Deep Learning for Industries

Demystifying Deep Learning

Hands on word2vec

Rahul Kumar
Chief AI Scientist
@BotSupply.ai / Jatana.ai

About

@hellorahulk

https://github.com/goodrahstar/

https://medium.com/@hellorahulk

www.hellorahulk.com

Deep Learning for Industries

DE3p Larenn1g mhica3ns wrok smliair to hOw biarns wrok.

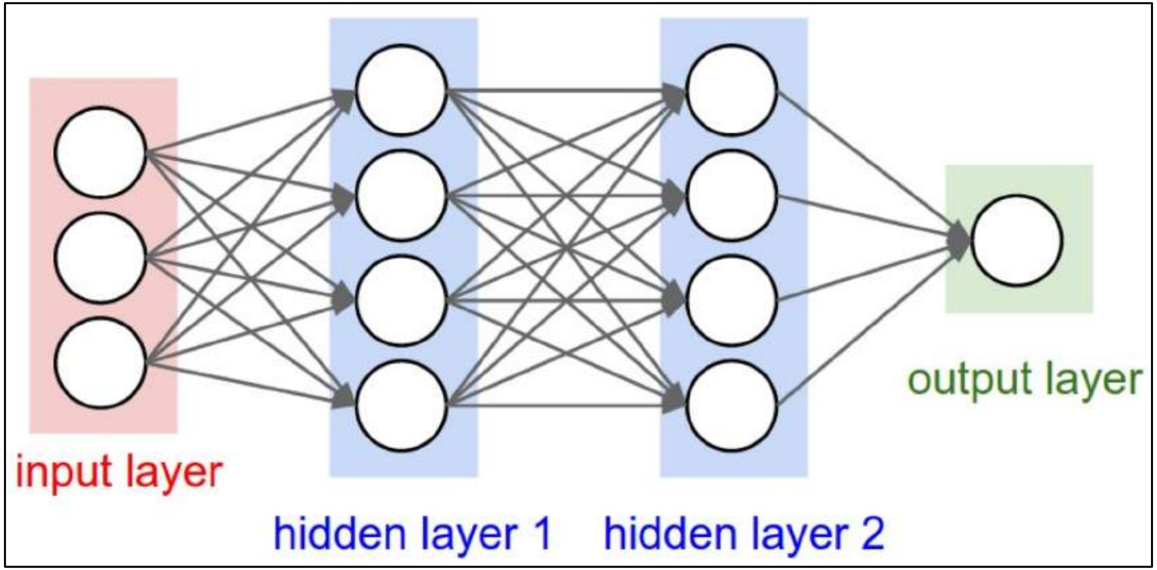
Tehse mahcnies wrok by s33nig f22Uy pa773rns and cnonc3t1ng t3Hm t0 fU22y cnoc3tps. T3hy wRok l4y3r by ly43r, j5ut lK1e a f1L37r, t4k1NG cmopl3x scn33s aNd br3k41ng tH3m dwon itno s1pmLe iD34s.

Deep Learning for Industries

Deep Learning mechanism work similar to how brain work.

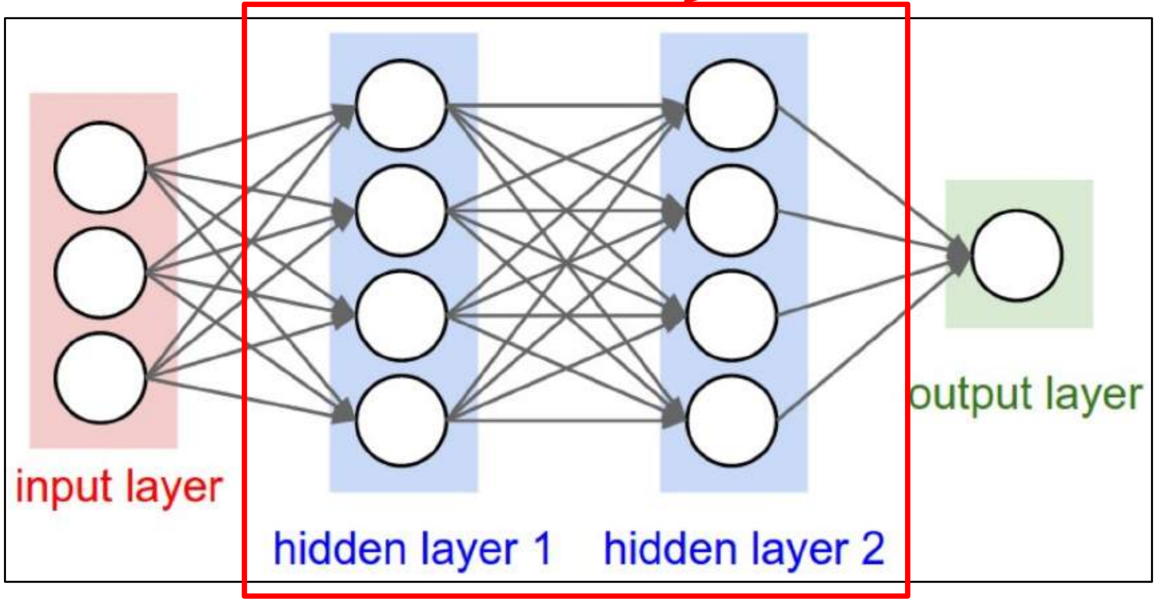
These machines works by seeing funny patterns and connecting them to funny concepts. They work layer by layer, just like a filter, taking complex scenes and breaking them down into simple ideas.

So far...



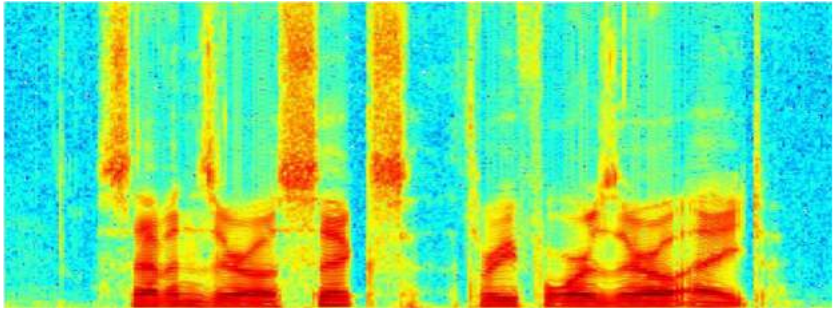
So far...

Will discuss in detail



Some input vector (very few assumptions made).

In many real-world applications input vectors have **structure**.



Spectrograms

"Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

Text

Images

Neural Networks:
A pinch of history

Hubel & Wiesel,

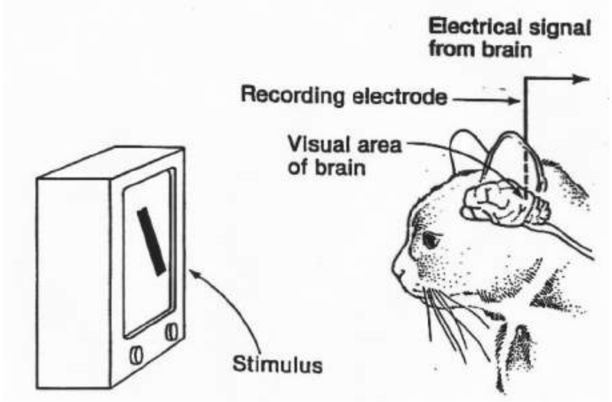
1959

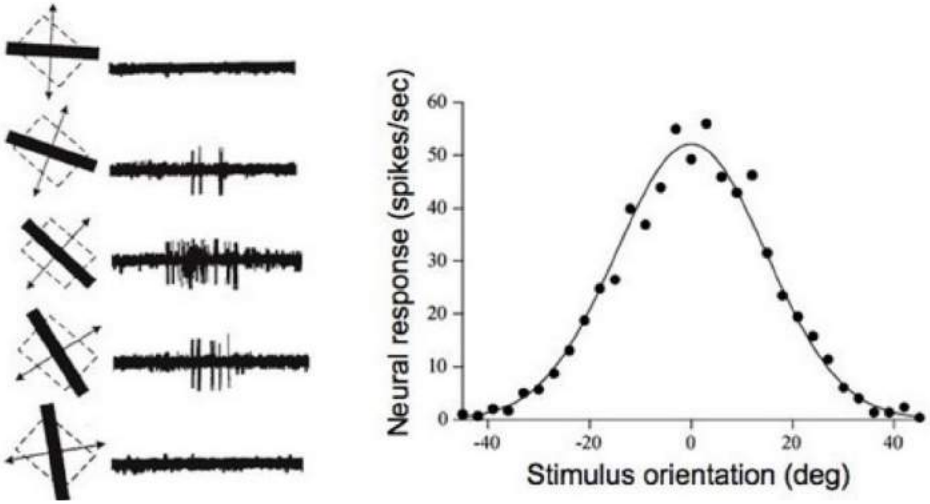
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...





Computer Vision **2011**

Computer
Vision
2011

4.1. Image Features and Kernels

We selected or designed several state-of-art features that are potentially useful for scene classification. GIST features [21] are proposed specifically for scene recognition tasks. Dense SIFT features are also found to perform very well at the 15-category dataset [17]. We also evaluate sparse SIFTs as used in "Video Google" [27]. HOG features provide excellent performance for object and human recognition tasks [4, 9], so it is interesting to examine their utility for scene recognition. While SIFT is known to be very good at finding repeated image content, the self-similarity descriptor (SSIM) [26] relates images using their internal layout of local self-similarities. Unlike GIST, SIFT, and HOG, which are all local gradient-based approaches, SSIM may provide a distinct, complementary measure of scene layout that is somewhat appearance invariant. As a baseline, we also include Tiny Images [28], color histograms and straight line histograms. To make our color and texton histograms more invariant to scene layout, we also build histograms for specific geometric classes as determined by [13]. The geometric classification of a scene is then itself used as a feature, hopefully being invariant to appearance but responsive to layout.

GIST: The GIST descriptor [21] computes the output energy of a bank of 24 filters. The filters are Gabor-like filters tuned to 8 orientations at 4 different scales. The square output of each filter is then averaged on a 4×4 grid.

HOG2x2: First, histogram of oriented edges (HOG) descriptors [4] are densely extracted on a regular grid at steps of 8 pixels. HOG features are computed using the code available online provided by [9], which gives a 31-dimension descriptor for each node of the grid. Then, 2×2 neighboring HOG descriptors are stacked together to form a descriptor with 124 dimensions. The stacked descriptors spatially overlap. This 2×2 neighbor stacking is important because the higher feature dimensionality provides more descriptive power. The descriptors are quantized into 300 visual words by k-means. With this visual word representation, three-level spatial histograms are computed on grids of 1×1, 2×2 and 4×4. Histogram intersection[17] is used to define the similarity of two histograms at the same pyramid level for two images. The kernel matrices at the three levels are normalized by their respective means, and linearly combined together using equal weights.

Dense SIFT: As with HOG2x2, SIFT descriptors are densely extracted [17] using a flat rather than Gaussian window at two scales (4 and 8 pixel radii) on a regular grid at steps of 5 pixels. The three descriptors are stacked together for each HSV color channels, and quantized into 300 visual words by k-means, and spatial pyramid histograms are used as kernels[17].

Computer Vision **2011**

LBP: Local Binary Patterns (LBP) [20] is a powerful texture feature based on occurrence histogram of local binary patterns. We can regard the scene recognition as a texture classification problem of 2D images, and therefore apply this model to our problem. We also try the rotation invariant extension version [2] of LBP to examine whether rotation invariance is suitable for scene recognition.

Sparse SIFT histograms: As in "Video Google" [27], we build SIFT features at Hessian-affine and MSER [19] interest points. We cluster each set of SIFTs, independently, into dictionaries of 1,000 visual words using k-means. An image is represented by two histograms counting the number of sparse SIFTs that fall into each bin. An image is represented by two 1,000 dimension histograms where each SIFT is soft-assigned, as in [22], to its nearest cluster centers. Kernels are computed with χ^2 distance.

SSIM: Self-similarity descriptors [26] are computed on a regular grid at steps of five pixels. Each descriptor is obtained by computing the correlation map of a patch of 5×5 in a window with radius equal to 40 pixels, then quantizing it in 3 radial bins and 10 angular bins, obtaining 30 dimensional descriptor vectors. The descriptors are then quantized into 300 visual words by k-means and we use χ^2 distance on spatial histograms for the kernels.

Tiny Images: The most trivial way to match scenes is to compare them directly in color image space. Reducing the image dimensions drastically makes this approach more computationally feasible and less sensitive to exact alignment. This method of image matching has been examined thoroughly by Torralba et al.[28] for the purpose of object recognition and scene classification.

Line Features: We detect straight lines from Canny edges using the method described in Video Compass [15]. For each image we build two histograms based on the statistics of detected lines– one with bins corresponding to line angles and one with bins corresponding to line lengths. We use an RBF kernel to compare these unnormalized histograms. This feature was used in [11].

Texton Histograms: We build a 512 entry universal texton dictionary [18] by clustering responses to a bank of filters with 8 orientations, 2 scales, and 2 elongations. For each image we then build a 512-dimensional histogram by assigning each pixel's set of filter responses to the nearest texton dictionary entry. We compute kernels from normalized χ^2 distances.

Color Histograms: We build joint histograms of color in CIE L*a*b* color space for each image. Our histograms have 4, 14, and 14 bins in L, a, and b respectively for a total of 784 dimensions. We compute distances between these histograms using χ^2 distance on the normalized histograms.

Geometric Probability Map: We compute the geometric class probabilities for image regions using the method of Hoiem et al. [13]. We use only the ground, vertical, porous, and sky classes because they are more reliably classified. We reduce the probability maps for each class to 8×8 and use an RBF kernel. This feature was used in [11].

Computer Vision **2011**

Geometry Specific Histograms: Inspired by "Illumination Context" [16], we build color and texton histograms for each geometric class (ground, vertical, porous, and sky). Specifically, for each color and texture sample, we weight its contribution to each histogram by the probability that it belongs to that geometric class. These eight histograms are compared with χ^2 distance after normalization.

\+ code complexity :(

vector describing
various image statistics

[224x224x3]

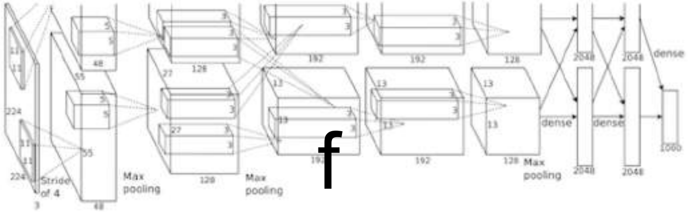
Feature
Extraction

f

1000 numbers,
indicating class scores

training

[224x224x3]



f

1000 numbers,
indicating class scores

training

4.1. Image Features and Kernels

We selected or designed several state-of-art features that are potentially useful for scene classification. GIST features [21] are proposed specifically for scene recognition tasks. Dense SIFT features are also found to perform very well at the 15-category dataset [17]. We also evaluate sparse SIFTs as used in "Video Google" [27]. HOG features provide excellent performance for object and human recognition tasks [4, 9], so it is interesting to examine their utility for scene recognition. While SIFT is known to be very good at finding repeated image content, the self-similarity descriptor (SSIM) [26] relates images using their internal layout of local self-similarities. Unlike GIST, SIFT, and HOG, which are all local gradient-based approaches, SSIM may provide a distinct, complementary measure of scene layout that is somewhat appearance invariant. As a baseline, we also include Tiny Images [28], color histograms and straight line histograms. To make our color and texton histograms more invariant to scene layout, we also build histograms for specific geometric classes as determined by [13]. The geometric classification of a scene is then itself used as a feature, hopefully being invariant to appearance but responsive to layout.

GIST: The GIST descriptor [21] computes the output energy of a bank of 24 filters. The filters are Gabor-like filters tuned to 8 orientations at 4 different scales. The square output of each filter is then averaged on a 4×4 grid.

HOG2x2: First, histogram of oriented edges (HOG) descriptors [4] are densely extracted on a regular grid at steps of 8 pixels. HOG features are computed using the code available online provided by [9], which gives a 31-dimension descriptor for each node of the grid. Then, 2×2 neighboring HOG descriptors are stacked together to form a descriptor with 124 dimensions. The stacked descriptors spatially overlap. This 2×2 neighbor stacking is important because the higher feature dimensionality provides more descriptive power. The descriptors are quantized into 300 visual words by k-means. With this visual word representation, three-level spatial histograms are computed on grids of 1×1, 2×2 and 4×4. Histogram intersection[17] is used to define the similarity of two histograms at the same pyramid level for two images. The kernel matrices at the three levels are normalized by their respective means, and linearly combined together using equal weights.

Dense SIFT: As with HOG2x2, SIFT descriptors are densely extracted [17] using a flat rather than Gaussian window at two scales (4 and 8 pixel radii) on a regular grid at steps of 5 pixels. The three descriptors are stacked together for each HSV color channels, and quantized into 300 visual words by k-means, and spatial pyramid histograms are used as kernels[17].

LBP: Local Binary Patterns (LBP) [20] is a powerful texture feature based on occurrence histogram of local binary patterns. We can regard the scene recognition as a texture classification problem of 2D images, and therefore apply this model to our problem. We also try the rotation invariant extension version [2] of LBP to examine whether rotation invariance is suitable for scene recognition.

Sparse SIFT histograms: As in "Video Google" [27], we build SIFT features at Hessian-affine and MSER [19] interest points. We cluster each set of SIFTs, independently, into dictionaries of 1,000 visual words using k-means. An image is represented by two histograms counting the number of sparse SIFTs that fall into each bin. An image is represented by two 1,000 dimension histograms where each SIFT is soft-assigned, as in [22], to its nearest cluster centers. Kernels are computed with χ^2 distance.

SSIM: Self-similarity descriptors [26] are computed on a regular grid at steps of five pixels. Each descriptor is obtained by computing the correlation map of a patch of 5×5 in a window with radius equal to 40 pixels, then quantizing it in 3 radial bins and 10 angular bins, obtaining 30 dimensional descriptor vectors. The descriptors are then quantized into 300 visual words by k-means and we use χ^2 distance on spatial histograms for the kernels.

Tiny Images: The most trivial way to match scenes is to compare them directly in color image space. Reducing the image dimensions drastically makes this approach more computationally feasible and less sensitive to exact alignment. This method of image matching has been examined thoroughly by Torralba et al.[28] for the purpose of object recognition and scene classification.

Line Features: We detect straight lines from Canny edges using the method described in Video Compass [15]. For each image we build two histograms based on the statistics of detected lines– one with bins corresponding to line angles and one with bins corresponding to line lengths. We use an RBF kernel to compare these unnormalized histograms. This feature was used in [11].

Texton Histograms: We build a 512 entry universal texton dictionary [18] by clustering responses to a bank of filters with 8 orientations, 2 scales, and 2 elongations. For each image we then build a 512-dimensional histogram by assigning each pixel's set of filter responses to the nearest texton dictionary entry. We compute kernels from normalized χ^2 distances.

Color Histograms: We build joint histograms of color in CIE L*a*b* color space for each image. Our histograms have 4, 14, and 14 bins in L, a, and b respectively for a total of 784 dimensions. We compute distances between these histograms using χ^2 distance on the normalized histograms.

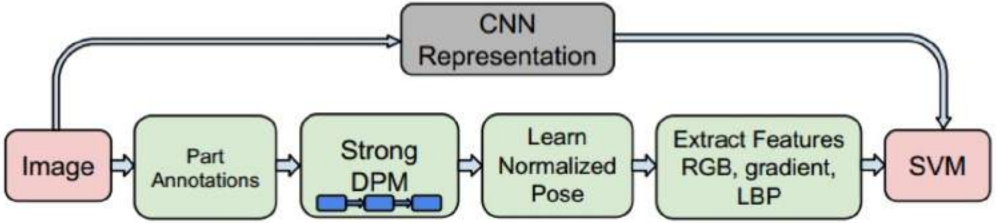
Geometric Probability Map: We compute the geometric class probabilities for image regions using the method of Hoiem et al. [13]. We use only the ground, vertical, porous, and sky classes because they are more reliably classified. We reduce the probability maps for each class to 8×8 and use an RBF kernel. This feature was used in [11].

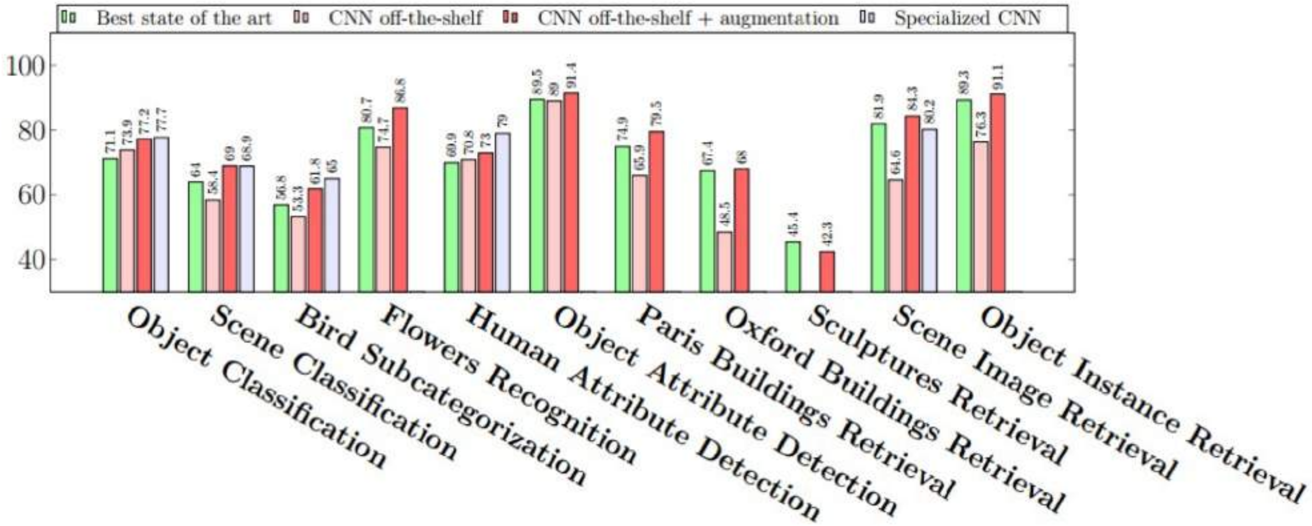
Geometry Specific Histograms: Inspired by "Illumination Context" [16], we build color and texton histograms for each geometric class (ground, vertical, porous, and sky). Specifically, for each color and texture sample, we weight its contribution to each histogram by the probability that it belongs to that geometric class. These eight histograms are compared with χ^2 distance after normalization.

"Run the image through 20 layers of 3x3 convolutions and train the filters with SGD."

DNN Approach

CNN Features off-the-shelf: an Astounding Baseline for Recognition [Razavian et al, 2014]

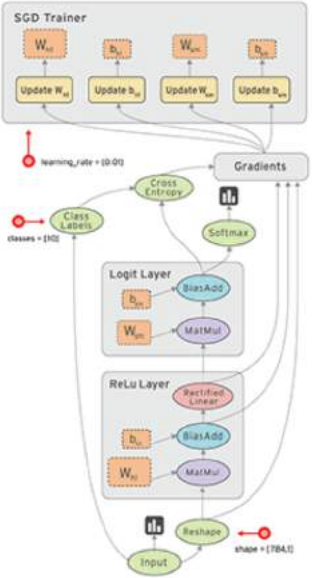




The power is easily accessible.

e.g. with TensorFlow

```
# Python 2
$ sudo pip install --upgrade tensorflow

# Python 3
$ sudo pip3 install --upgrade tensorflow
```




```python
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute
y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but TensorFlow will
# figure that out for us.)

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.
#We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))
# Learns best fit is W: [0.1], b: [0.3]
```

TensorFlow : Programming Paradigm

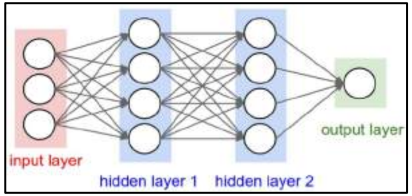
```python
# Import MINST data
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('data_dir', one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Load library and MNIST data

TensorFlow : Programming Paradigm

```python
# Import MINST data
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets(‘data_dir’, one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Design neural network architecture



TensorFlow : Programming Paradigm

```python
# Import MINST data
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('data_dir', one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Select optimization algorithm

TensorFlow : Programming Paradigm

```python
# Import MINST data
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('data_dir', one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Initialize the session and variables

TensorFlow : Programming Paradigm

```python
# Import MINST data
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets('data_dir', one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Train the model

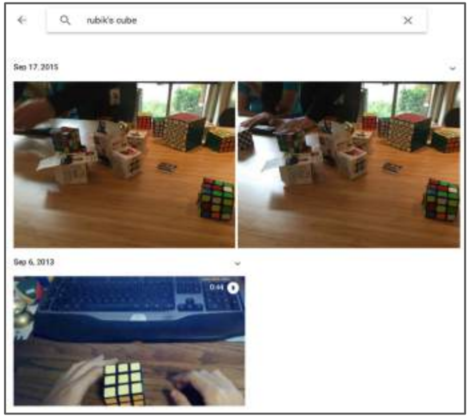
TensorFlow : Programming Paradigm

```python
# Import MINST data
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets(‘data_dir’, one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

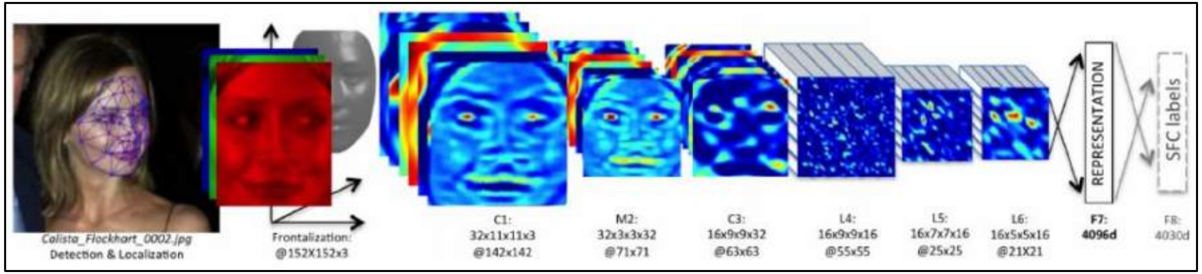
95.3%

Convolutional Neural Networks

ConvNets are everywhere…



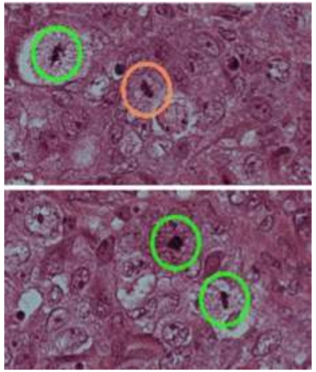
e.g. Google Photos search



Face Verification, Taigman et al. 2014 (FAIR)

[Goodfellow et al. 2014]

Self-driving cars

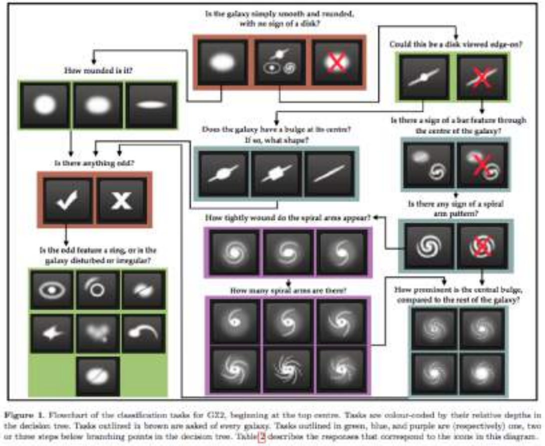


Ciresan et al. 2013

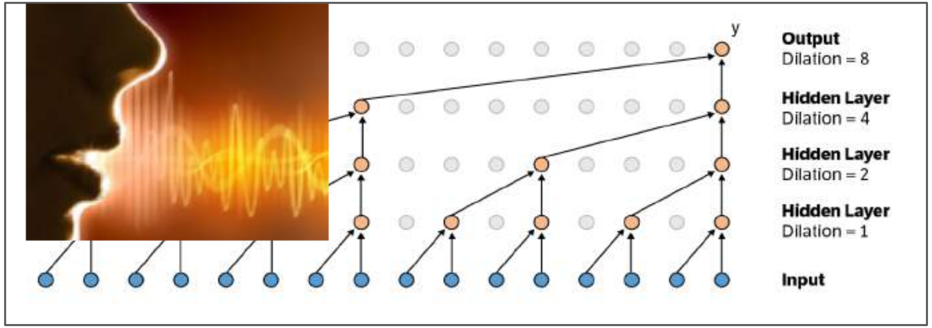
ConvNets are everywhere…

Whale recognition, Kaggle Challenge

Satellite image analysis
Mnih and Hinton, 2010



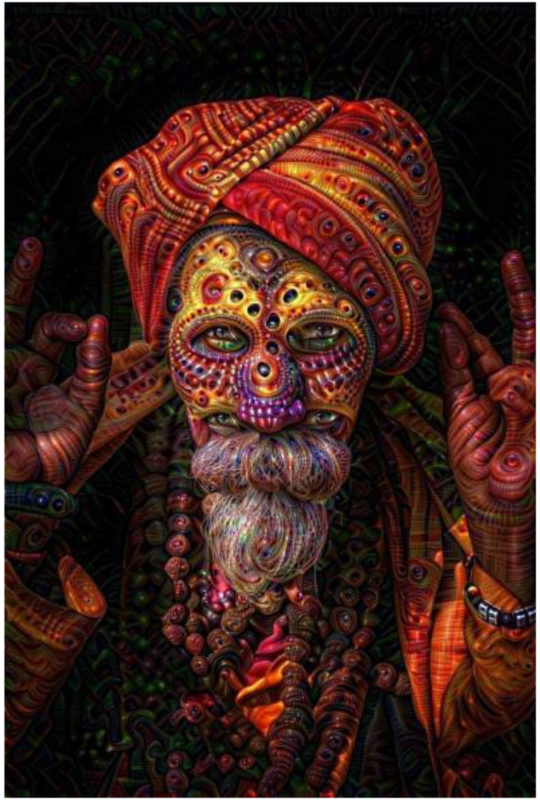
Galaxy Challenge Dielman et al. 2015

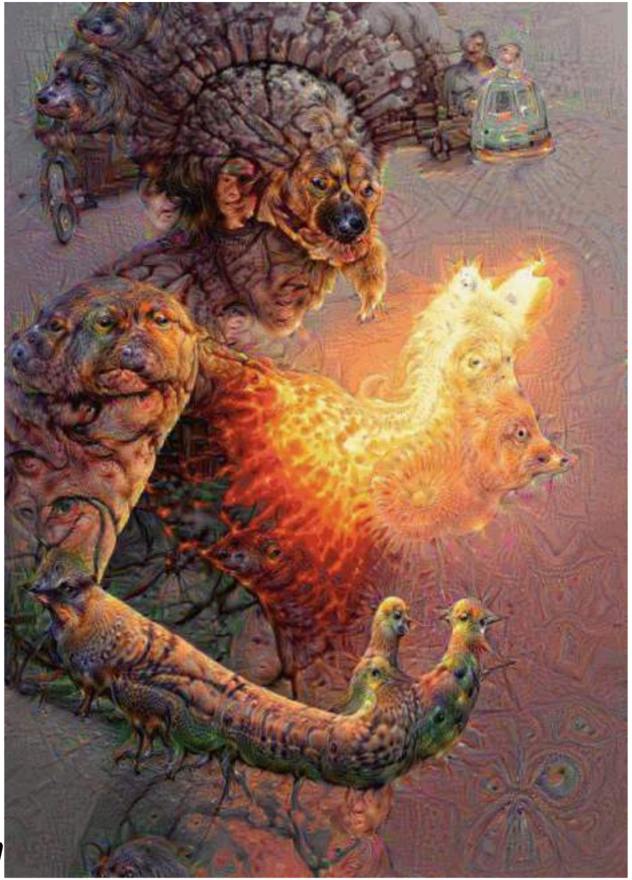


WaveNet, van den Oord et al. 2016

Image captioning, Vinyals et al. 2015

ConvNets are everywhere…

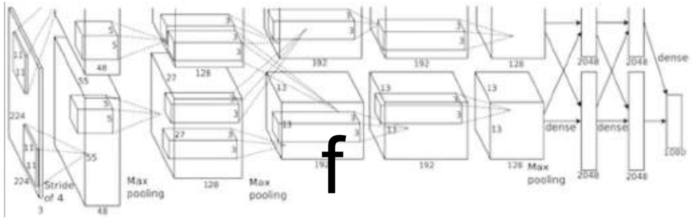




DeepDream *reddit.com/r/deepdream*

NeuralStyle, Gatys et al. 2015
deepart.io, Prisma, etc.

[224x224x3]



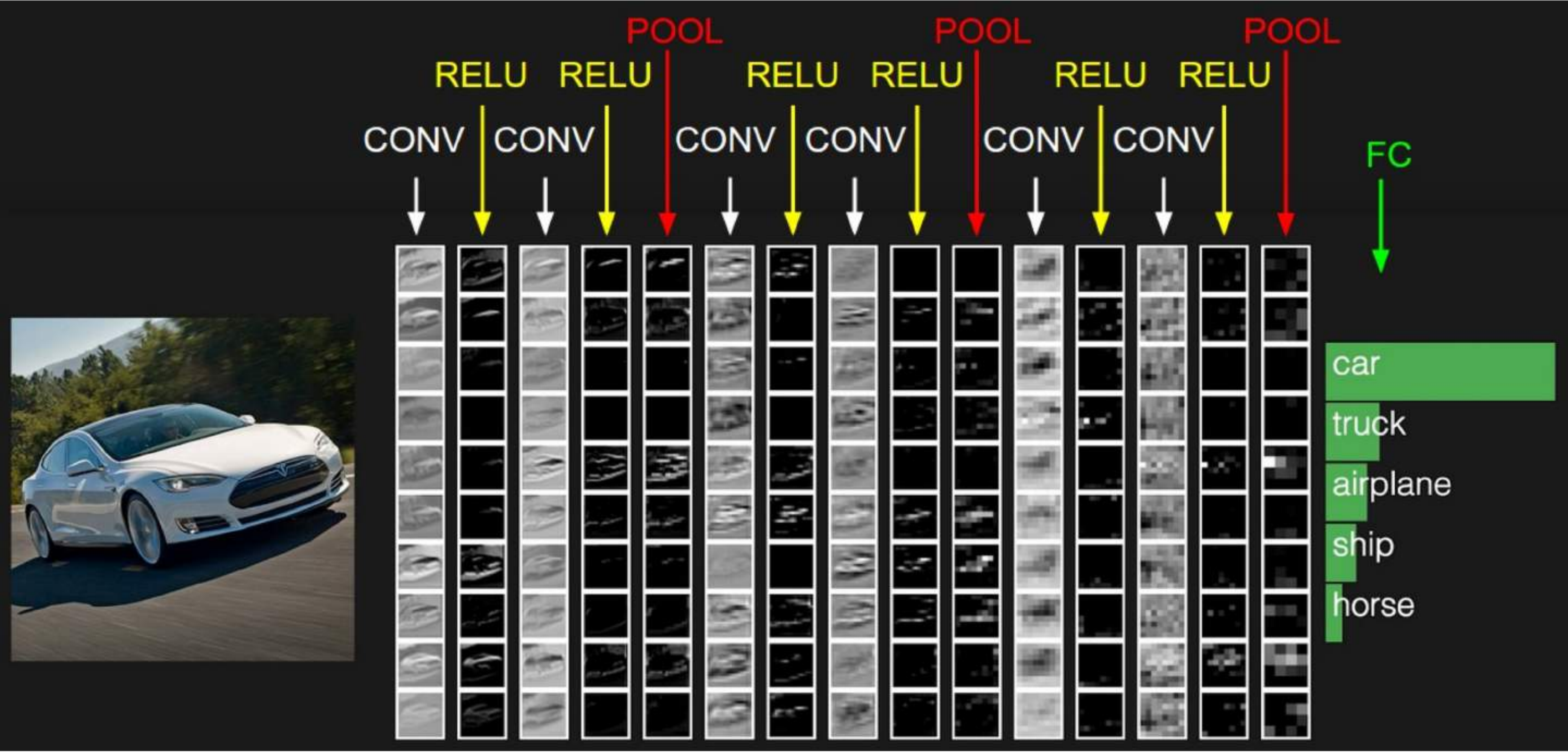
f

1000 numbers,
indicating class scores

training

Only two basic operations are involved throughout:

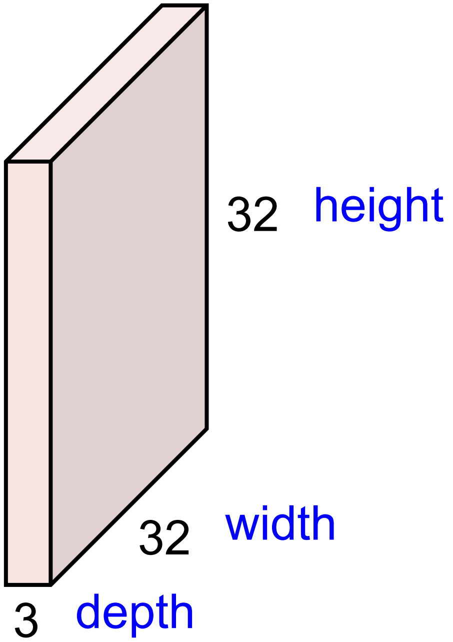
1. Dot products $w^T x$
2. Max operations max(.)



e.g. 200K numbers → e.g. 10 numbers

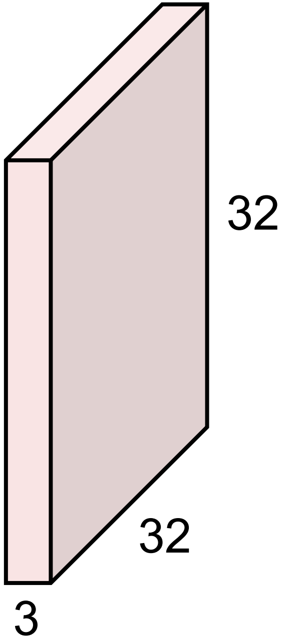
Convolution Layer

32x32x3 image

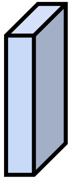


Convolution Layer

32x32x3 image



5x5x3 filter



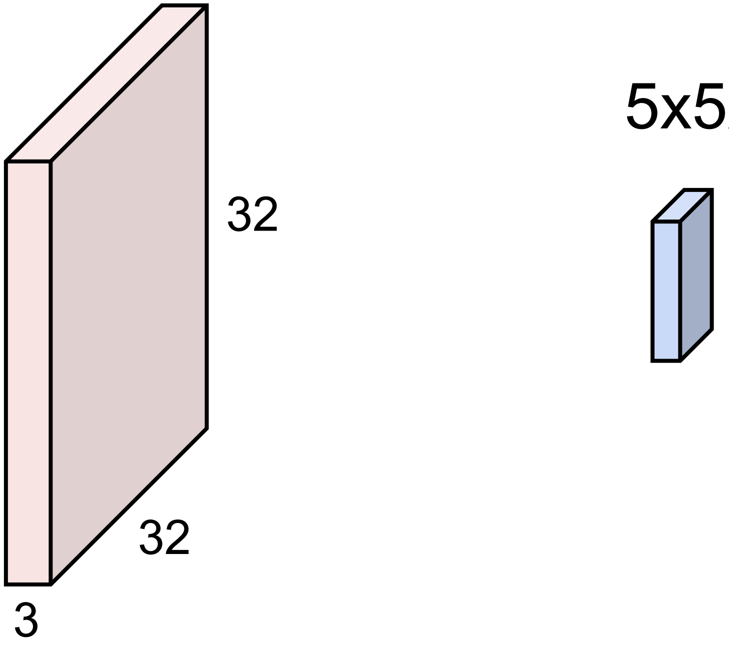
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

Filters always extend the full depth of the input volume

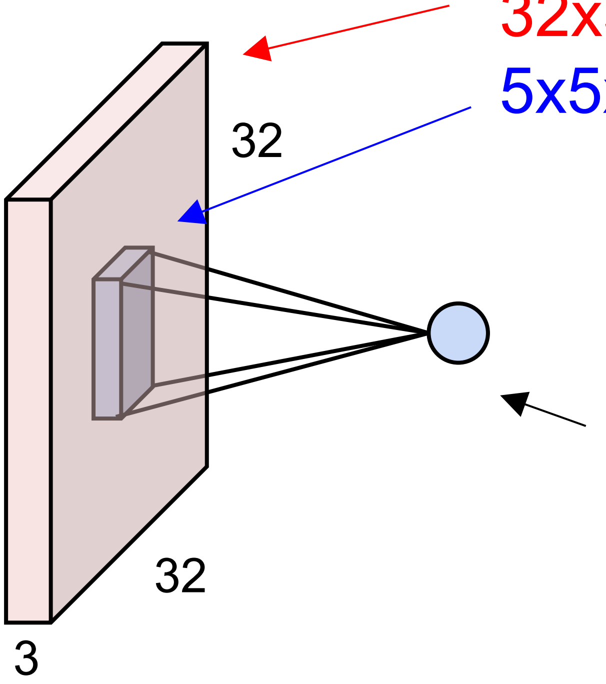
32x32x3 image

5x5x3 filter



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

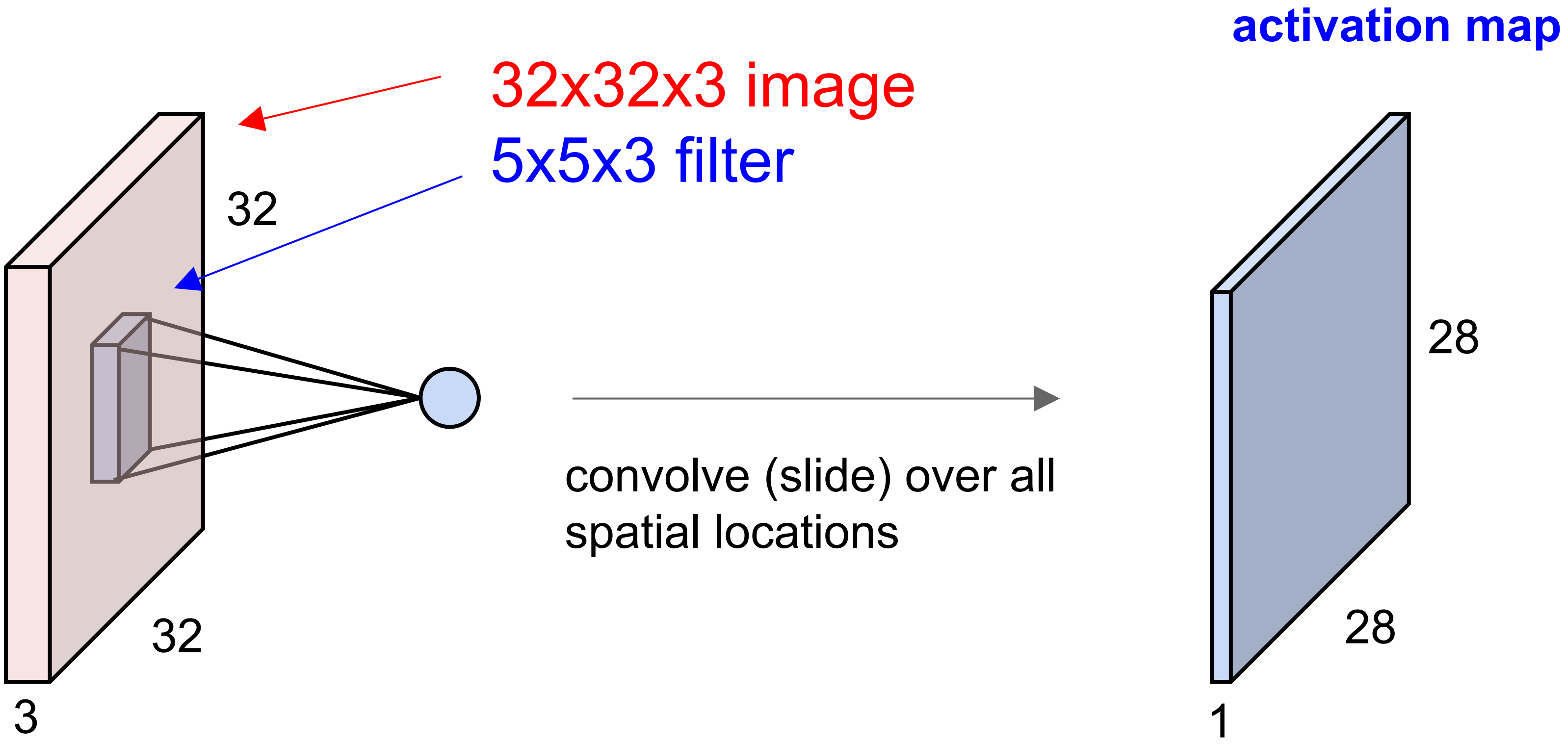
Convolution Layer



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)

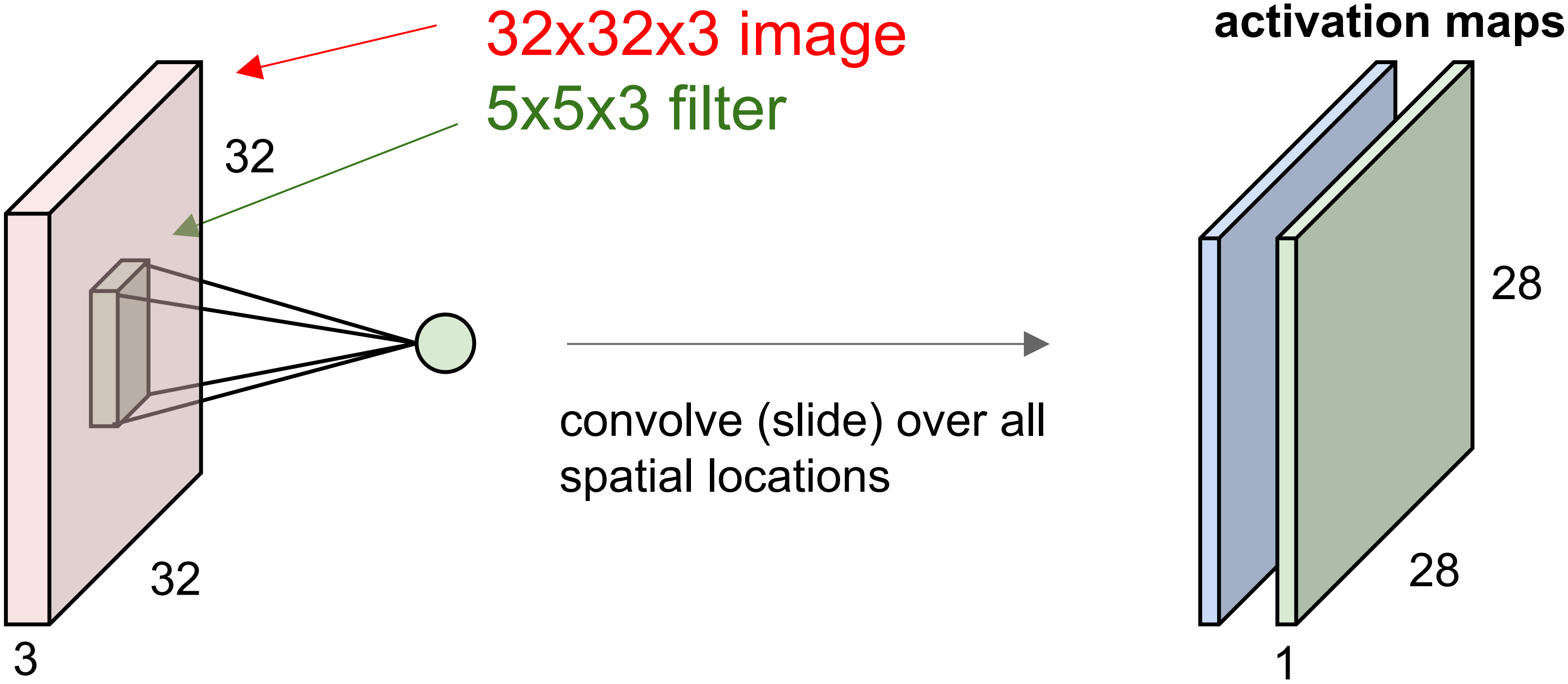
$$w^T x + b$$

Convolution Layer

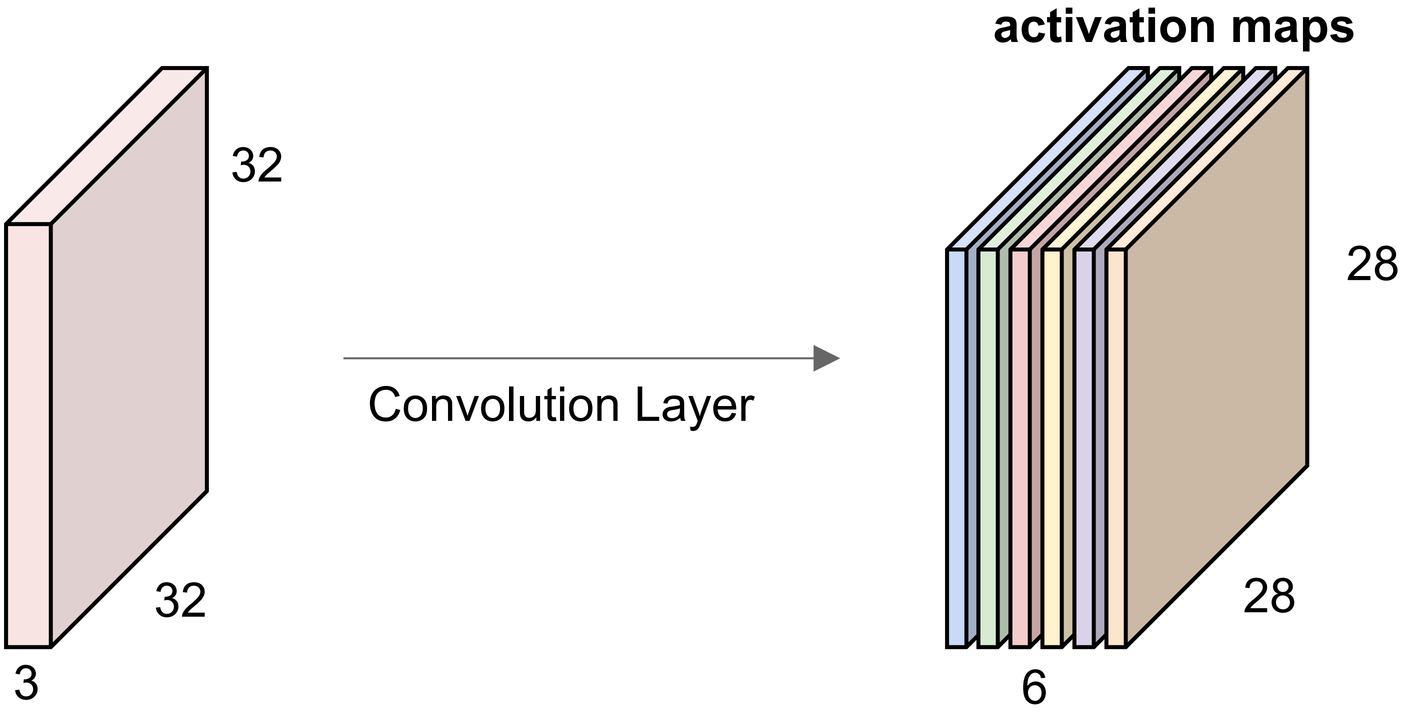


Convolution Layer

consider a second, green filter

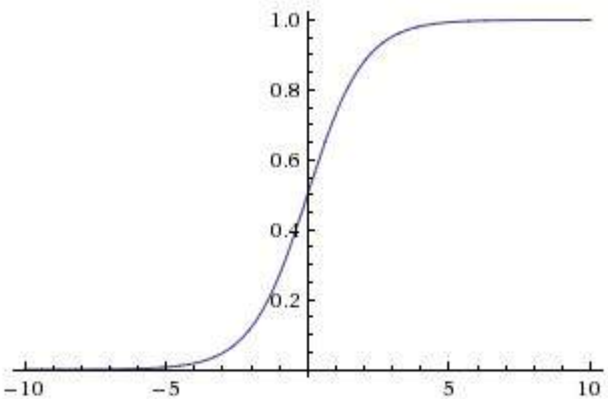


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

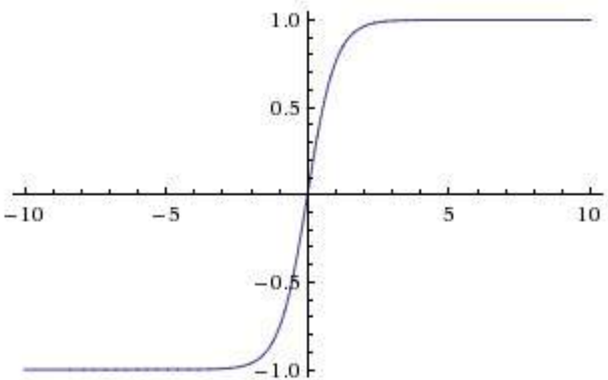


We stack these up to get a "new image" of size 28x28x6!

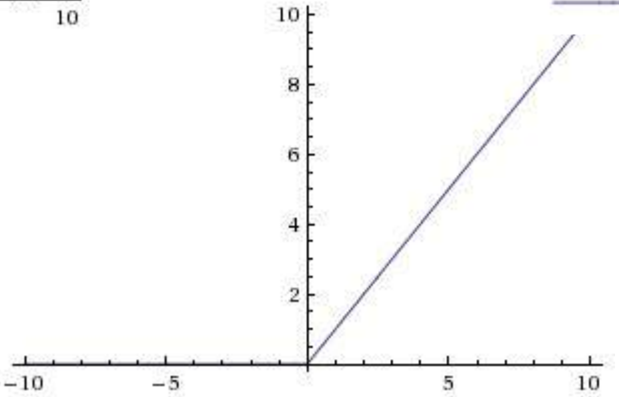
ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Sigmoid

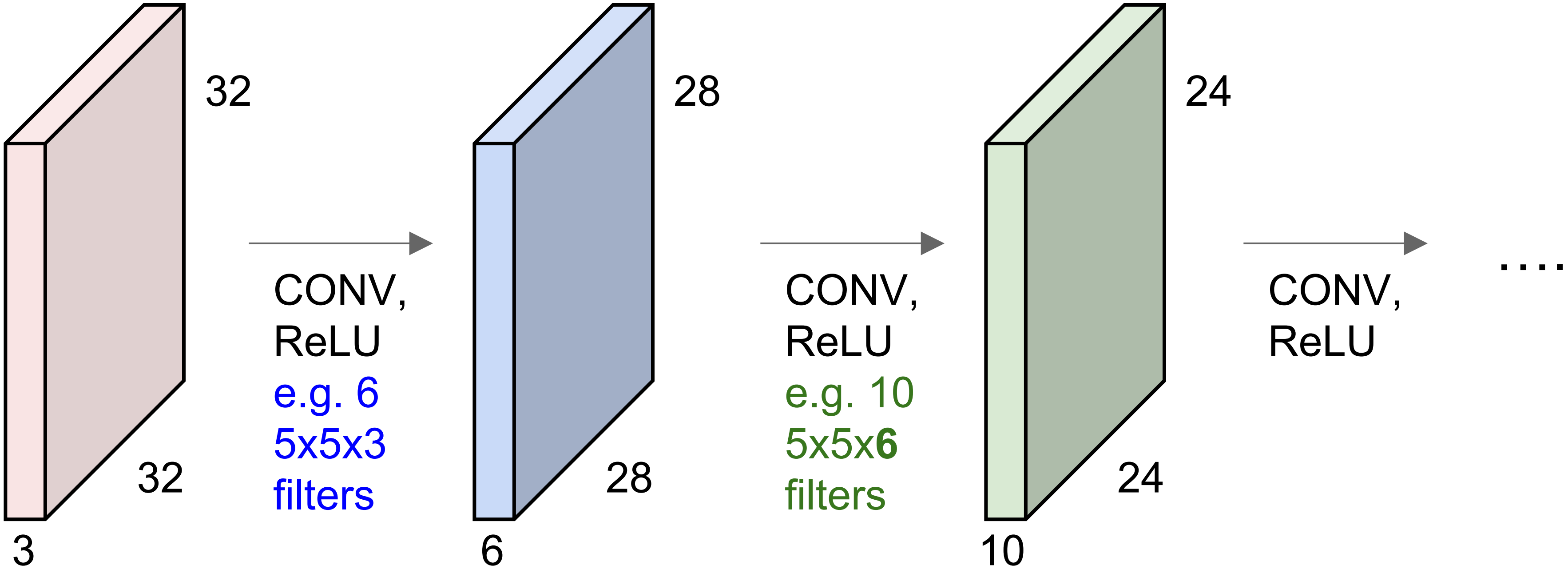


Tanh

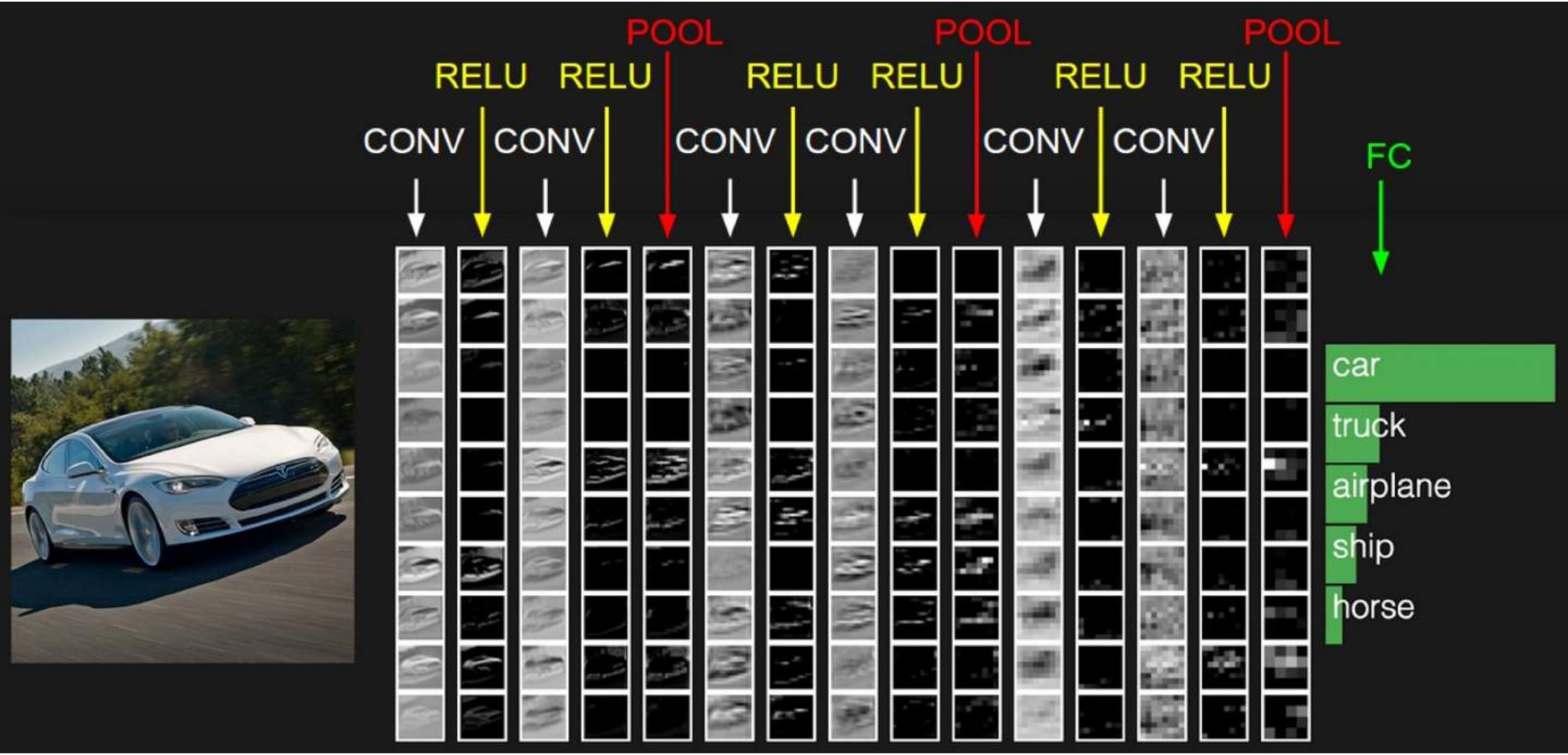


ReLU

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

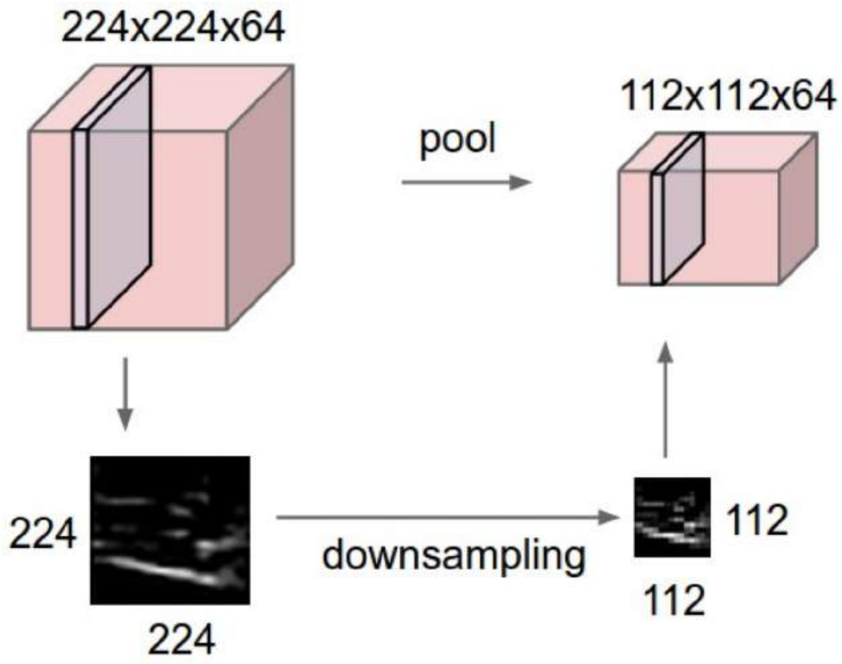


two more layers to go: POOL/FC

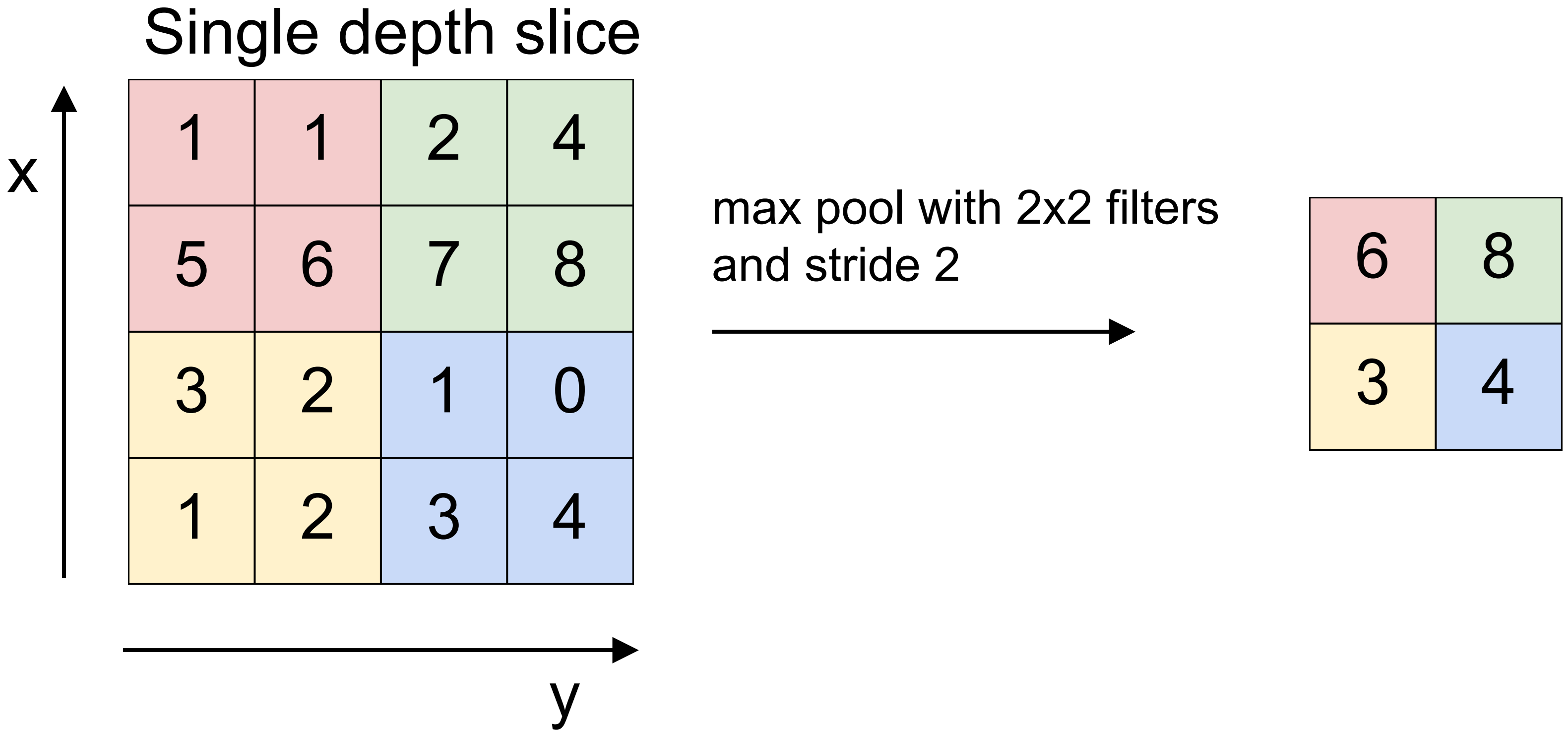


Pooling layer

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

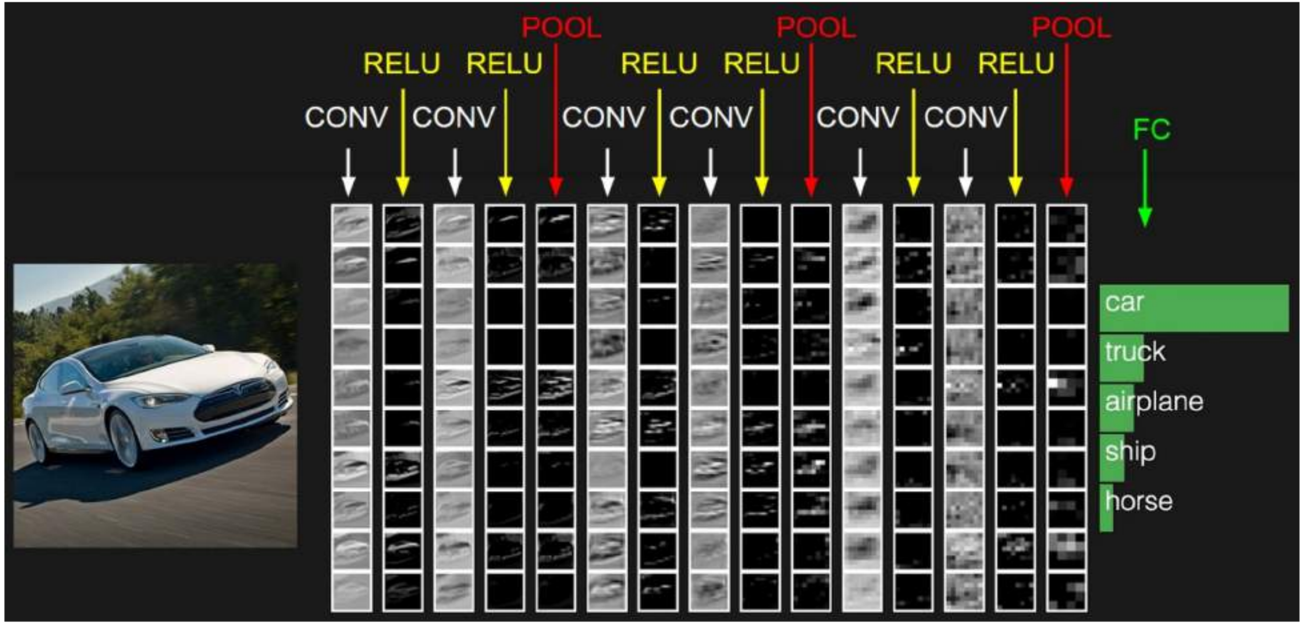


MAX POOLING


Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2
6 8
3 4

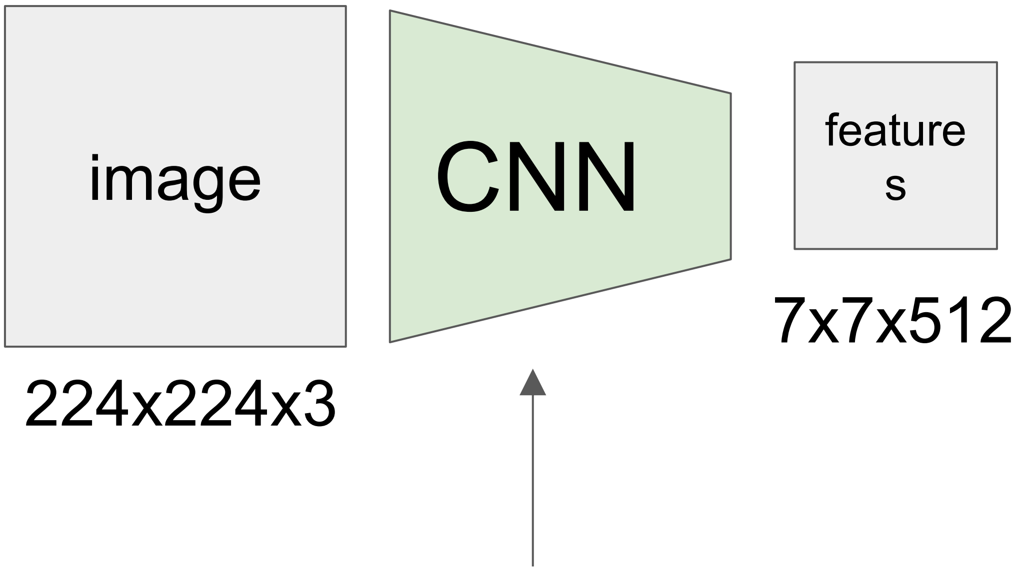
Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



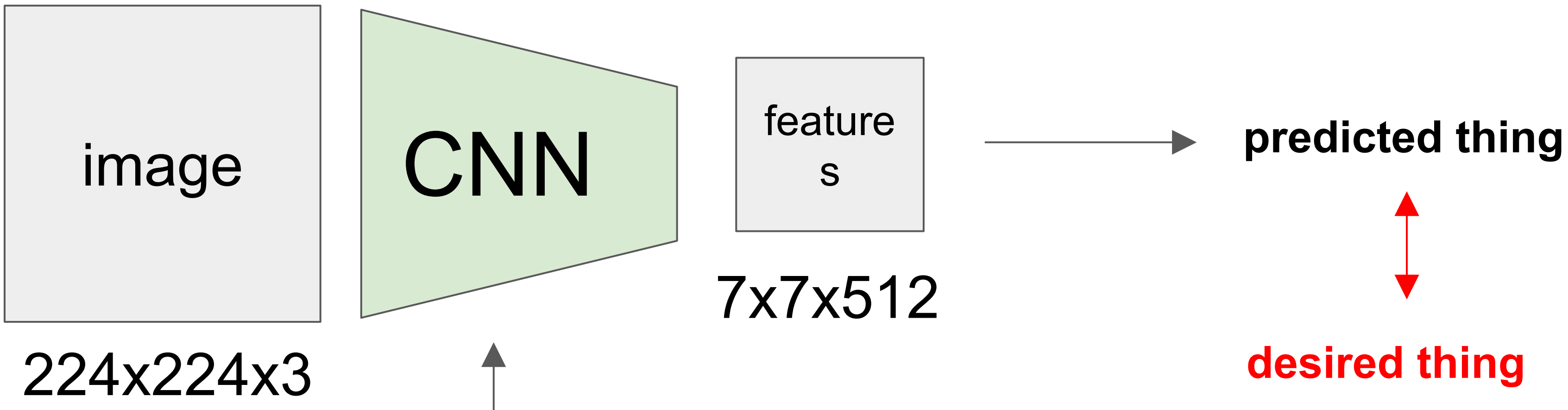
Addressing other tasks...

Addressing other tasks...



A block of compute with a few million calculations.

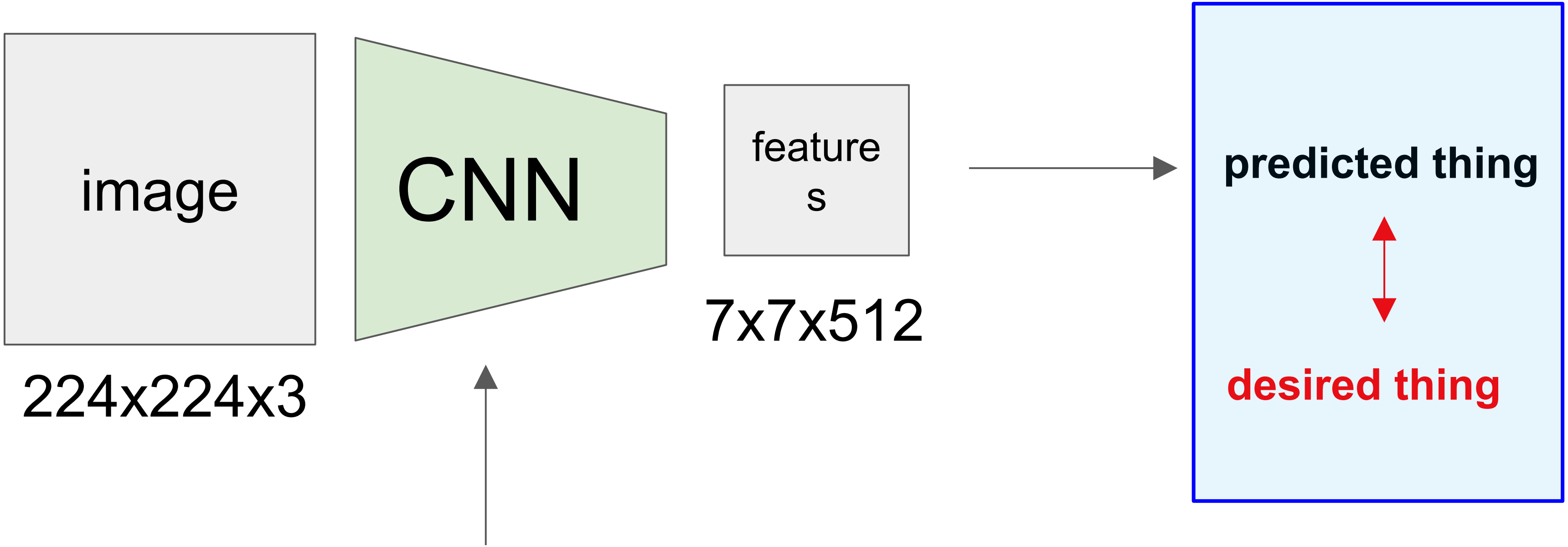
Addressing other tasks...



A block of compute with a few million calculations.

Addressing other tasks...

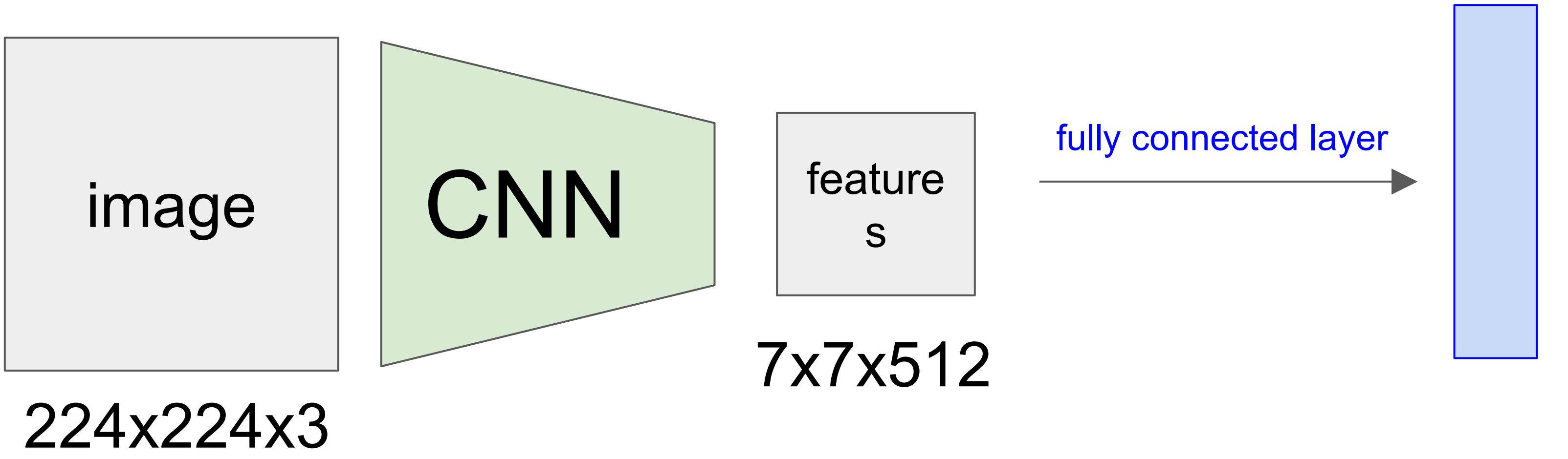
this part changes
from task to task



A block of compute with a few
million parameters.

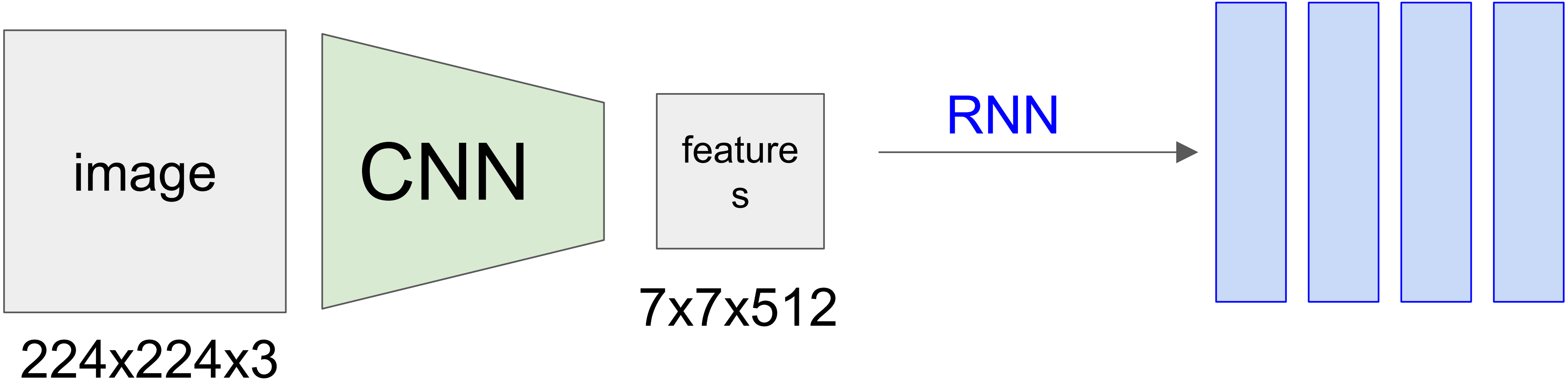
Image Classification

thing = a vector of probabilities for different classes



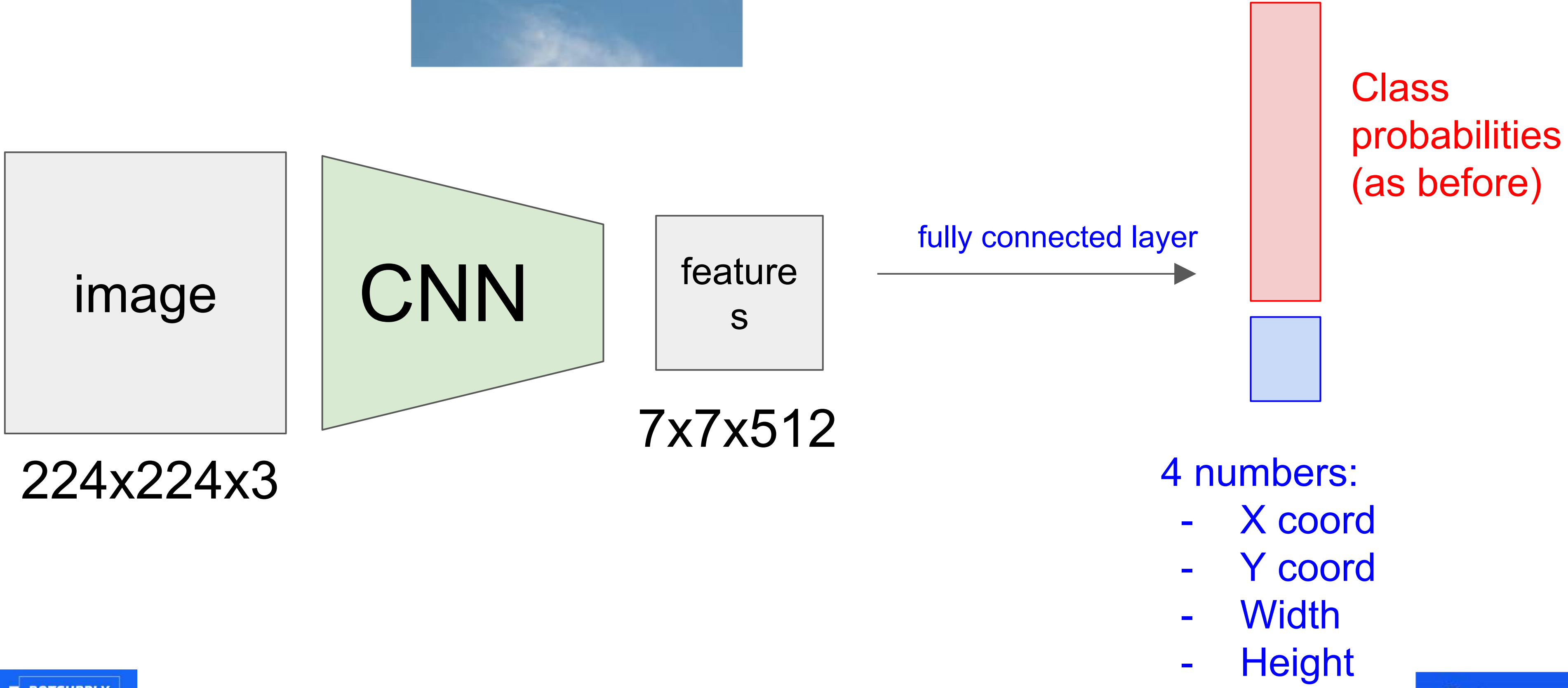
e.g. vector of 1000 numbers giving probabilities for different classes.

Image Captioning



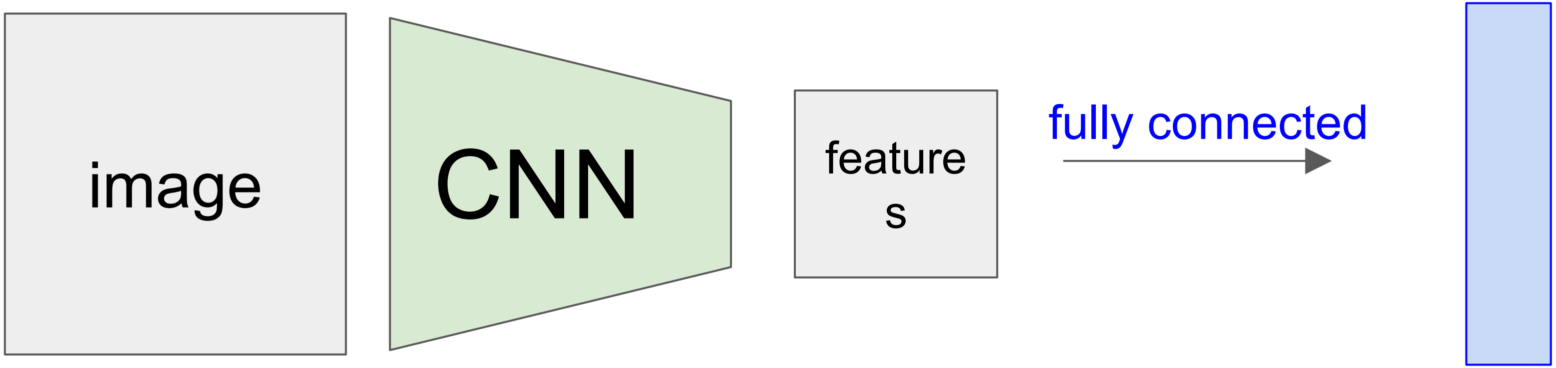
A sequence of 10,000-dimensional vectors giving probabilities of different words in the caption.

Localization



Reinforcement Learning

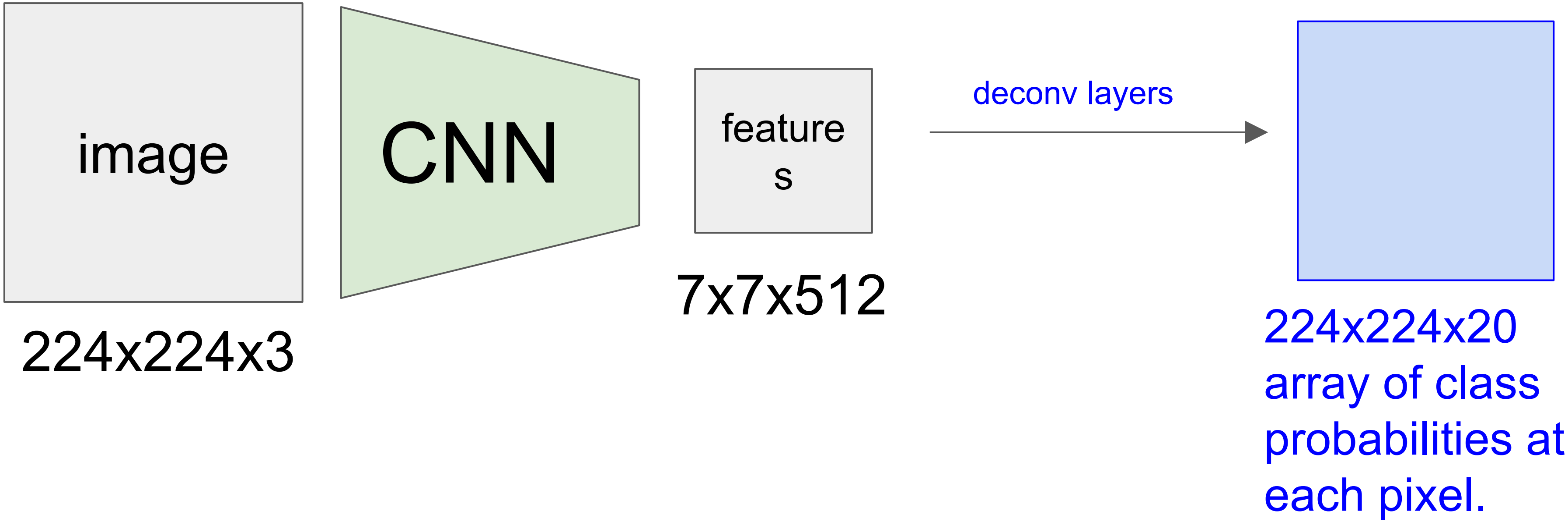
Mnih et al. 2015



160x210x3

e.g. vector of 8 numbers giving probability of wanting to take any of the 8 possible ATARI actions.

Segmentation



Hands on word2vec

Colab link : https://goo.gl/7H7mVo

"Deep learning" offer us great power - and pose unique risks.
Can we Vectorise them?

Thank you!

@hellorahulk