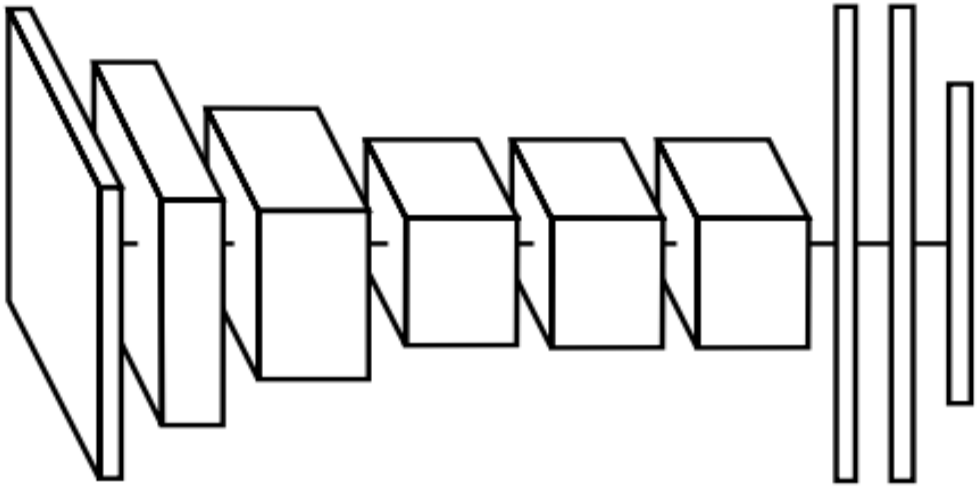
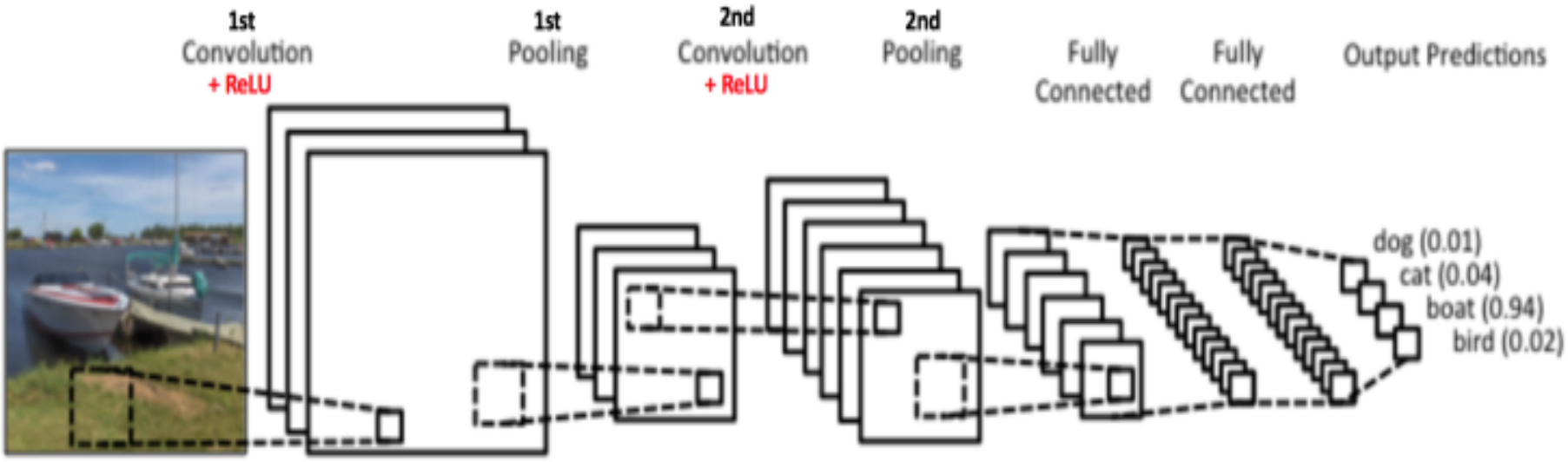
# Convolutional Neural Network(CNN)



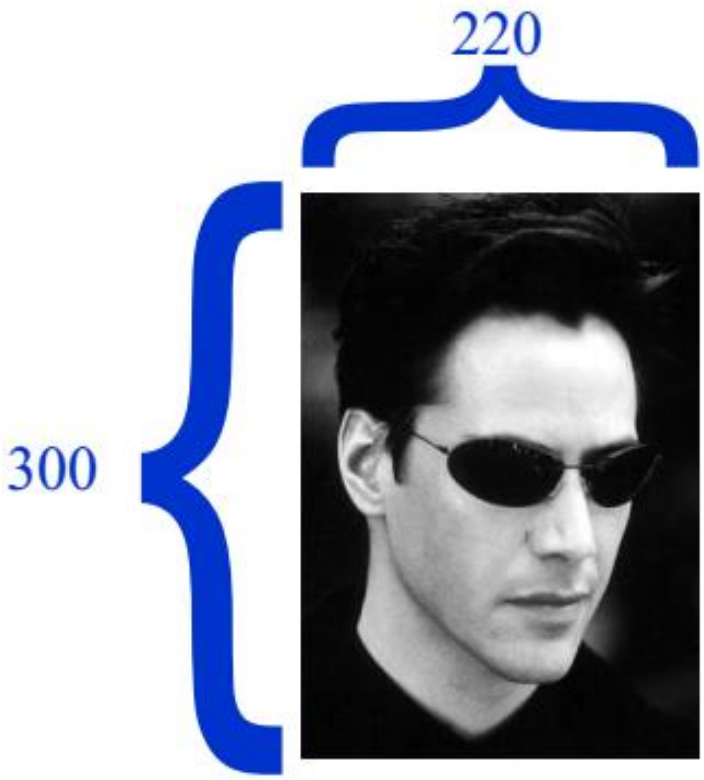
# Our goal: recognize the image as boat



# What is an image?

# Grey image

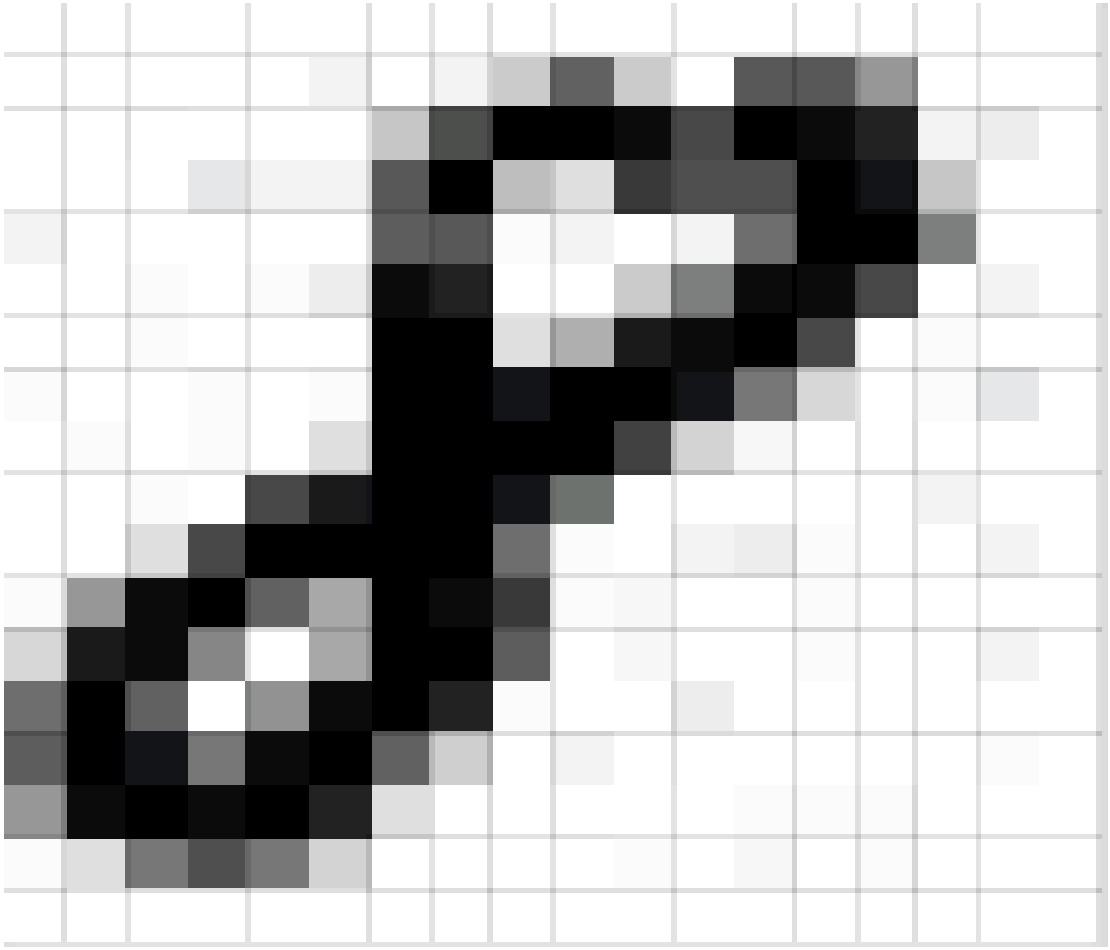
- A grid of numbers (*intensity values*)
- In computing, a *matrix*



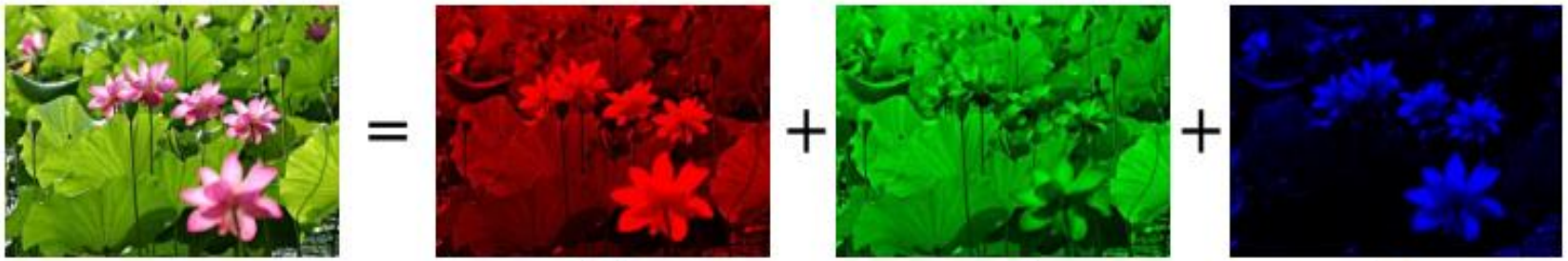

```
[ 10  30  40  106 123 … ;
   8  49  58  112 145 … ;
  16  53  86  123 152 … ;
                  …      ]
```

300 x 220 matrix

# Digit 8 in grey scale



# Color image



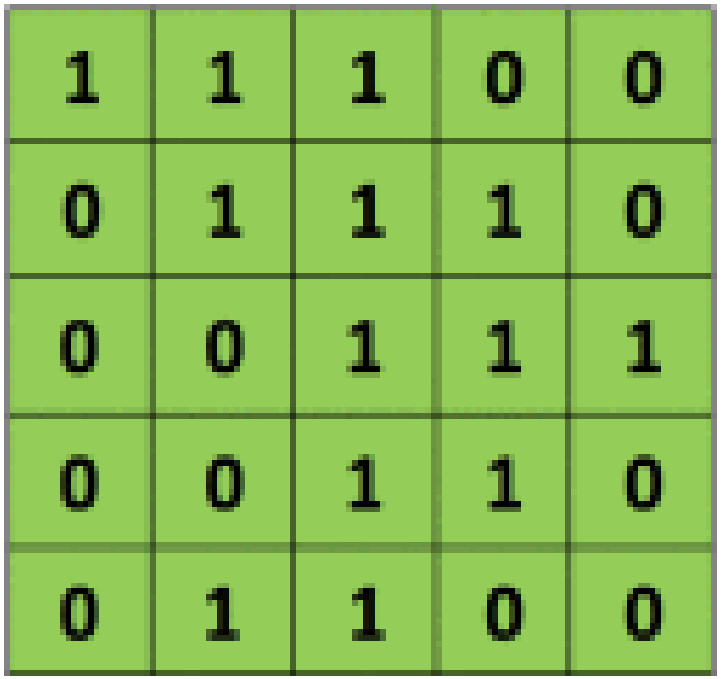
# Convolution

# Extracting features from images

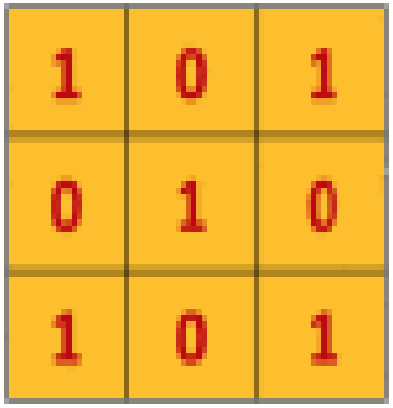
- Extract features by applying a sliding window function across image, called as convolution

- Why sliding window?
  - Detect the same feature at different positions in the input image

# How do you apply convolution across image?

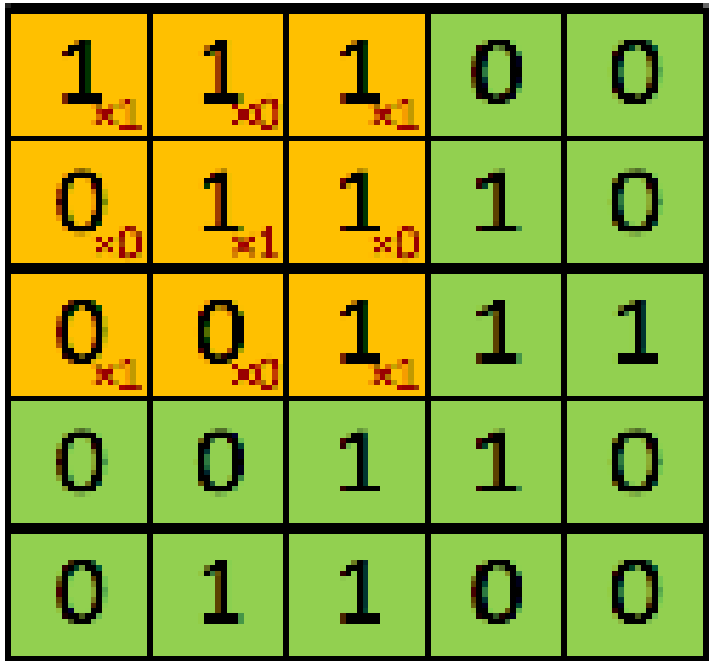



image

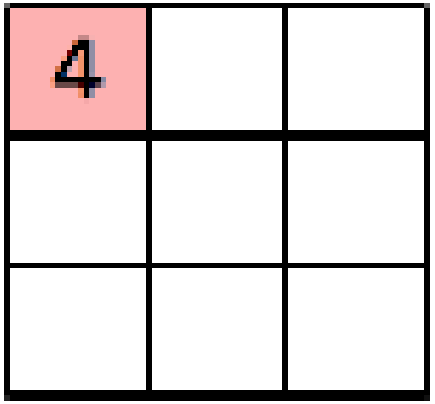



Convolutional filter

# How do you apply convolution across image?




Image




Feature map

# Example1

Different filters (weights) reveal a different characteristics of the input.

$*^{1/8}$

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 4 | 1 |
| 0 | 1 | 0 |

# Example1

Different filters (weights) reveal a different characteristics of the input.

$*$

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

Different filters (weights) reveal a different characteristics of the input.

$*$

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

# Example2: Feature maps

- The convolution of a filter1 (with red outline) slides over the input image to produce a feature map.

- The convolution of another filter2 (with the green outline), over the same image gives a different feature map

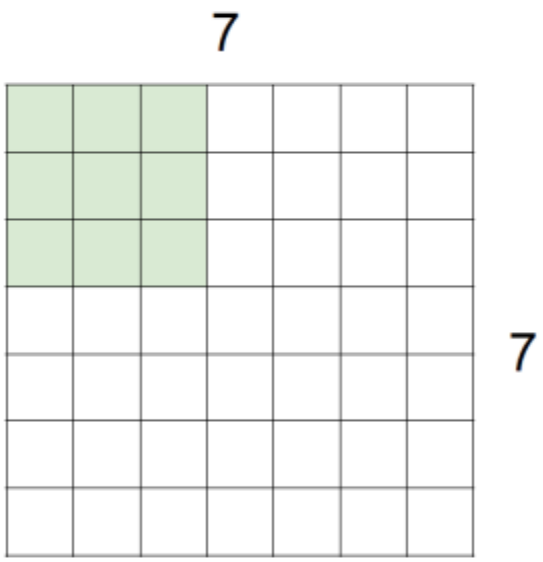
# Parameters for convolutional filters

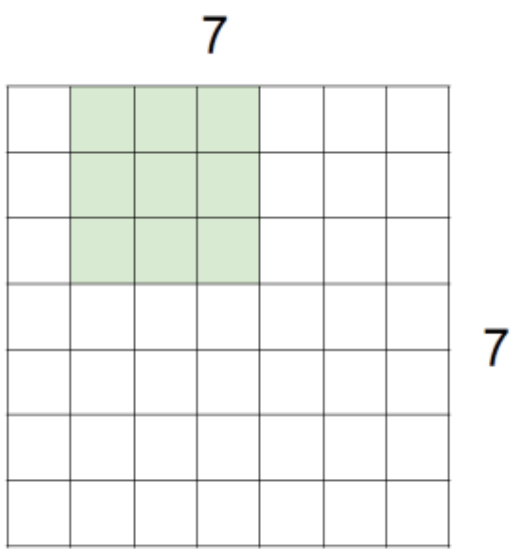
# Parameters for convolutional filters

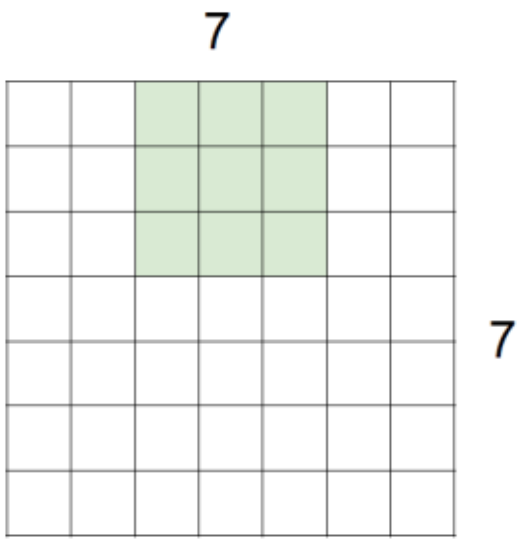
The size of the feature maps are controlled by following parameters of convolution filters:

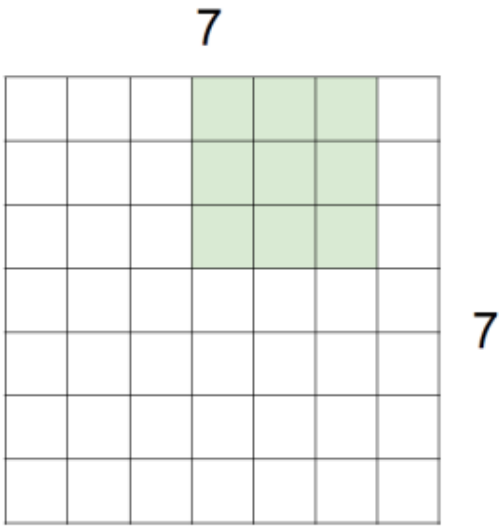
- Depth: It corresponds to the number of filters we use for the convolution operation
- Stride: It is the number of pixels by which we slide our filter matrix over the input matrix
- Padding: Adding zeros around the border, so that we can apply the filter to bordering elements of our input image matrix

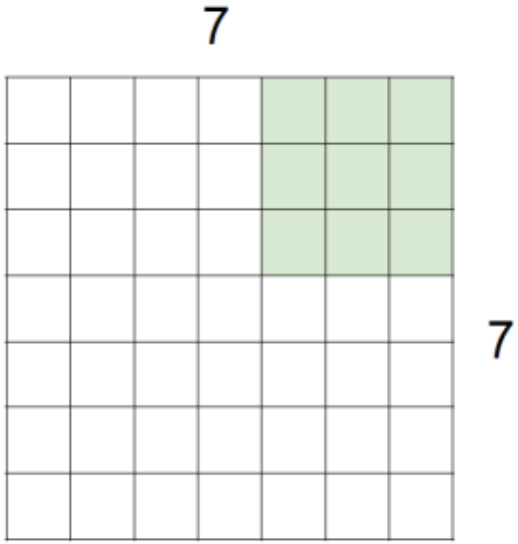
# 7 by 7 input, 3 by 3 filter with stride 1



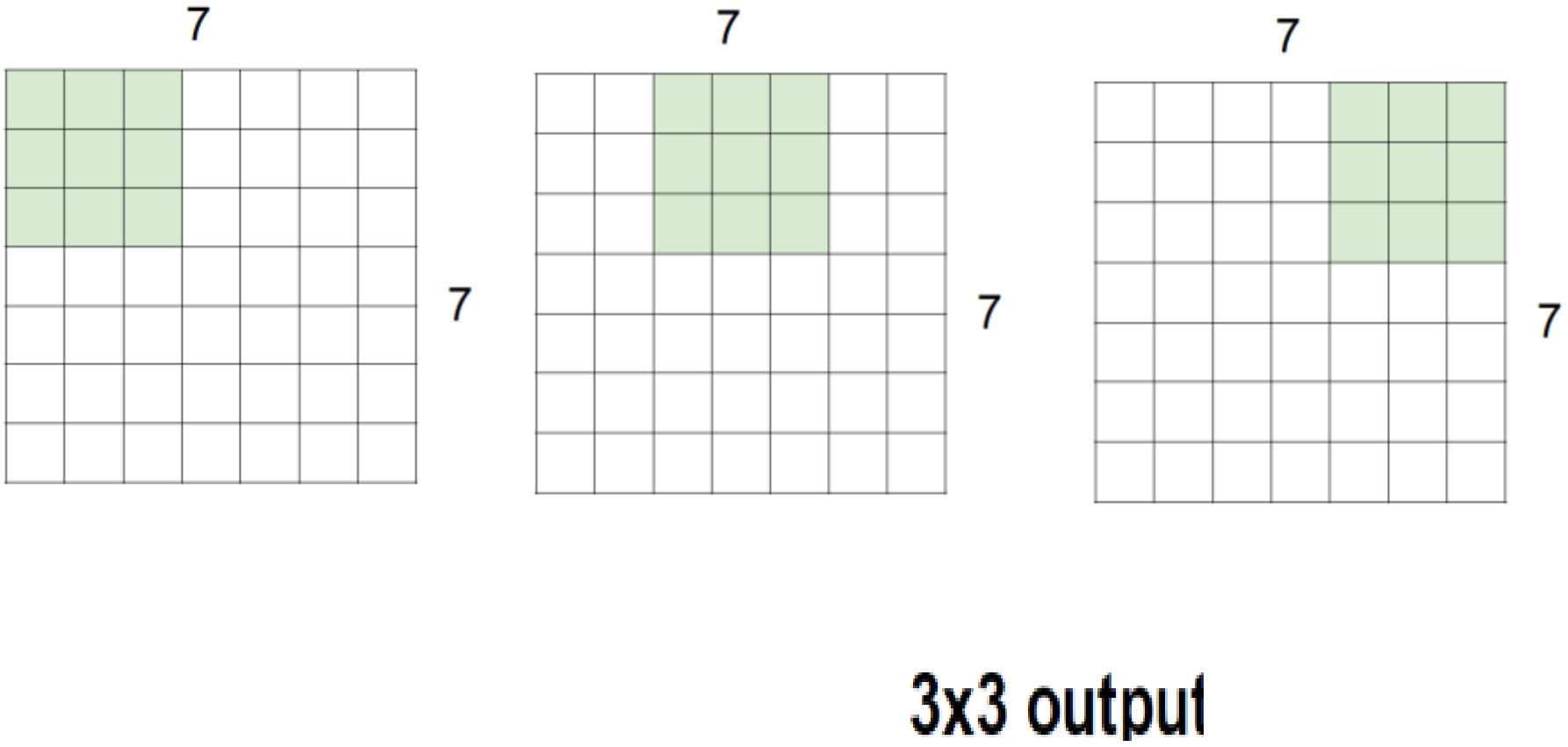



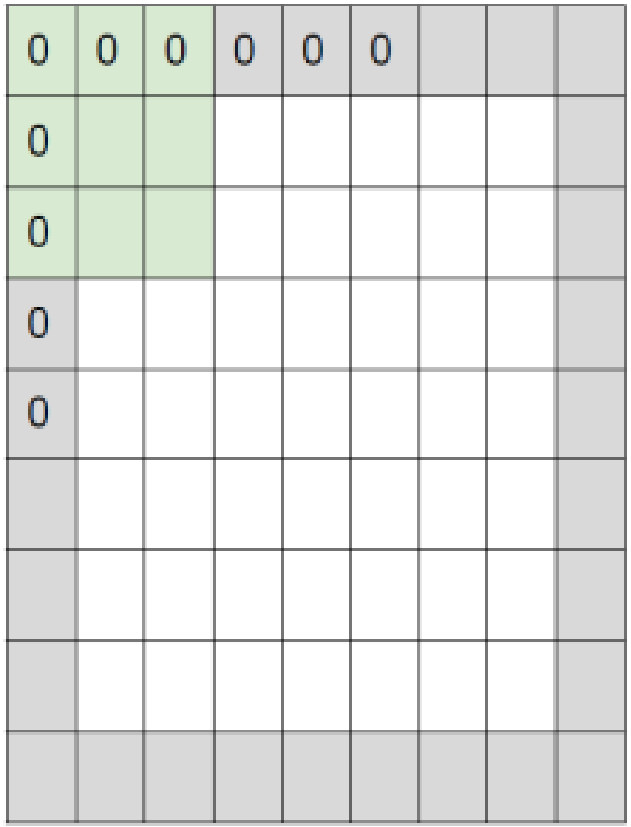




**5x5 output**

# 7 by 7 input, 3 by 3 filter with stride 2



# Convolution for edge cases: zero pad the border




7 by 7 input, 3 by 3 filter with stride 1, pad with 1 pixel border

7 by 7 output

# Generalization: zero pad the border

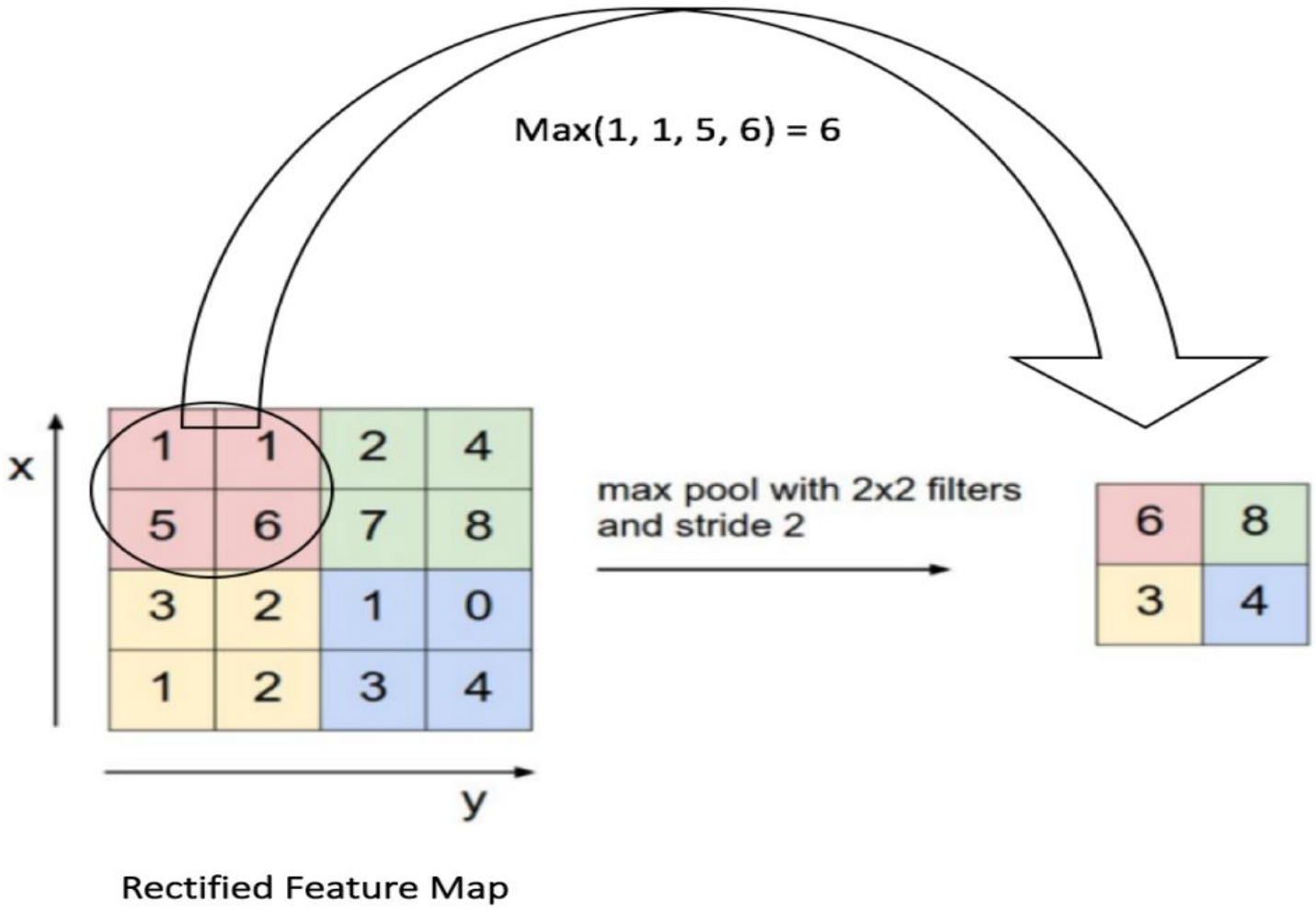
- In general, convolution layer will be having filters of size F by F with stride 1. The number of zero pad borders will be (F-1)/2 to preserve size spatially

- E.g.,

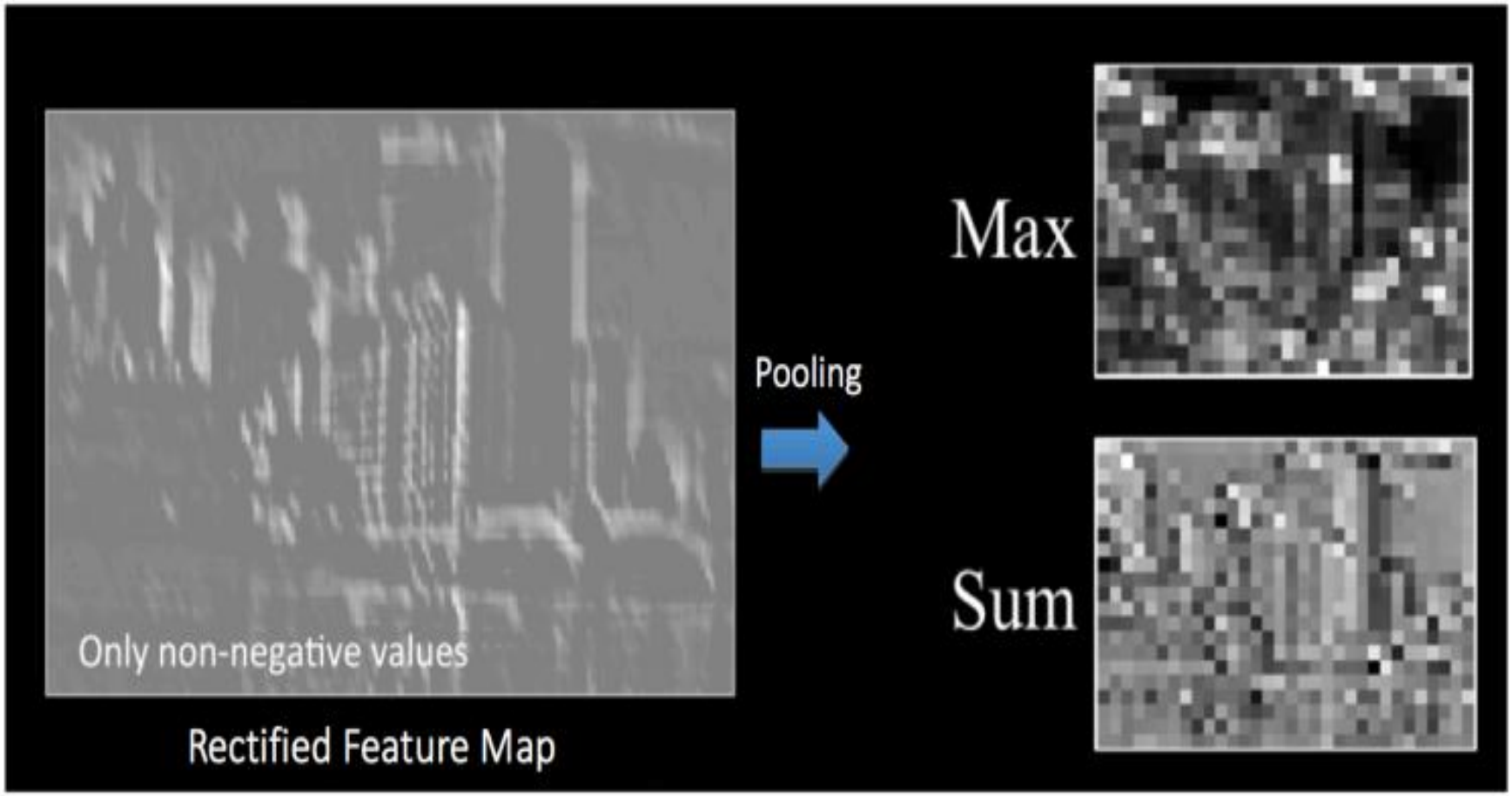
F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

# Pooling/Subsampling

# Max pooling


Max(1, 1, 5, 6) = 6
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4
Rectified Feature Map

# Max pooling


Only non-negative values
Rectified Feature Map
Pooling
Max
Sum

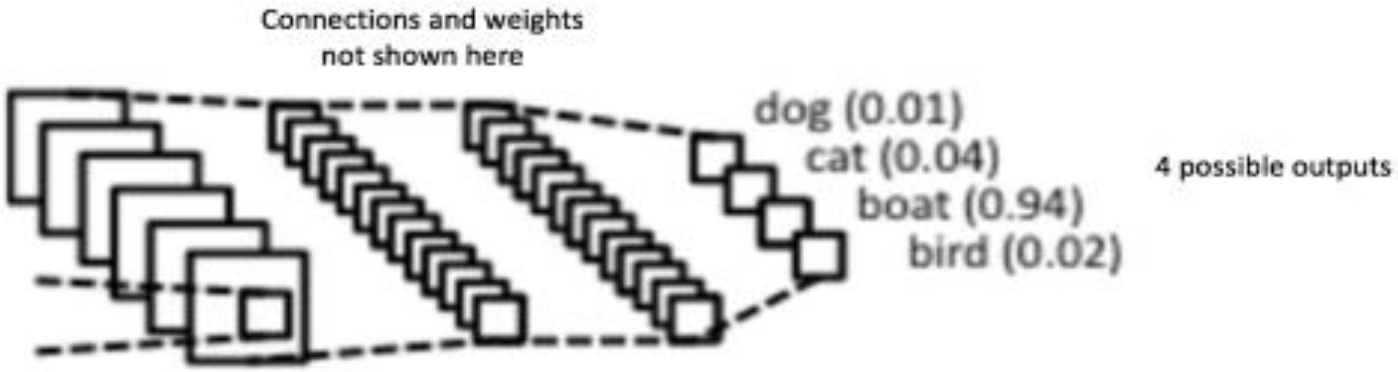
# What does pooling gives us?

- Makes the feature dimension smaller and fixed size

- Invariance to small transformations i.e., reduce the effect of noises and shift or distortion

# Fully connected layers

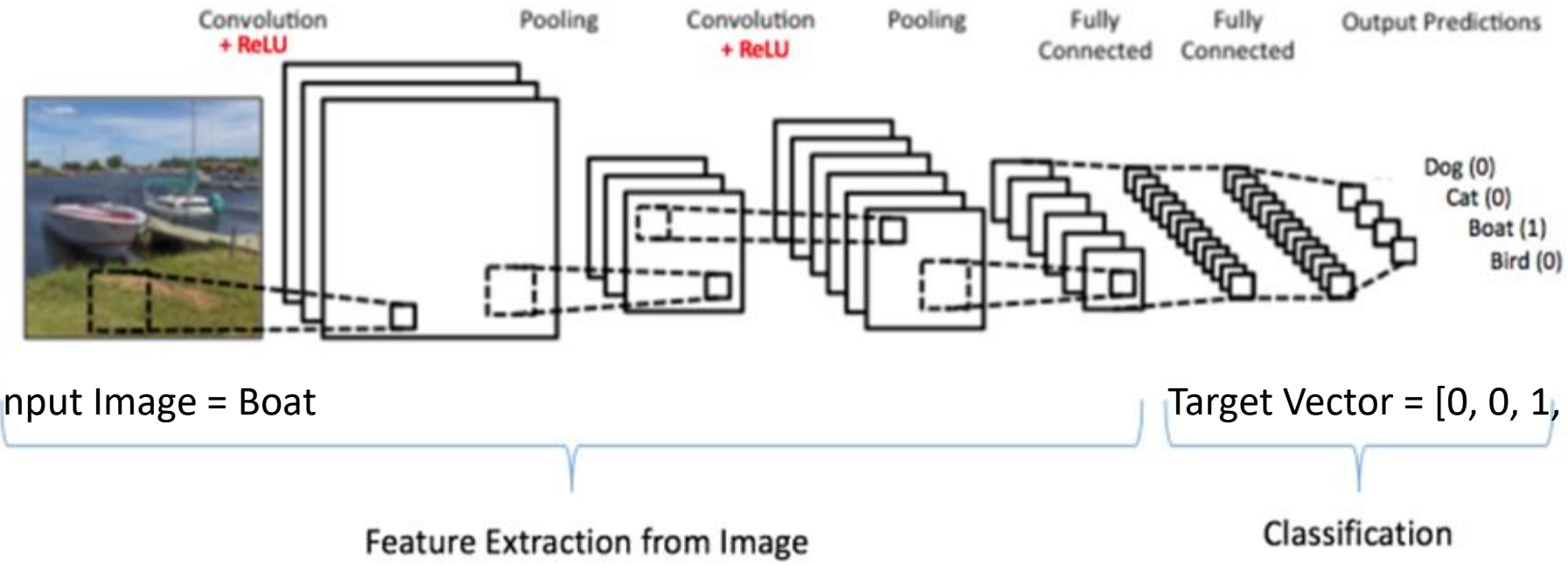
# FC layer




- The output from the convolutional and pooling layers represent high-level features of the input image

- The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset

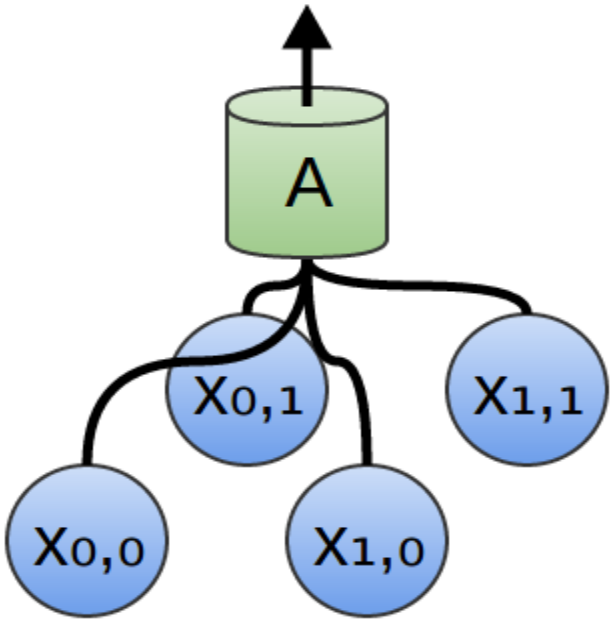
# The final convolutional neural network



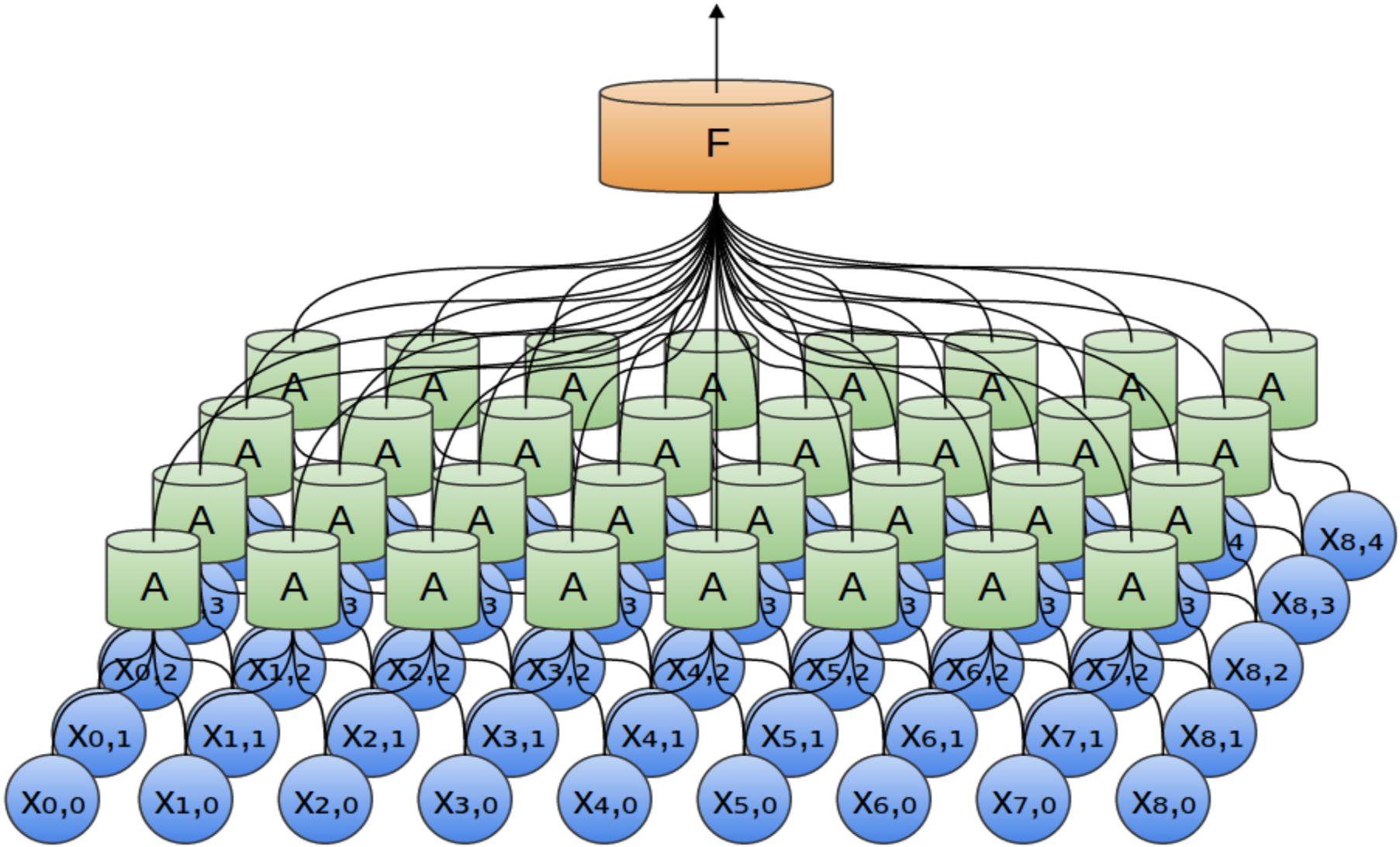

The Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier.

# Perceptron view of CNN

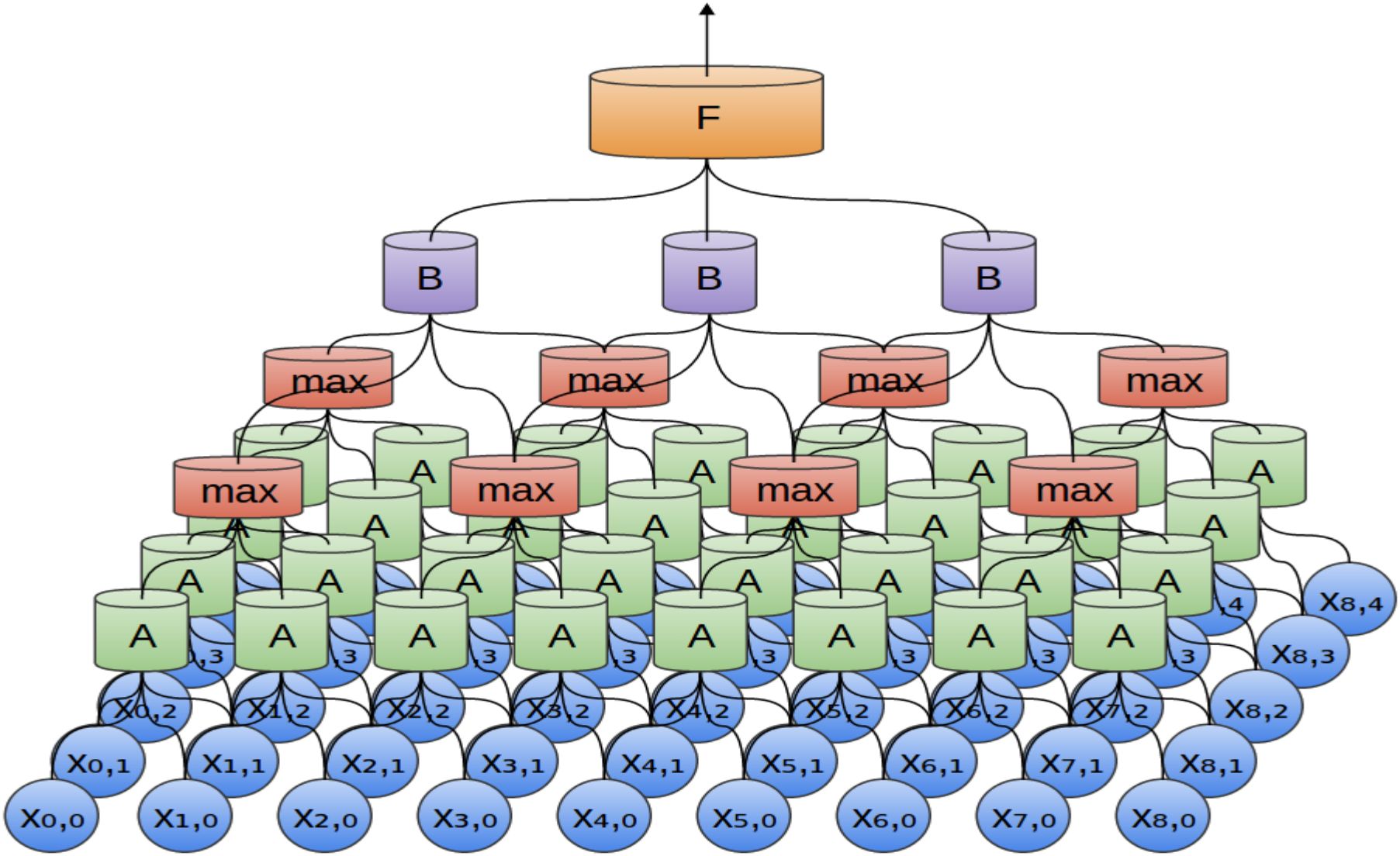
# Convolution Filter = replicated perceptrons



# Convolutional Filter = replicated perceptrons



# Conv filter + Maxpooling + Conv filter + FC

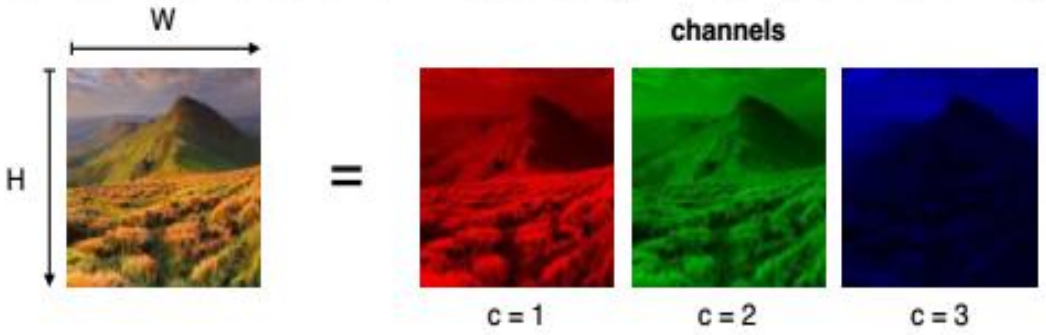


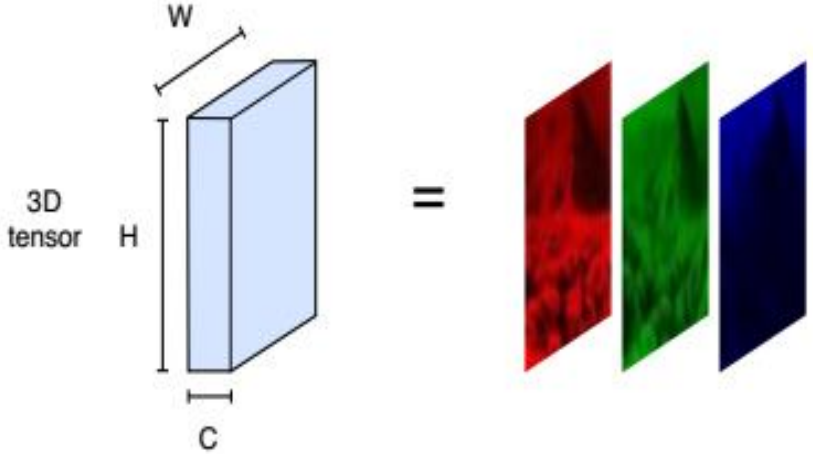
# Channels & Activation maps

# 3D Data

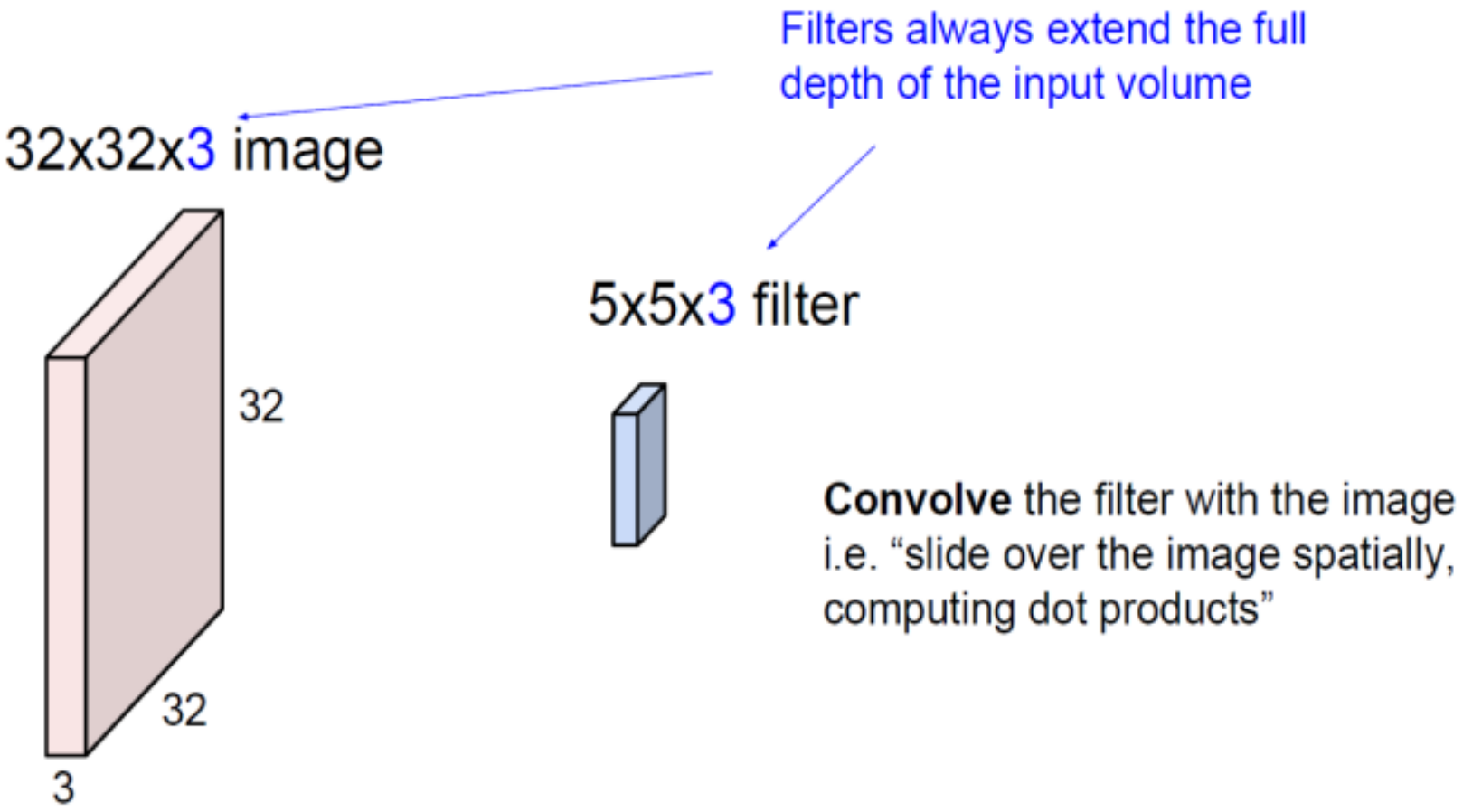
## Data = 3D Tensor

- There is a vector of feature channels (e.g. RGB) at each spatial location (pixel).

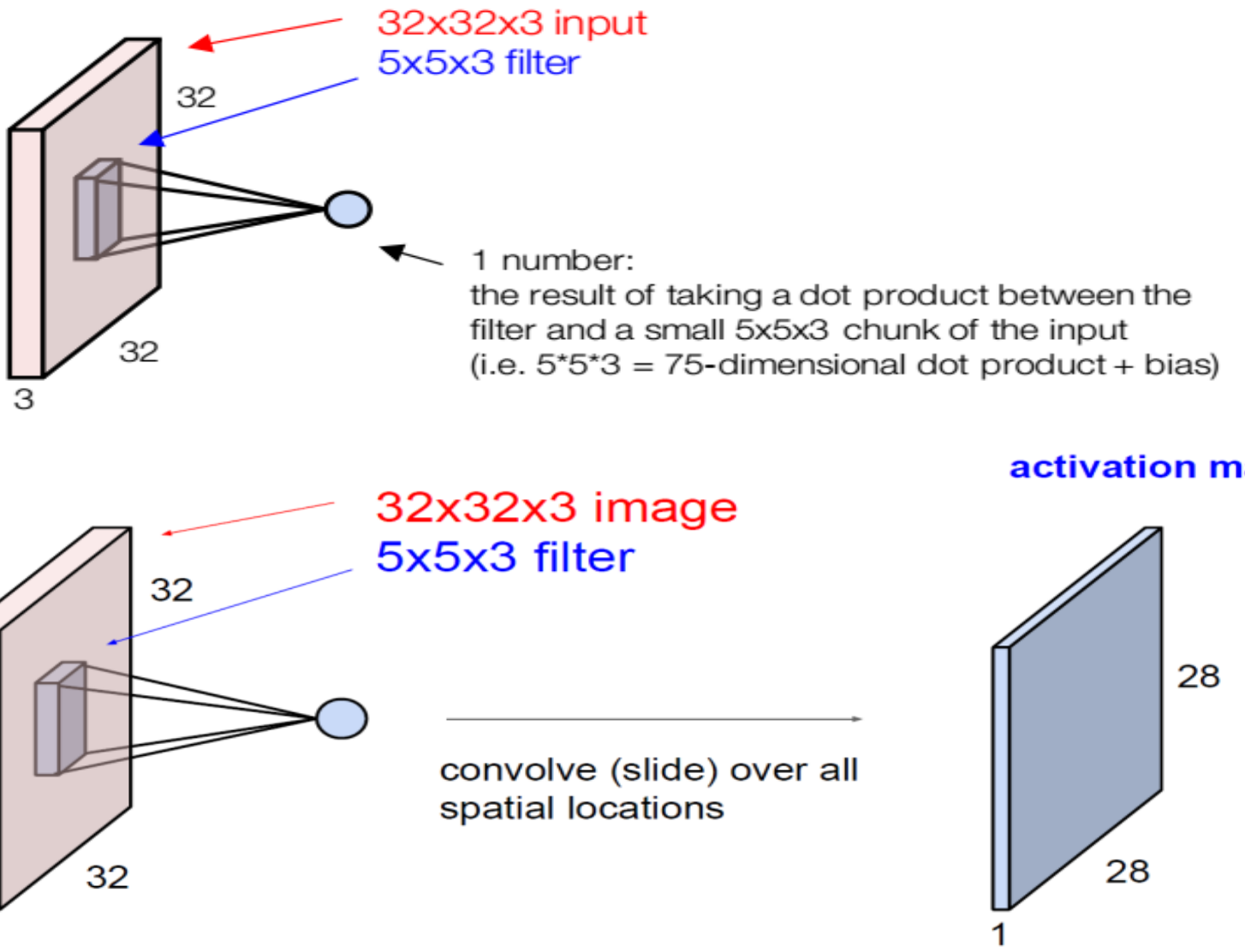


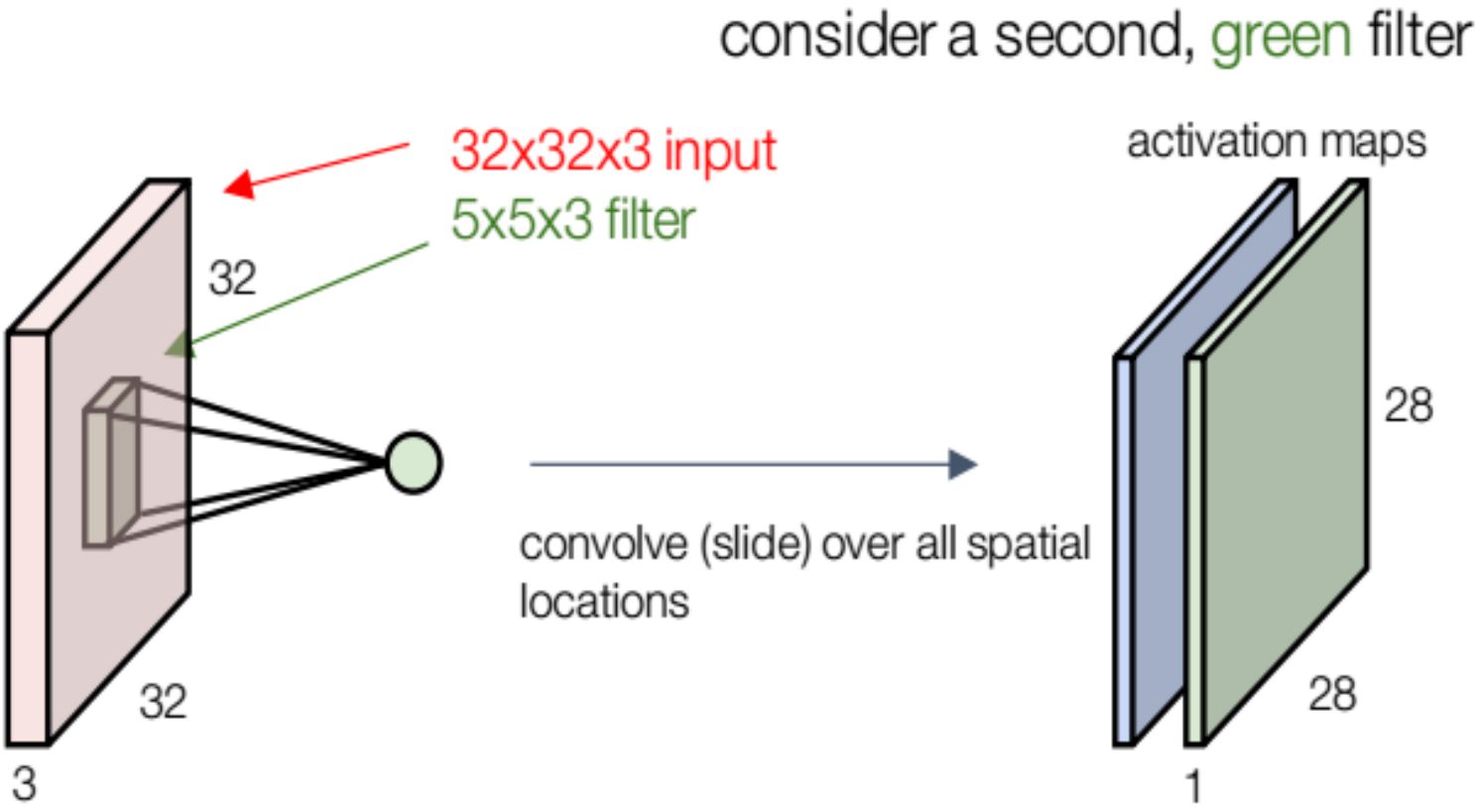
# Convolution with 3D Filters



# Convolution with 3D Filters

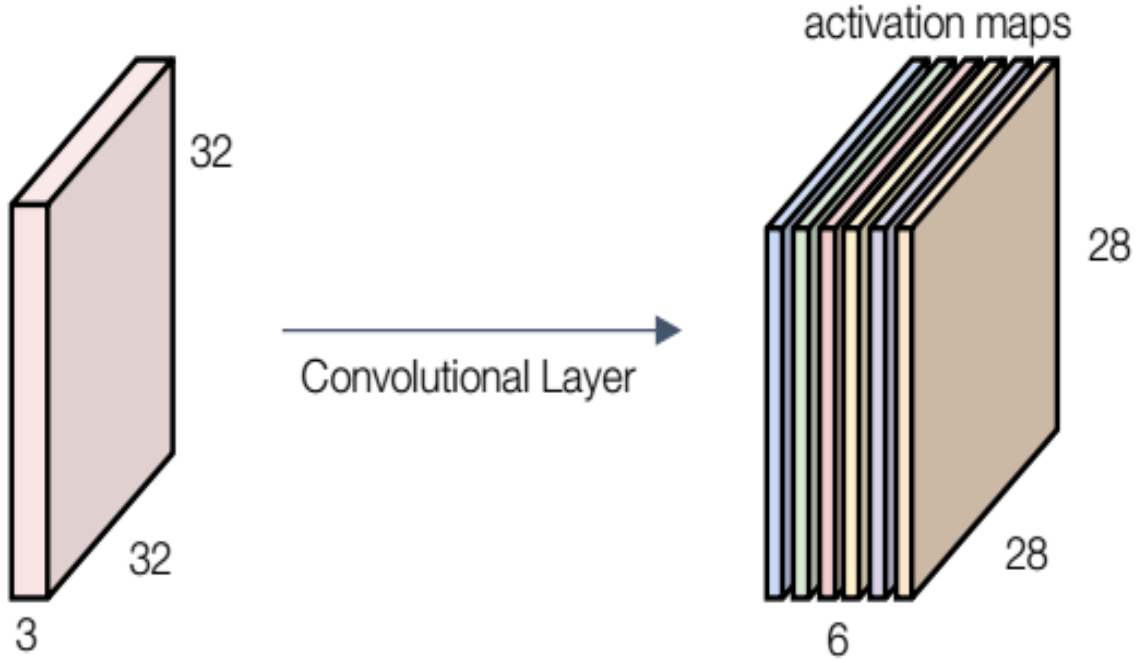


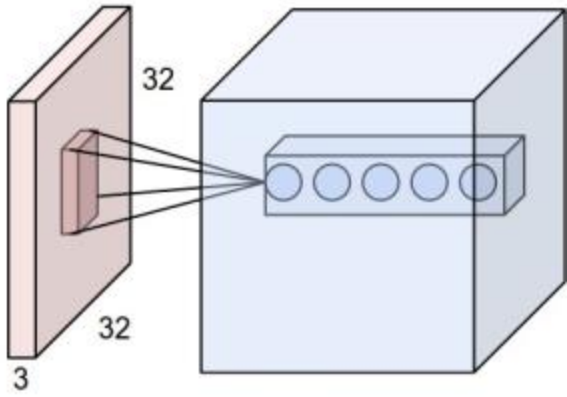
# Convolution with 3D Filters



# Convolution with 3D Filters

- Multiple filters produce multiple output channels
- For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

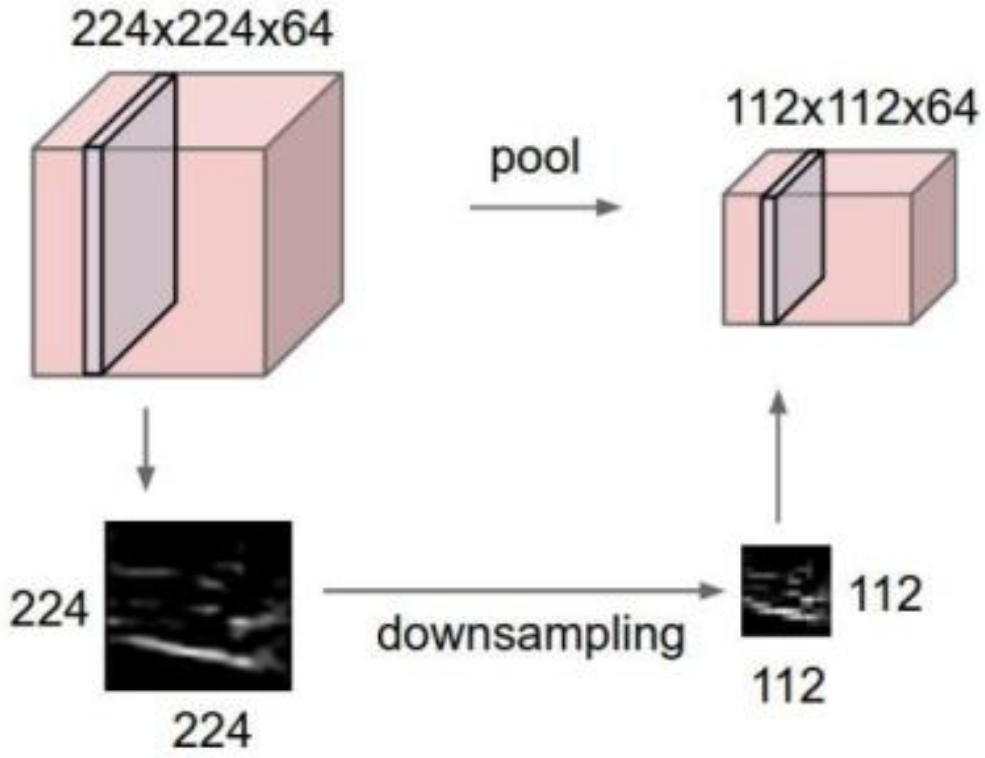


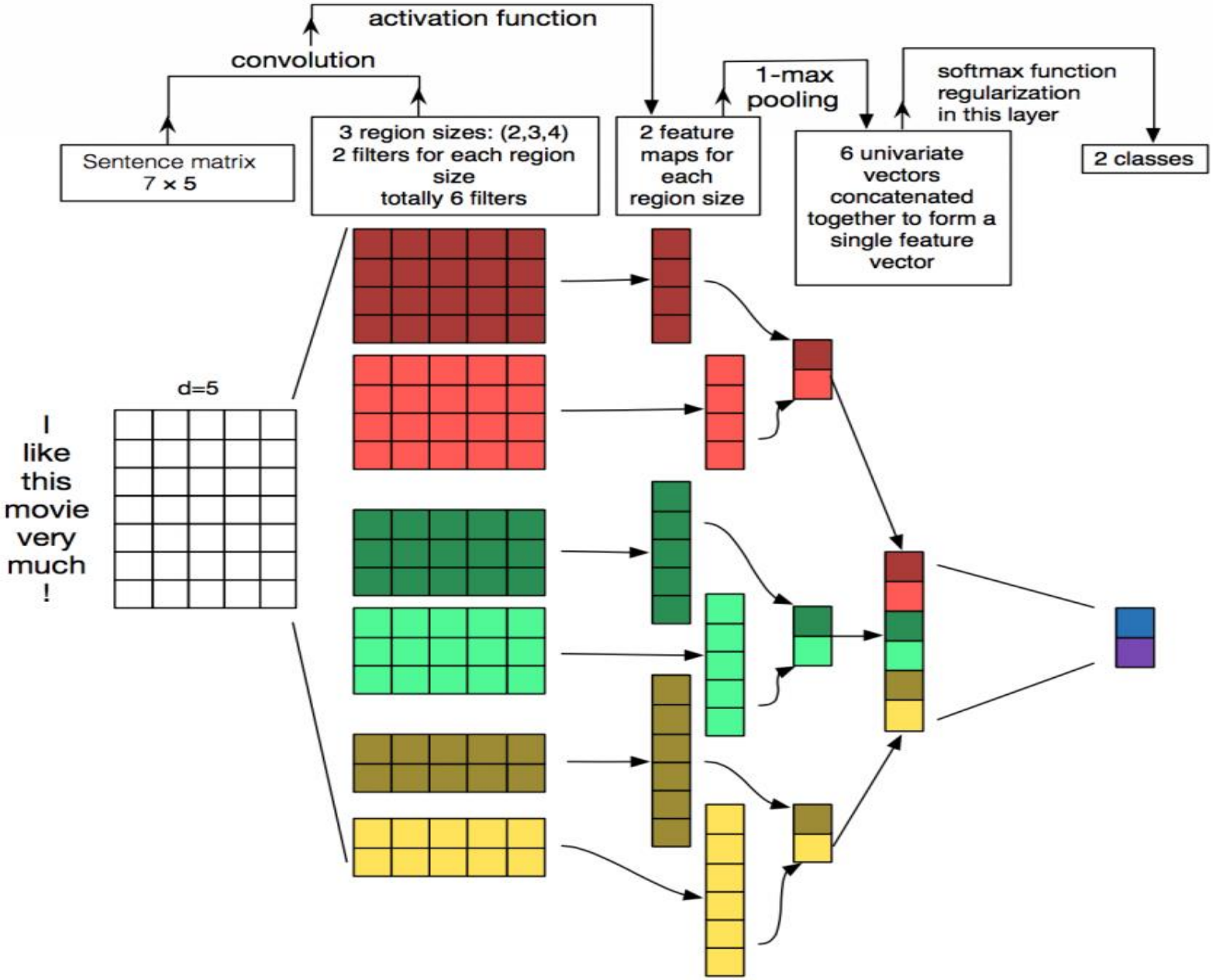

We stack these up to get an output of size 28x28x6.

# Pooling across feature maps

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



# CNN for NLP


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

# Filters + Pooling in NLP

- Each filter as detecting a specific feature, such as detecting if the sentence contains a negation like “not amazing” for example. If this phrase occurs somewhere in the sentence, the result of applying the filter to that region will yield a large value, but a small value in other regions.

- By performing the max operation you are keeping information about whether or not the feature appeared in the sentence, but you are losing information about where exactly it appeared.

# Channels in NLP

- In image recognition you typically have RGB (red, green, blue) channels. In NLP, you could have a separate channels for different word embeddings (word2vec and GloVe), or you could have a channel for the same sentence represented in different languages, or phrased in different ways.

# CNN learning algorithm

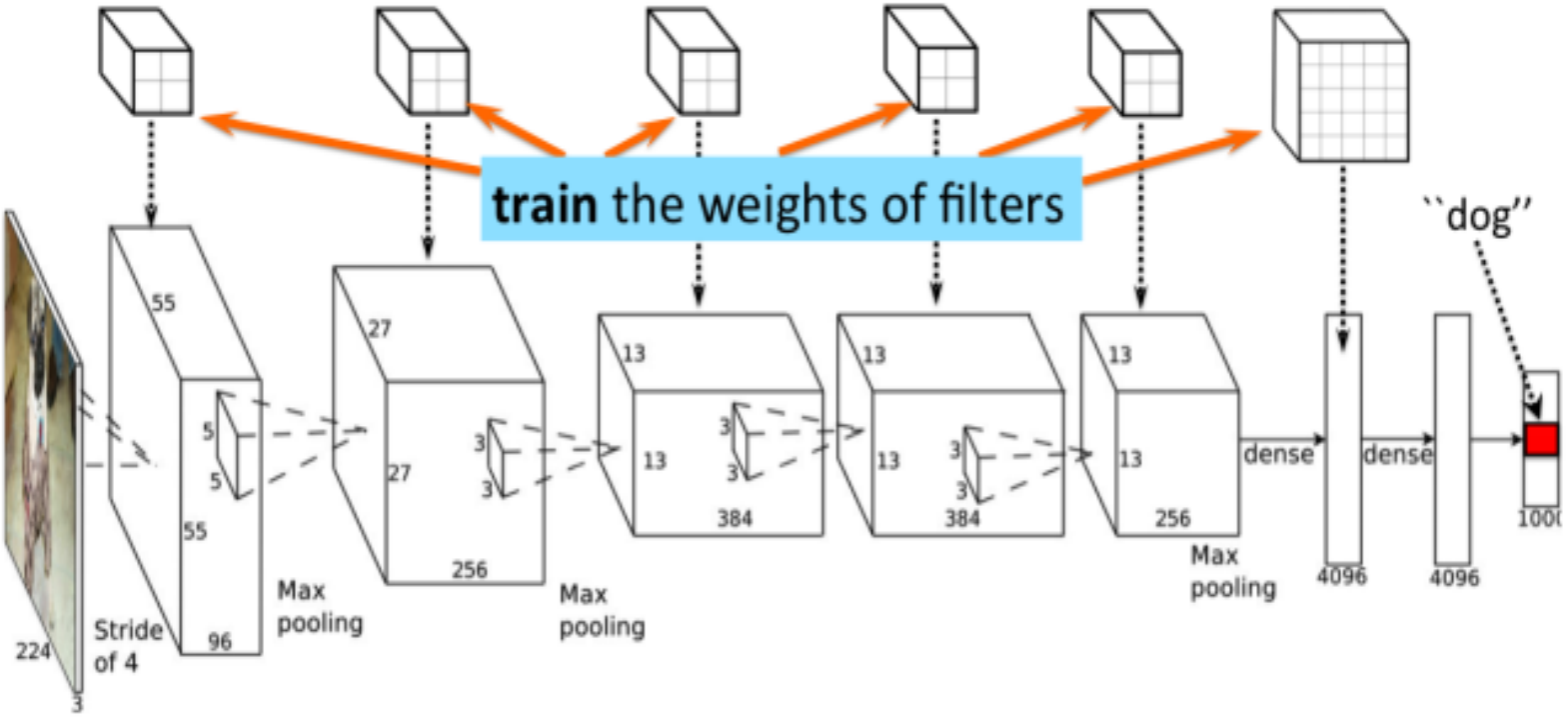
# Overview of the learning algorithm

- Initialize the CNN with random weights for filters weights and FC neuron weights

- Apply back propagation algorithm to adjust the weights based on supervised data

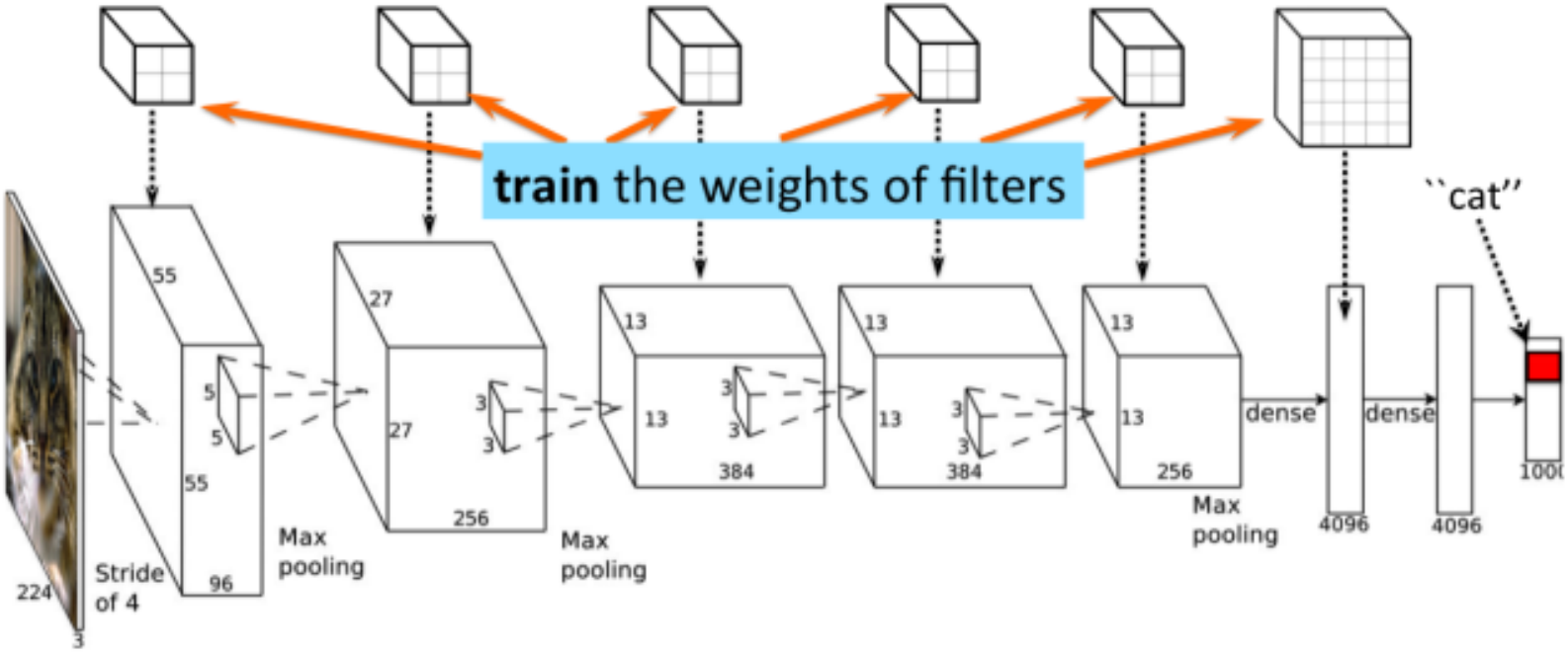
# CNN learning

- The trick is to not hand-fix the weights, but to train them. Train them such that when the network sees a picture of a dog, the last layer will say "dog".



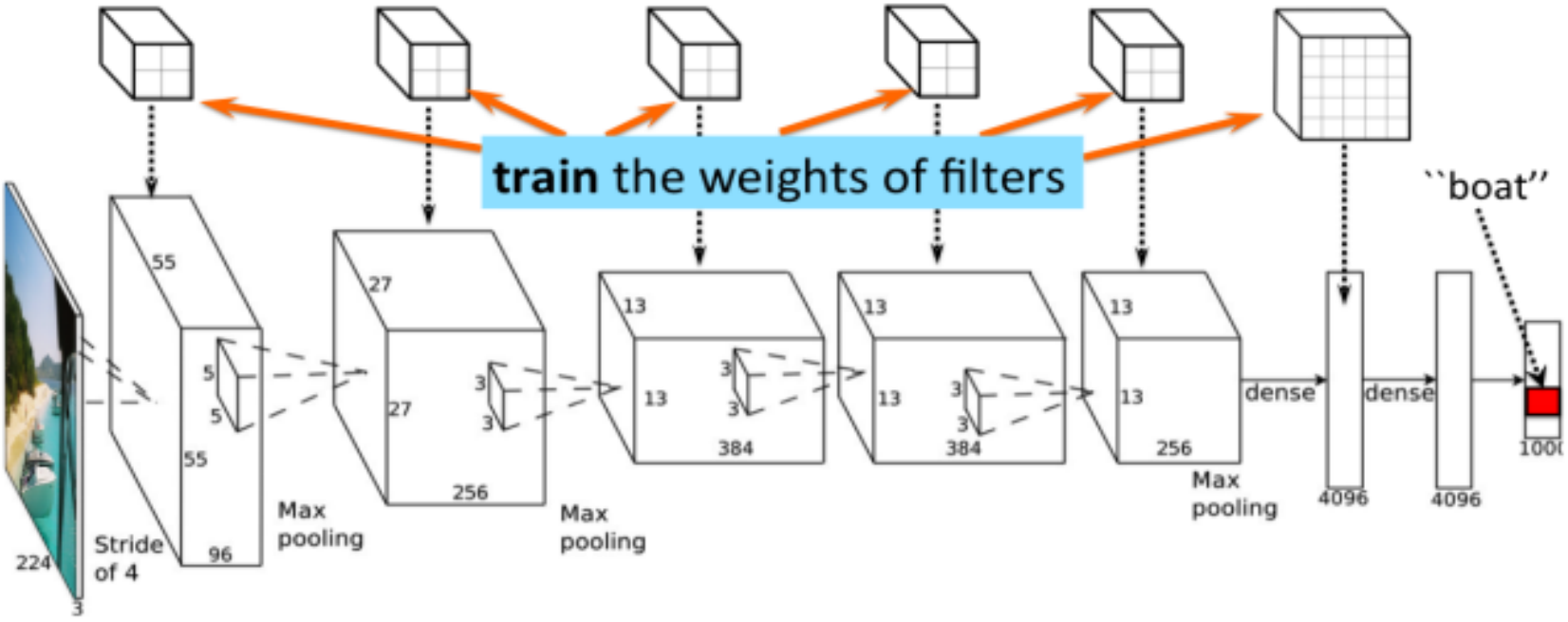
# CNN learning

- Or when the network sees a picture of a cat, the last layer will say "cat".



# CNN learning

- Or when the network sees a picture of a boat, the last layer will say "boat"... The more pictures the network sees, the better.




Train on **lots** of examples. Millions. Tens of millions. Wait a week for training to finish.

Share your network (the weights) with others who are not fortunate enough with GPU power.

# CNN for classification

- Once trained we feed in an image or a crop, run through the network, and read out the class with the highest probability in the last (classif) layer.

What's the class of this object?