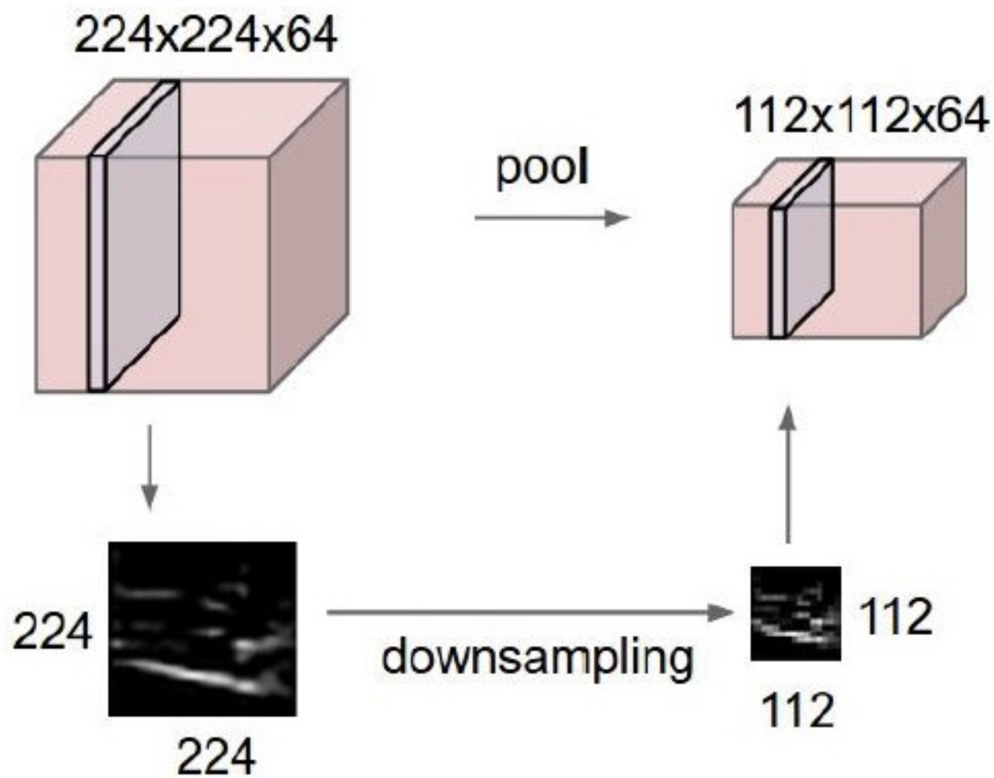


These numbers give the same information, but some of the locality info is lost

These Numbers tell us whether a pattern is present at the 16 locations

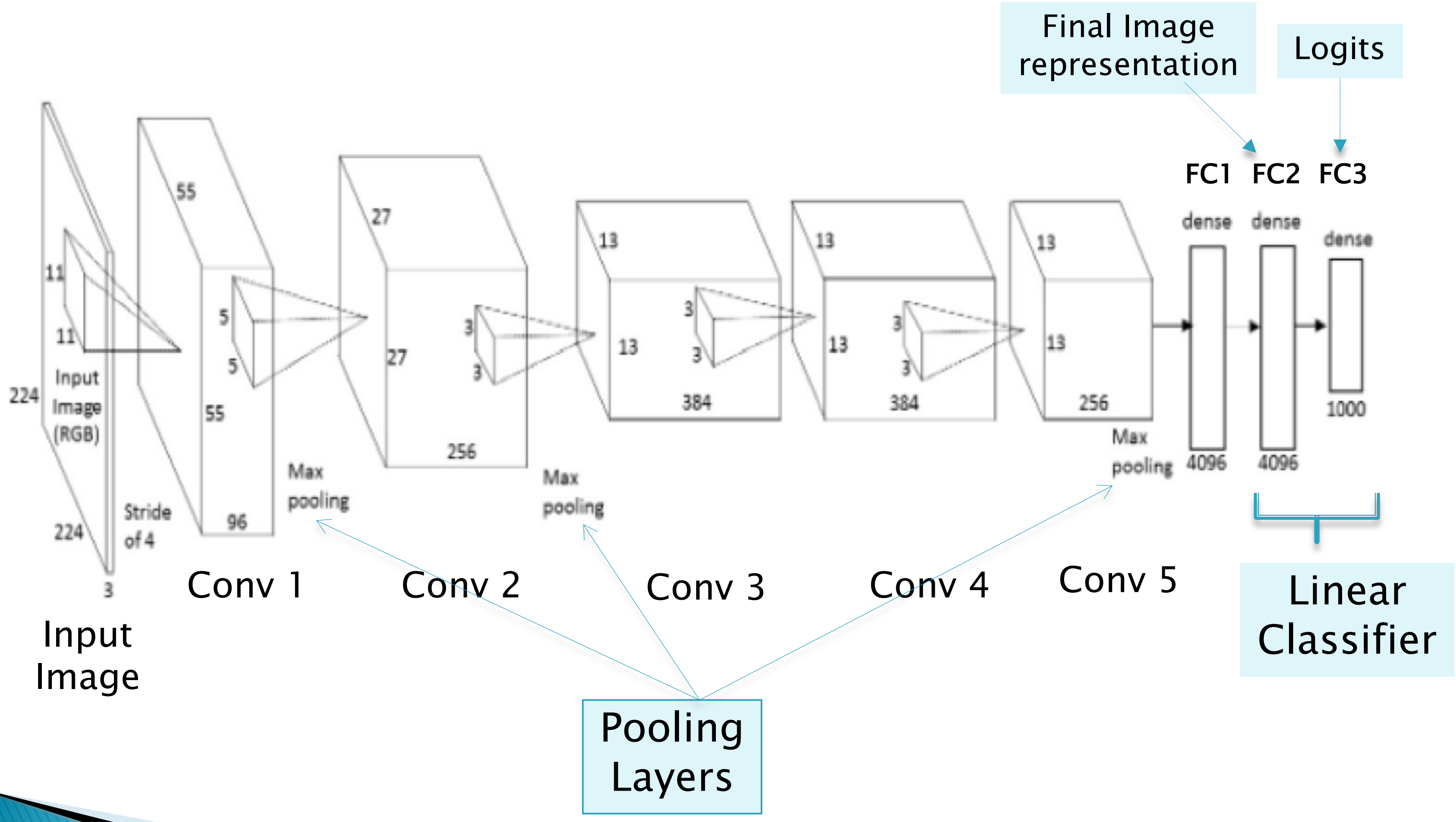
Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:

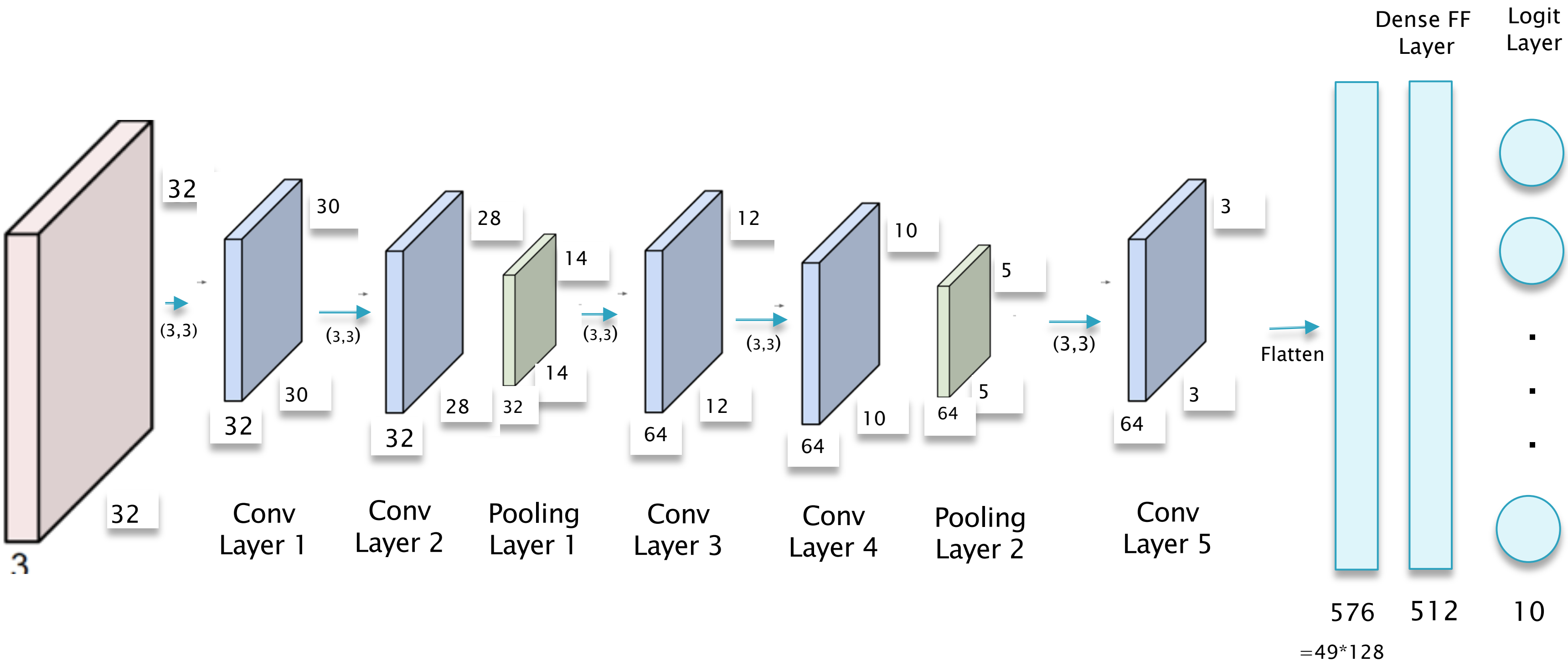


No Additional Parameters Needed!

A Complete CNN: AlexNet (2012)



CNNs in Keras



$$3*3*32 = 864 + 32 = 896$$

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

ConvNets in Keras

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 74, 74, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 36, 36, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 17, 17, 128)       0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584    
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776   
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513       
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
_________________________________________________________________
```

Further Reading

- Chapters 12: ConvNets Part 1
- Chollet: Chapter 8, Section 8.1