CONVOLUTIONAL NEURAL NETWORKS

CONTENTS

RECAP: WHAT IS WRONG WITH FULLY CONNECTED DEEP NEURAL NETWORKS?

- Consider an image classification problem (indoor/outdoor scene).
- Image size = 64 x 64 x 3 = 12288 (input layer)
- Suppose two hidden layers h1 = 1000 neurons, h2 = 500 neurons
- Outputs = 2 neurons (one for each class)
- How many parameters would we need to train??

RECAP: WHAT IS WRONG WITH FULLY CONNECTED DEEP NEURAL NETWORKS?

- Consider an image classification problem (indoor/outdoor scene).
- Image size = 64 x 64 x 3 = 12288 (input layer)
- Suppose two hidden layers h1 = 1000 neurons, h2 = 500 neurons
- Outputs = 2 neurons (one for each class)
- How many parameters would we need to train??

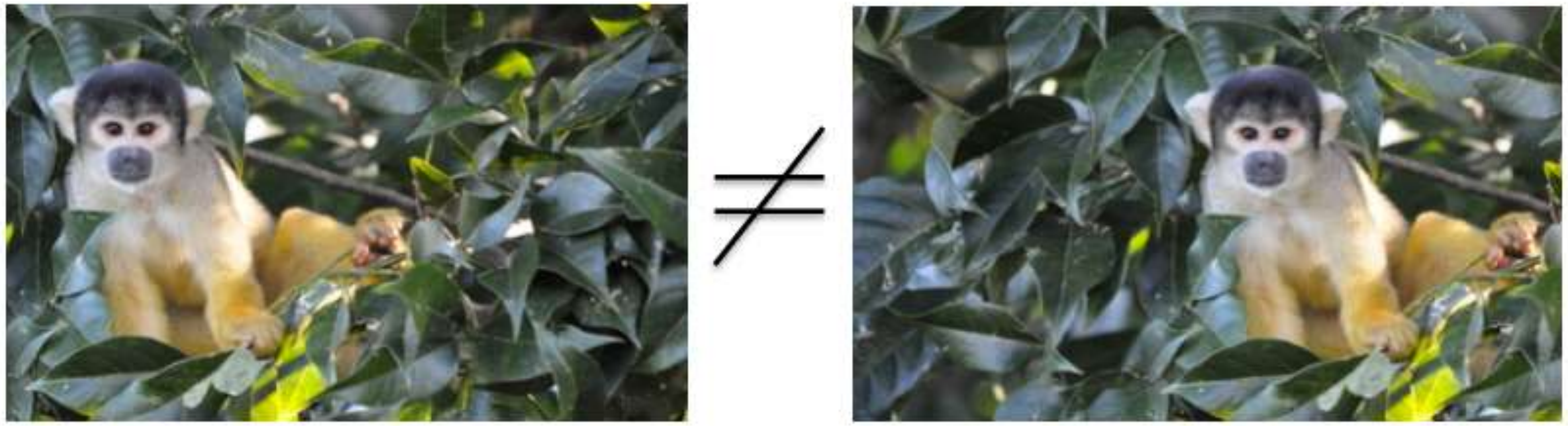
- 12288 x 1000 + 1000 x 500 + 500 x 2 = 12789000
  (12.7 million) very small image and very small network

- What if the image size = 256 x 256 x 3, and network has 4 layers with 1000 neurons and 1000 outputs?

RECAP: WHAT IS WRONG WITH FULLY CONNECTED DEEP NEURAL NETWORKS?

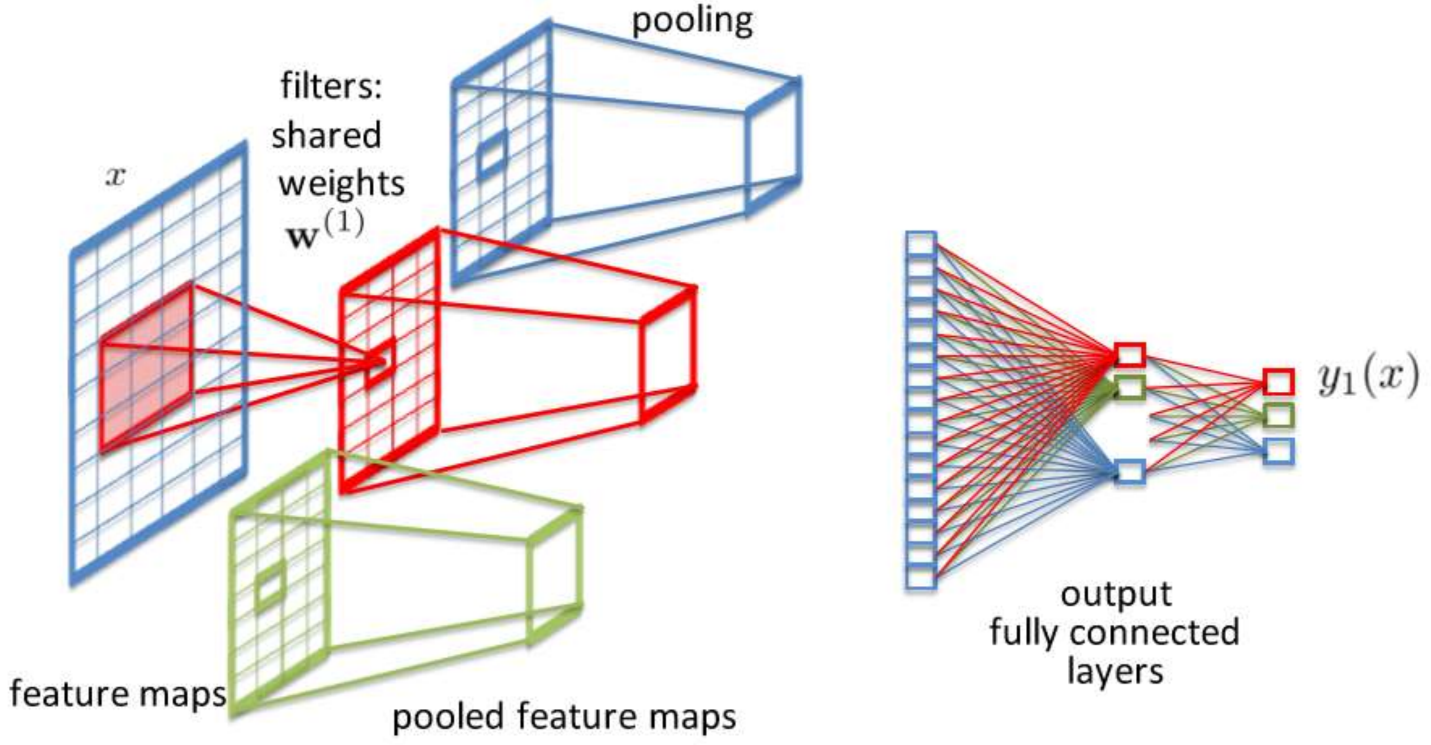
Too many parameters:
MLP[1000-1000-1000] - 2 000 000 parameters
require too much training data

Wasteful and do not generalize:
no spatial invariance for images



RECAP: THE SOLUTION?

- Convolutional neural networks (CNN)



BRIEF HISTORY: CNN

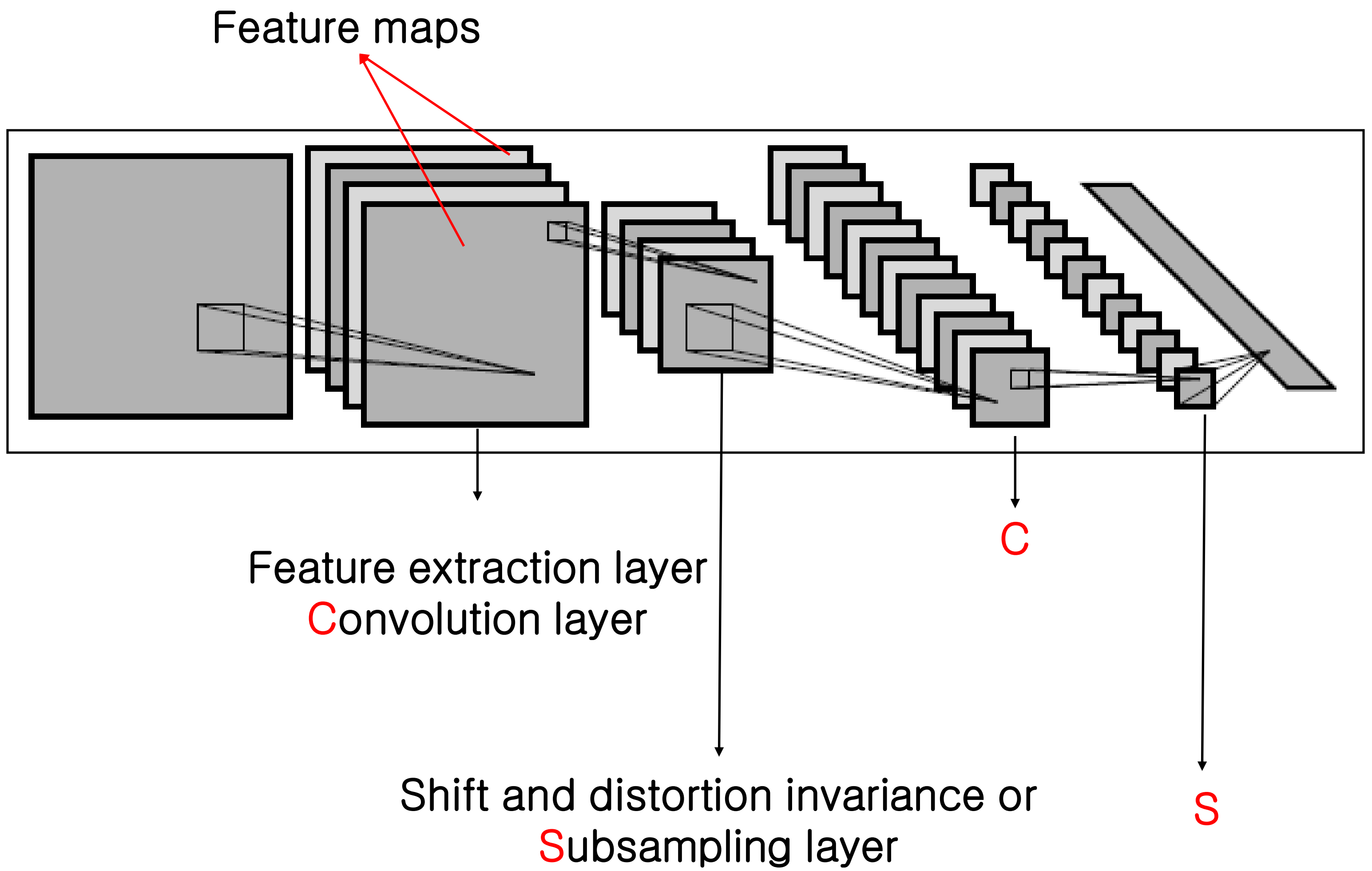
Yann LeCun, Professor of Computer Science
The Courant Institute of Mathematical Sciences
New York University
Room 1220, 715 Broadway, New York, NY 10003, USA.
(212)998-3283 yann@cs.nyu.edu

- In 1995, Yann LeCun and Yoshua Bengio introduced the concept of convolutional neural networks.

CONVOLUTIONAL NEURAL NETWORK

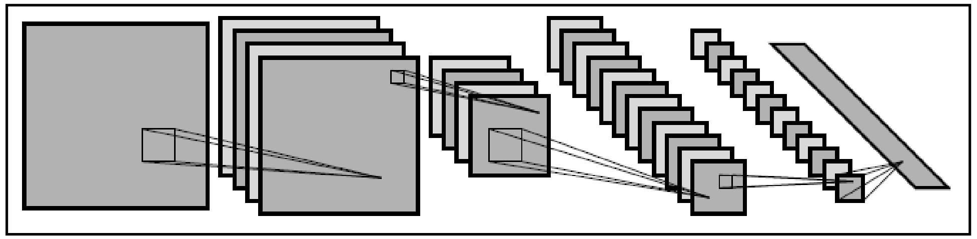
- CNN’s Were neuro-biologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.
- They designed a network structure that implicitly extracts relevant features.
- Convolutional Neural Networks are a special kind of multi-layer neural networks.
- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability (such as handwritten characters).

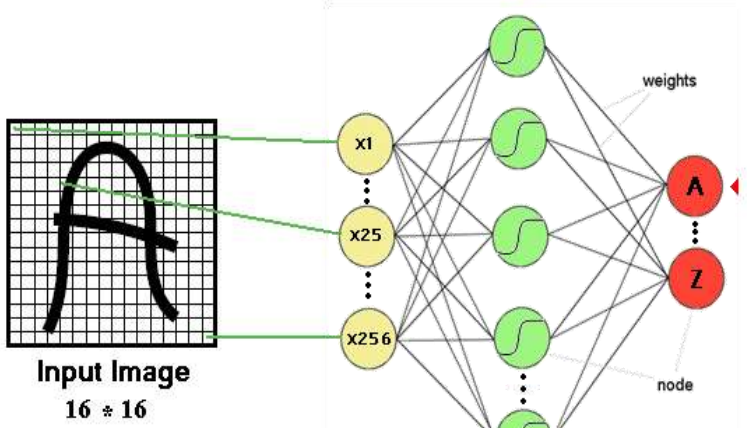
CNN BUILDING BLOCKS

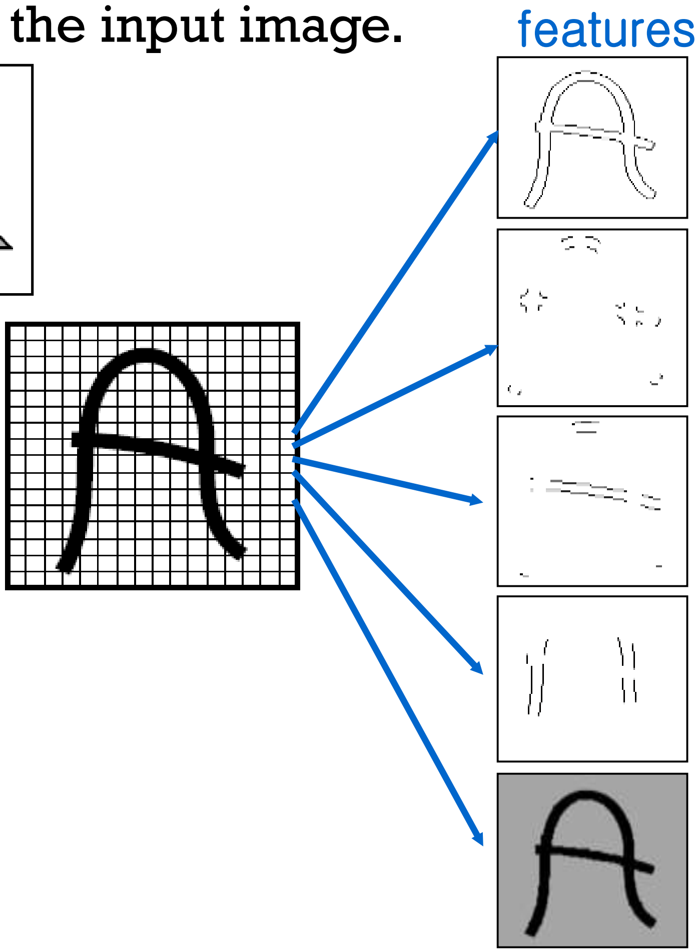


CONVOLUTION LAYER

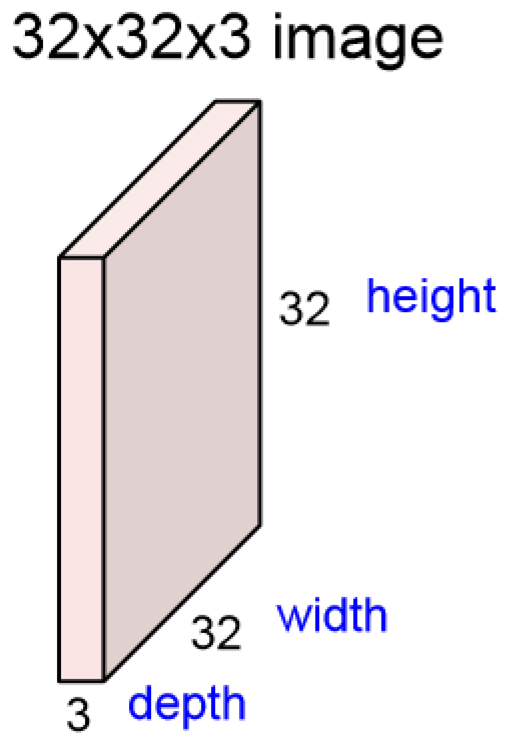
- Detect the same feature at different positions in the input image.



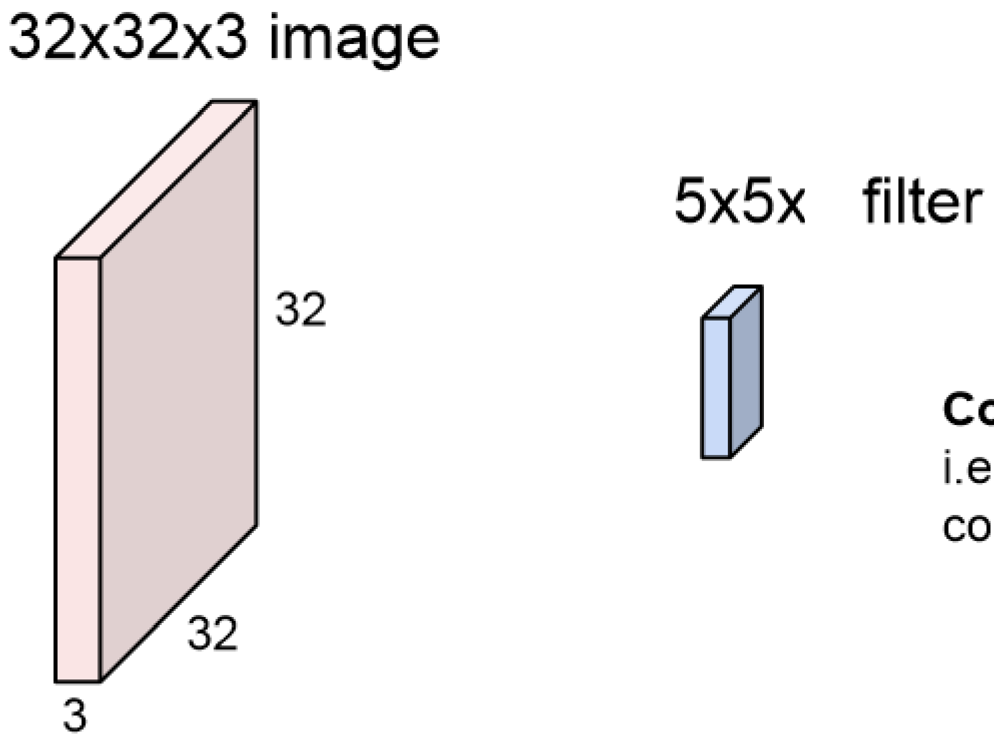




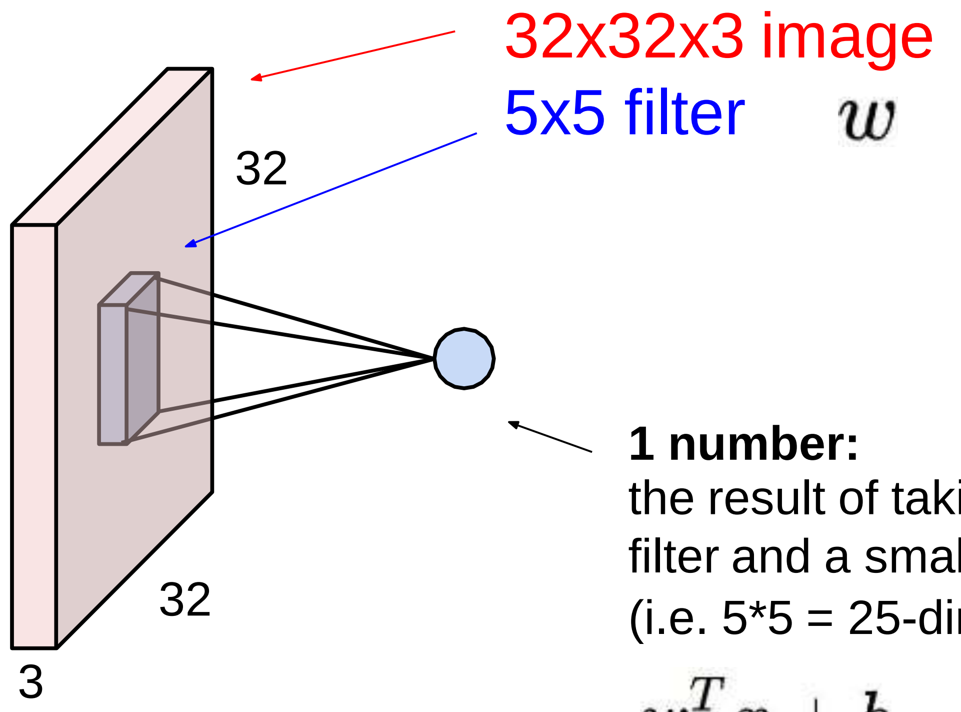
CONVOLUTION LAYER



CONVOLUTION LAYER

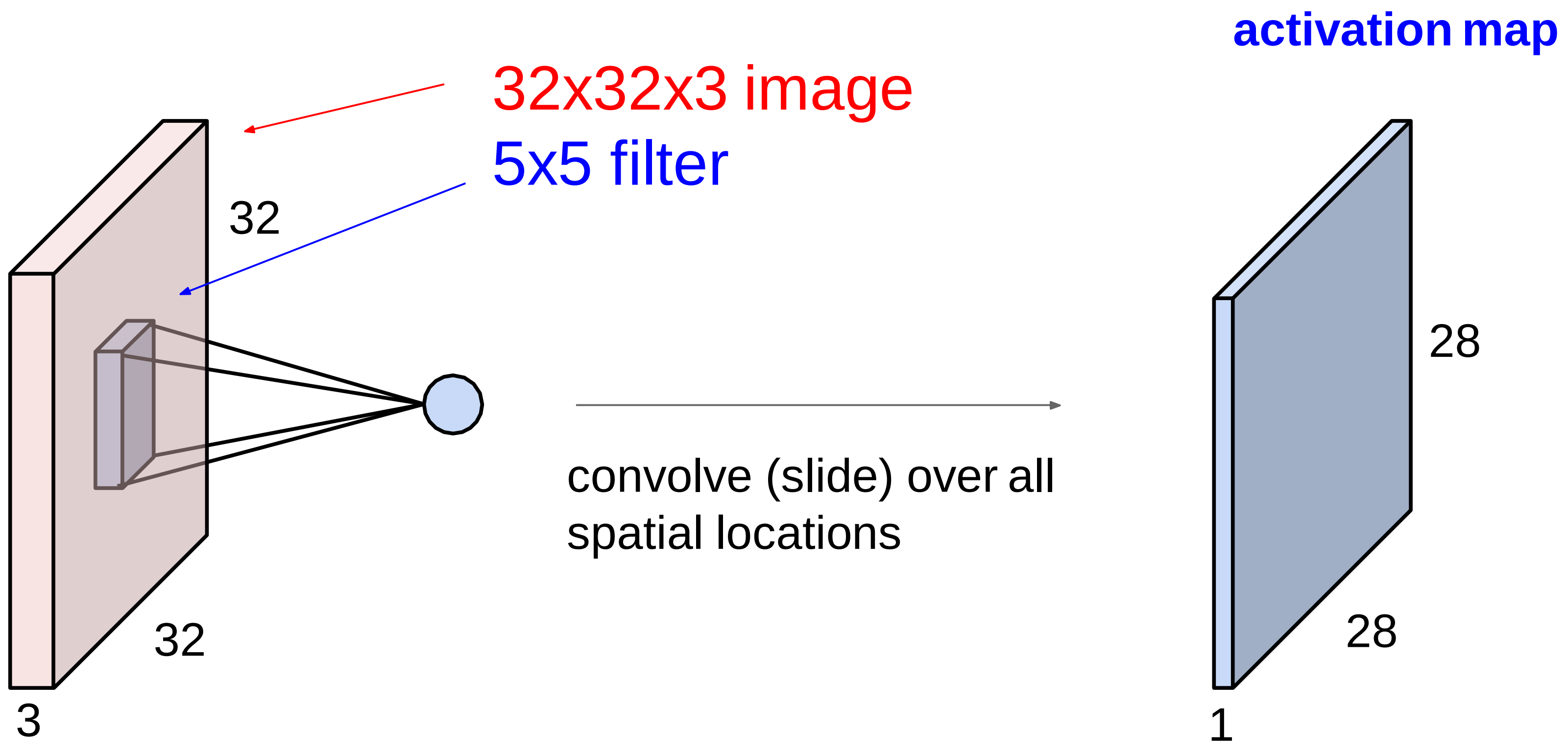


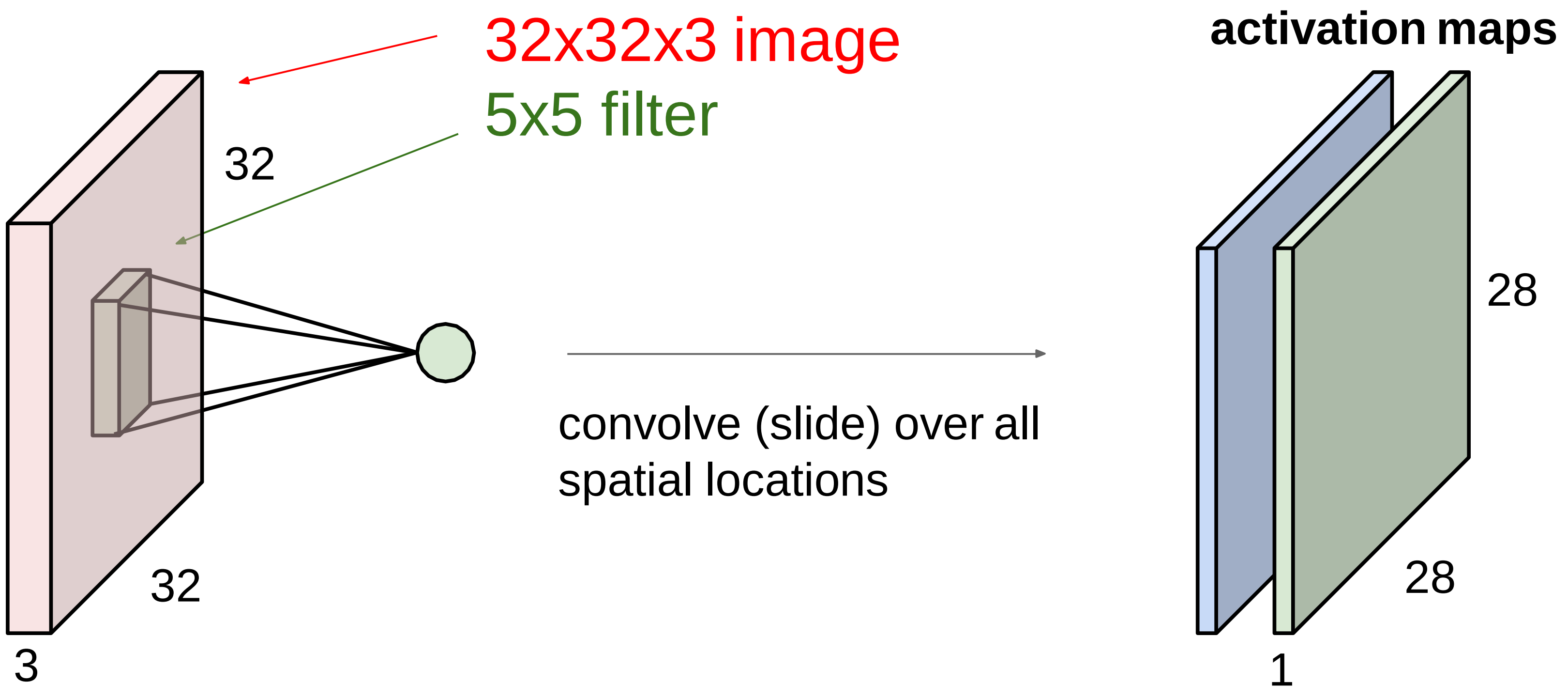
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”



1 number:
the result of taking a dot product between the filter and a small 5x5 chunk of the image
(i.e. 5*5 = 25-dimensional dot product + bias)

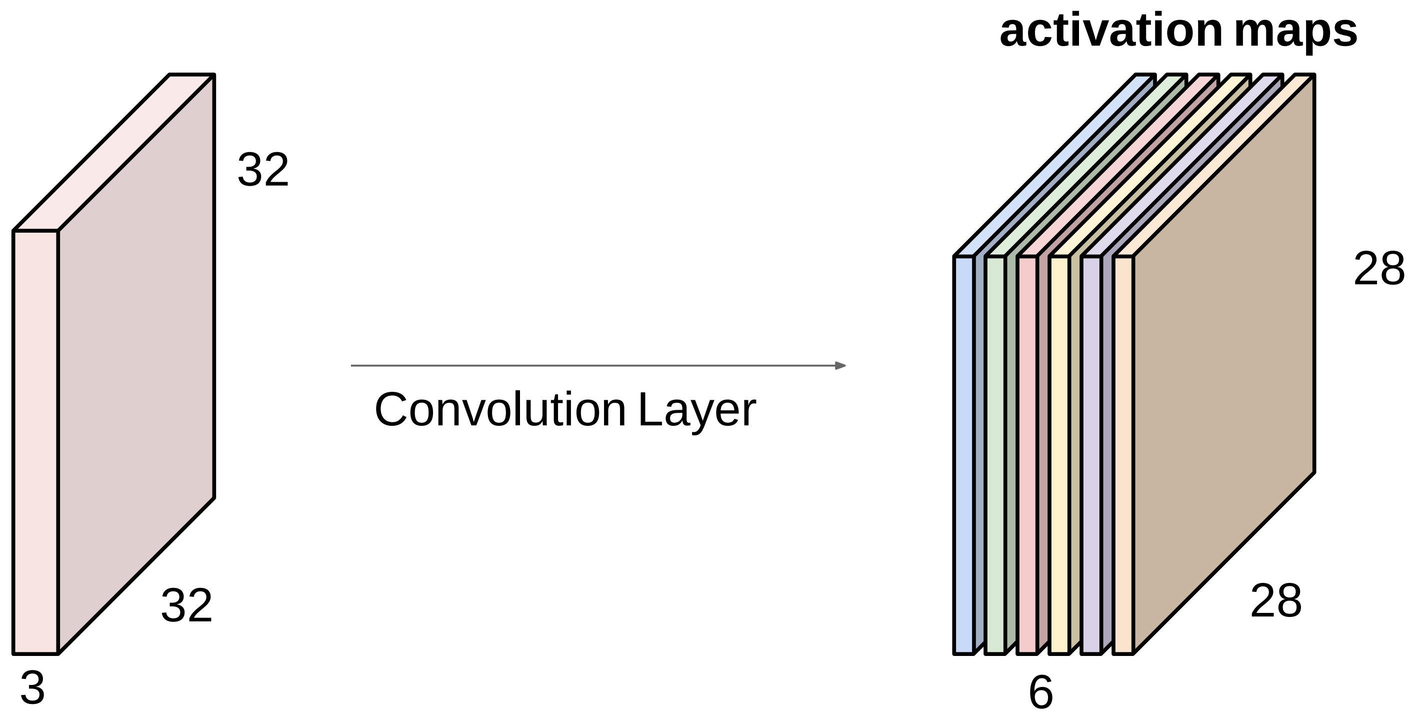
$$w^T x + b$$


activation map
32x32x3 image
5x5 filter
32
32
3
convolve (slide) over all spatial locations
28
28
1



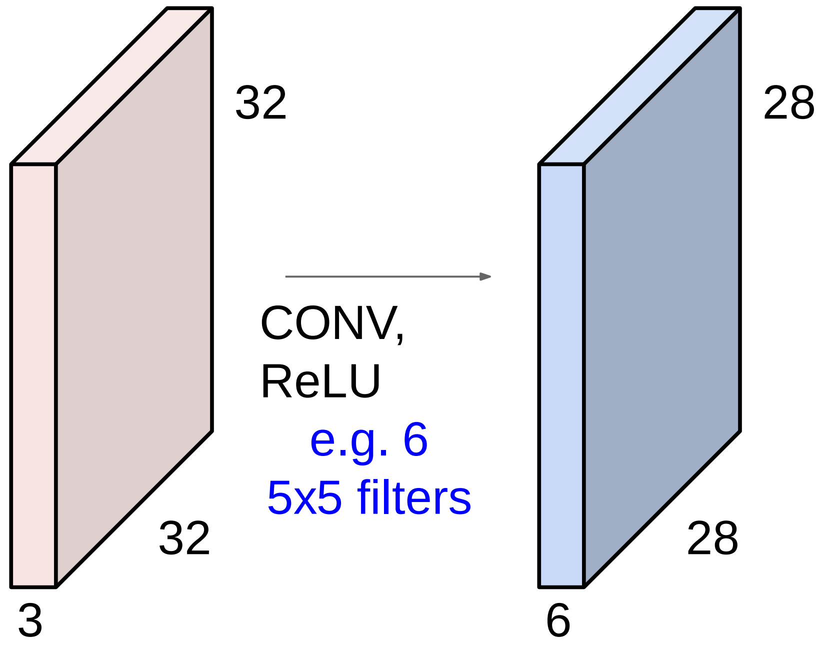
consider a second, green filter

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

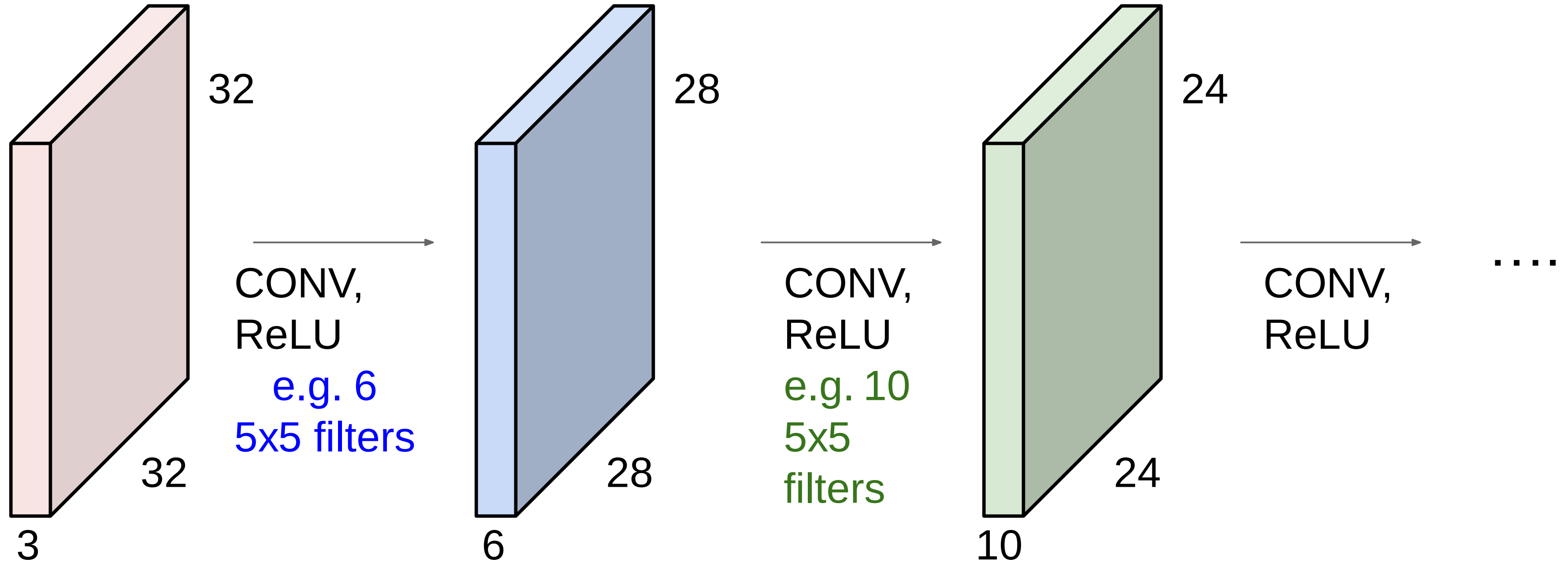


We stack these up to get a "new image" of size 28x28x6!

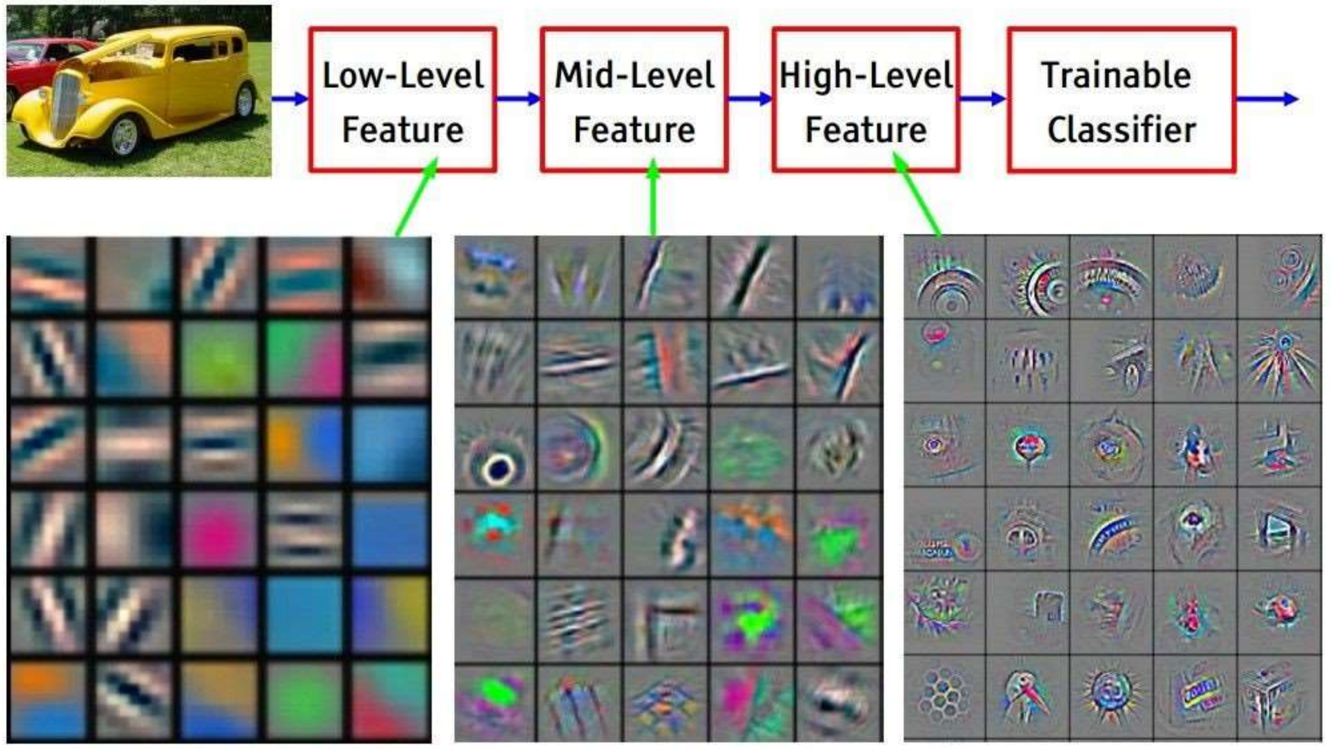
- **Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions



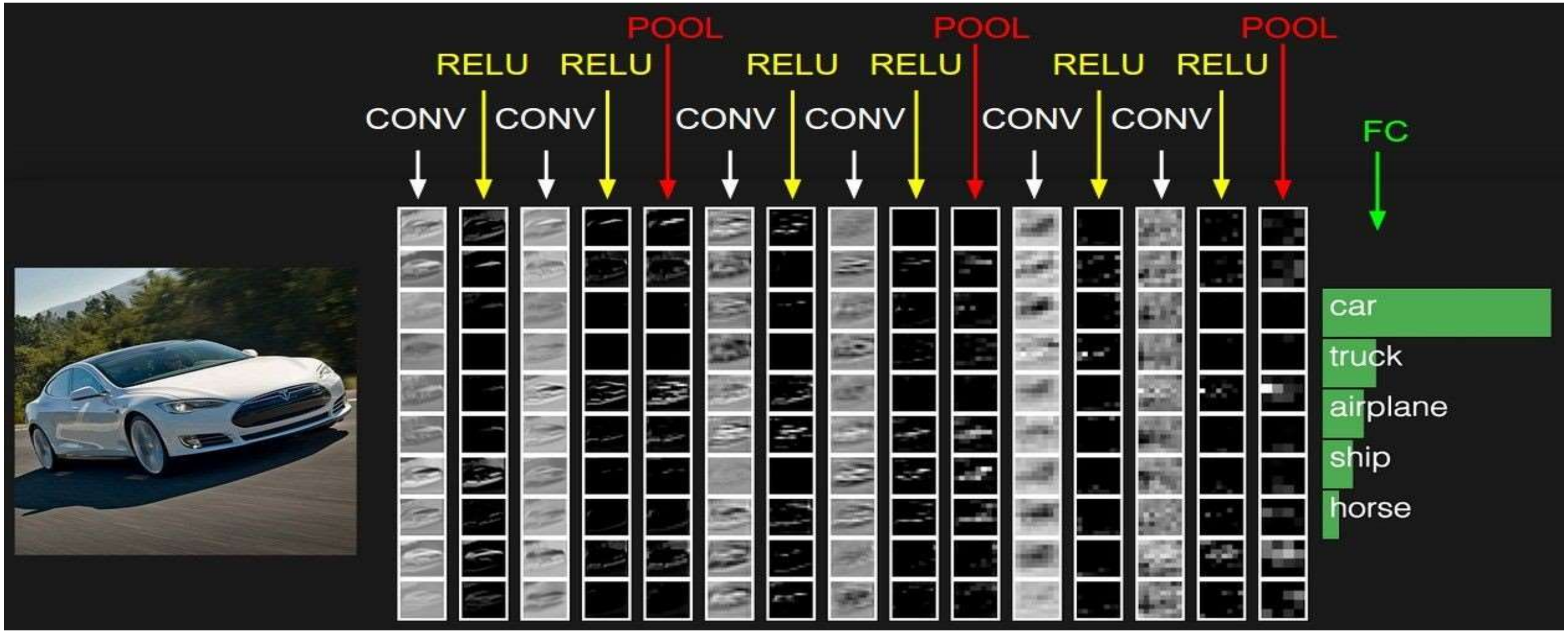
- **Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



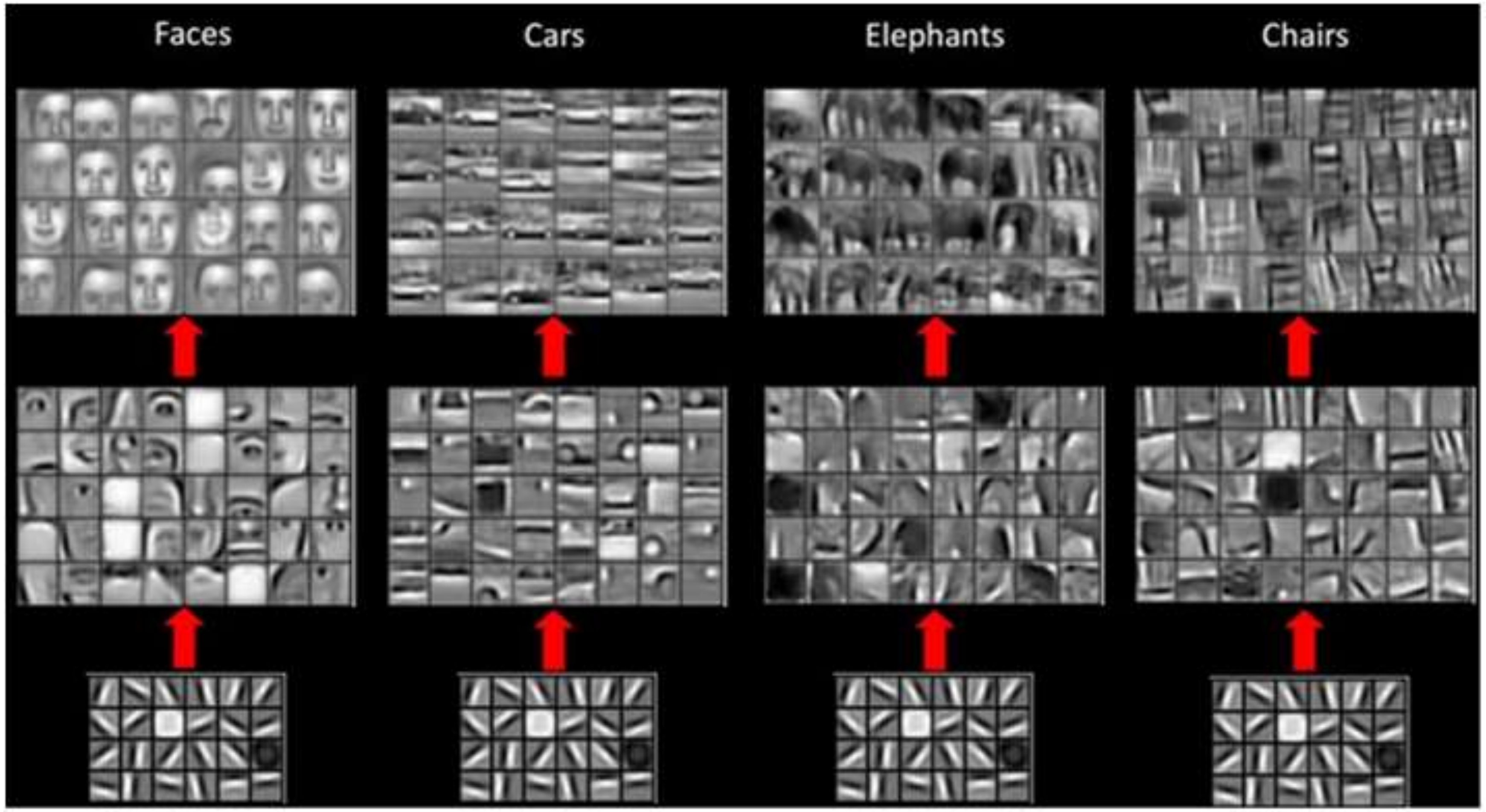
- **Preview**

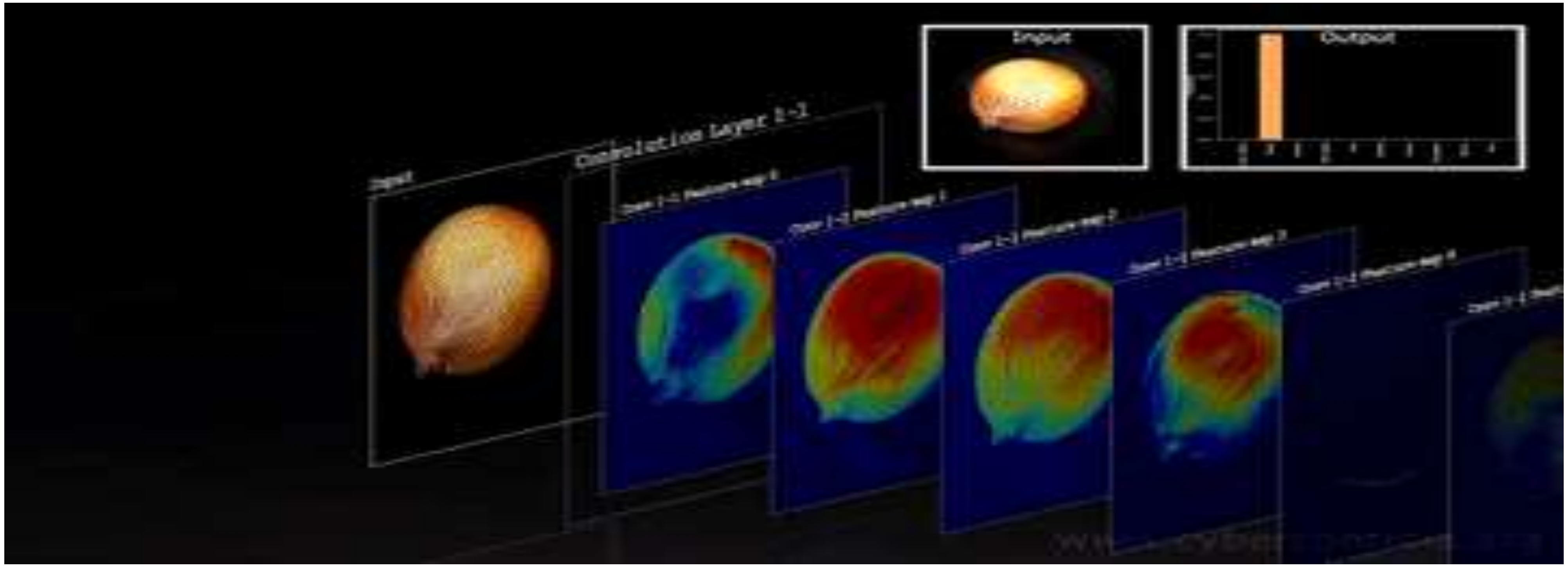


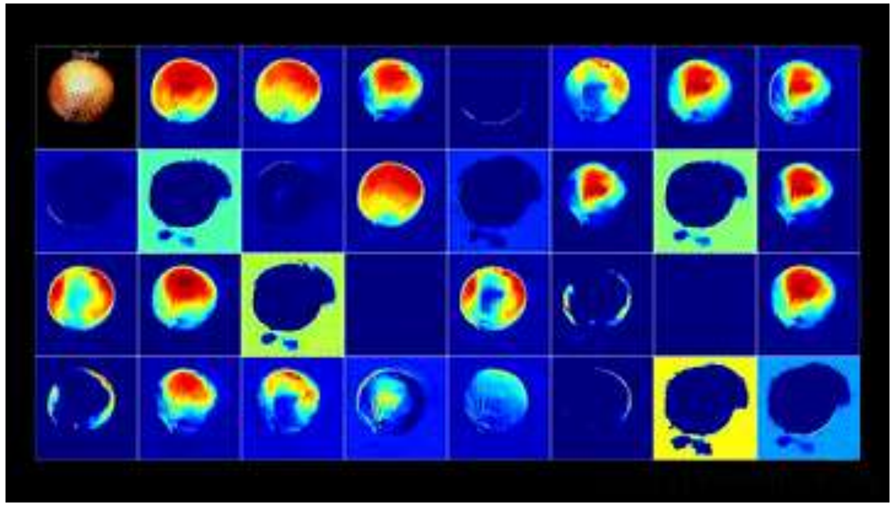
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

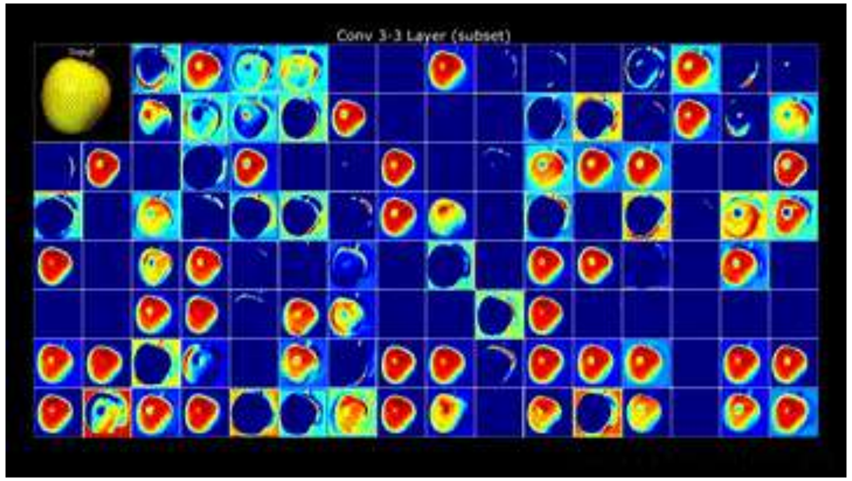


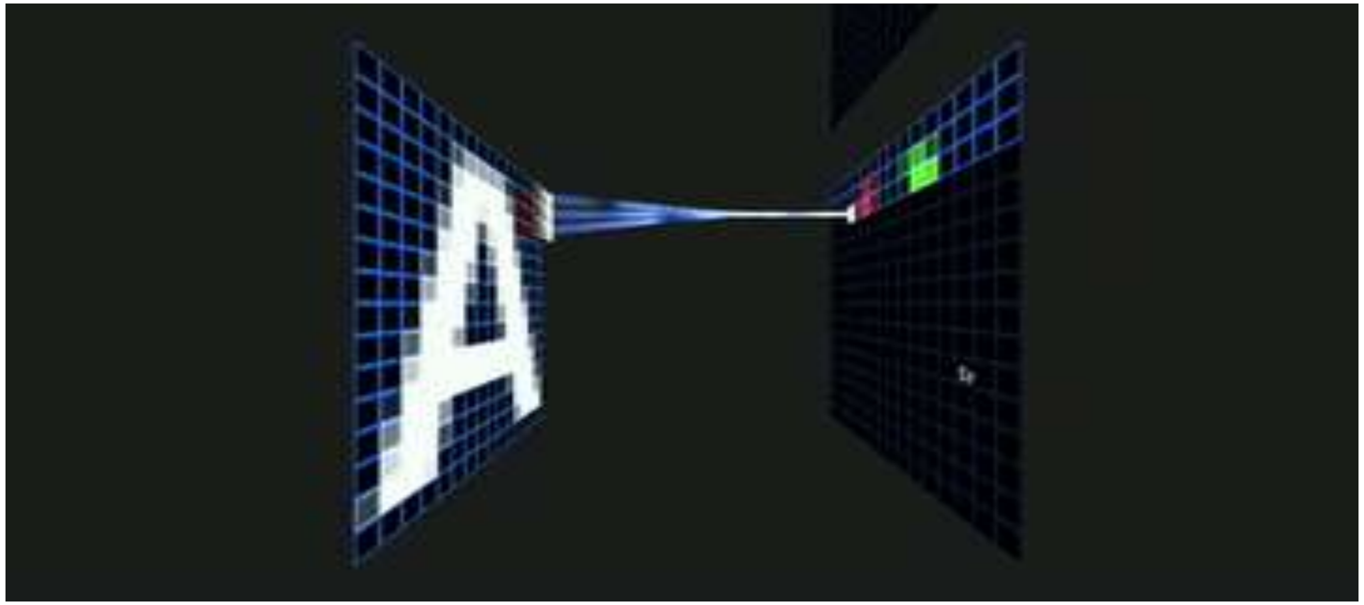
RELU
RELU
POOL
RELU
RELU
POOL
RELU
RELU
POOL
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

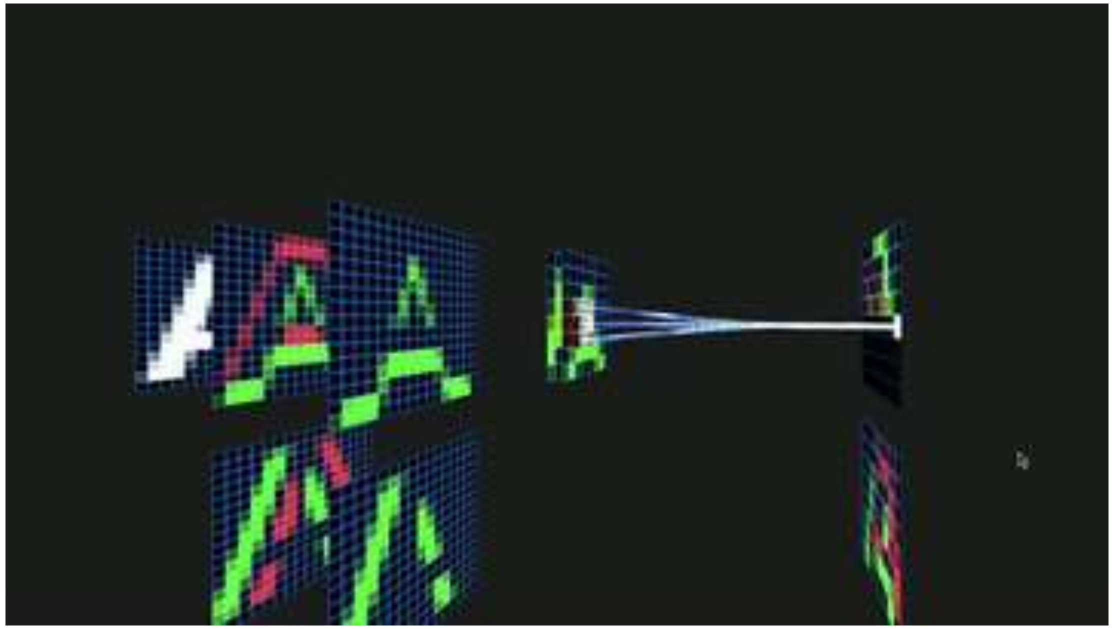

Faces
Cars
Elephants
Chairs


Input
Output
Input
Convolution Layer 1-1

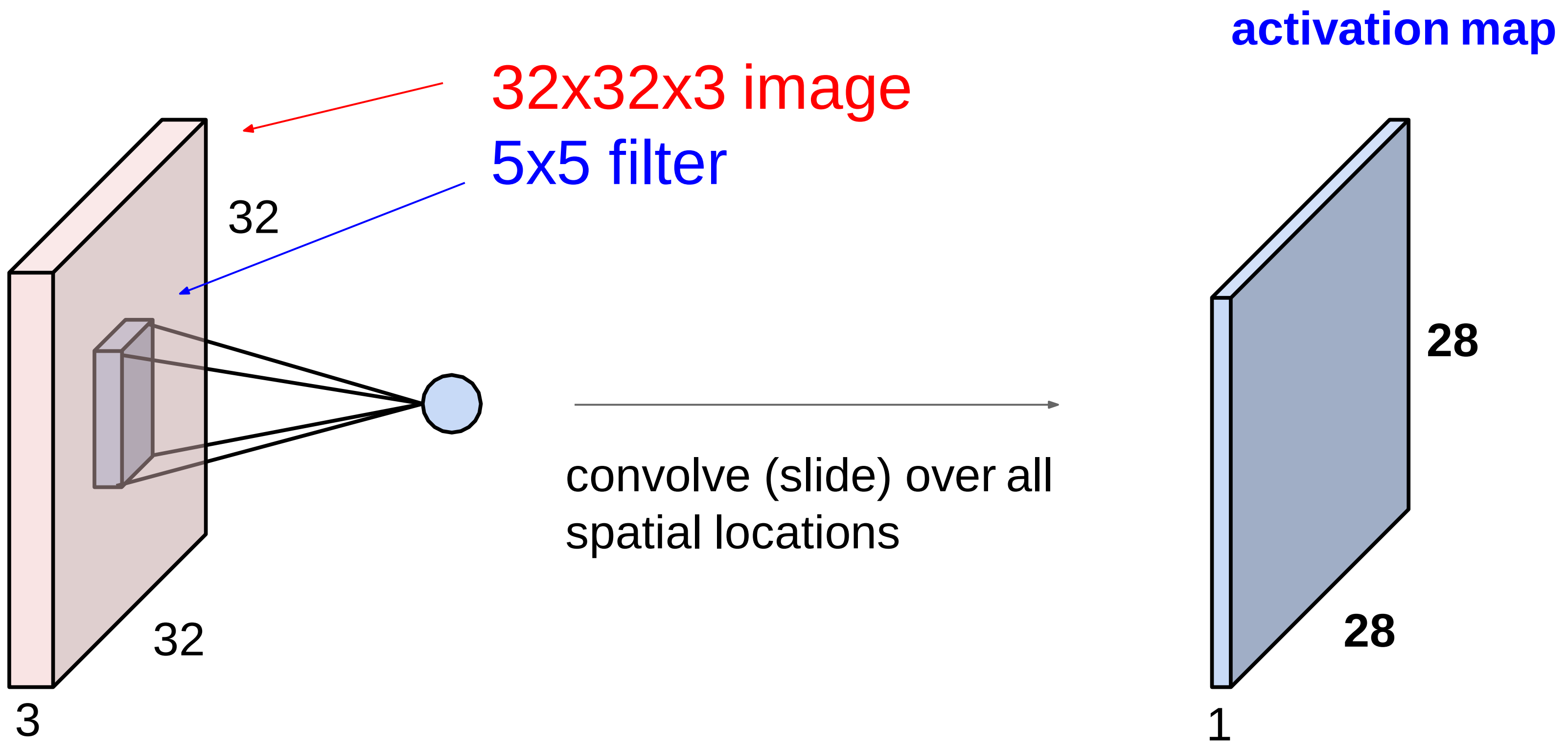

Input


Conv 3-3 Layer (subset)
Input

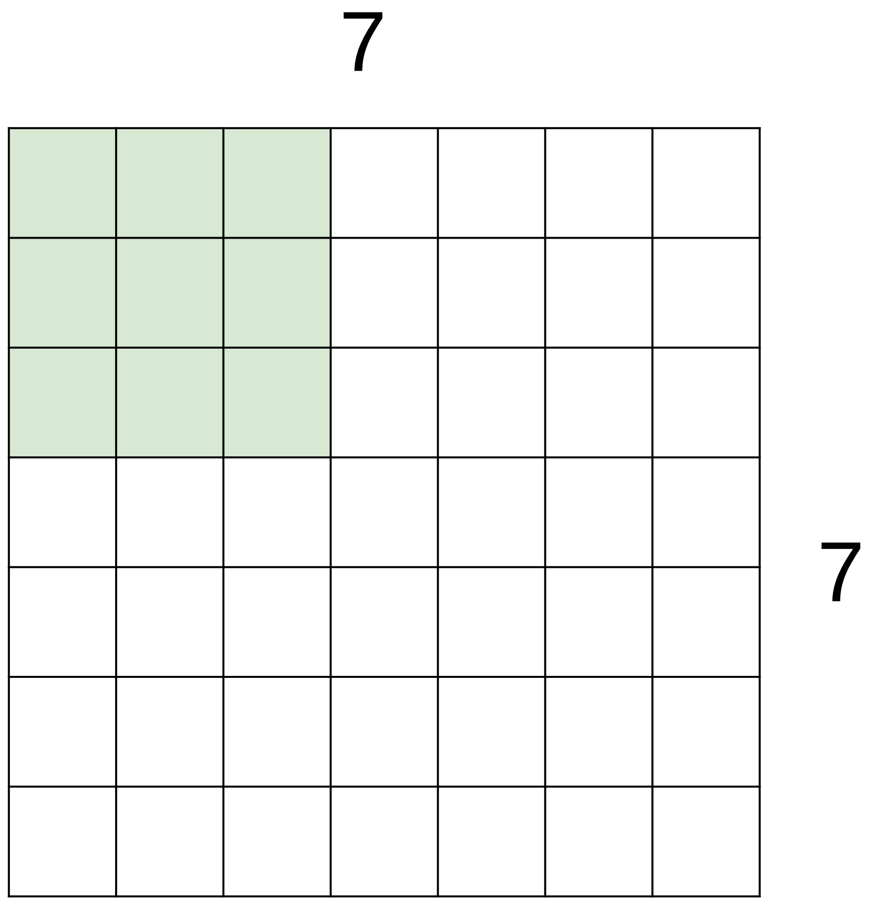




- A closer look at spatial dimensions:

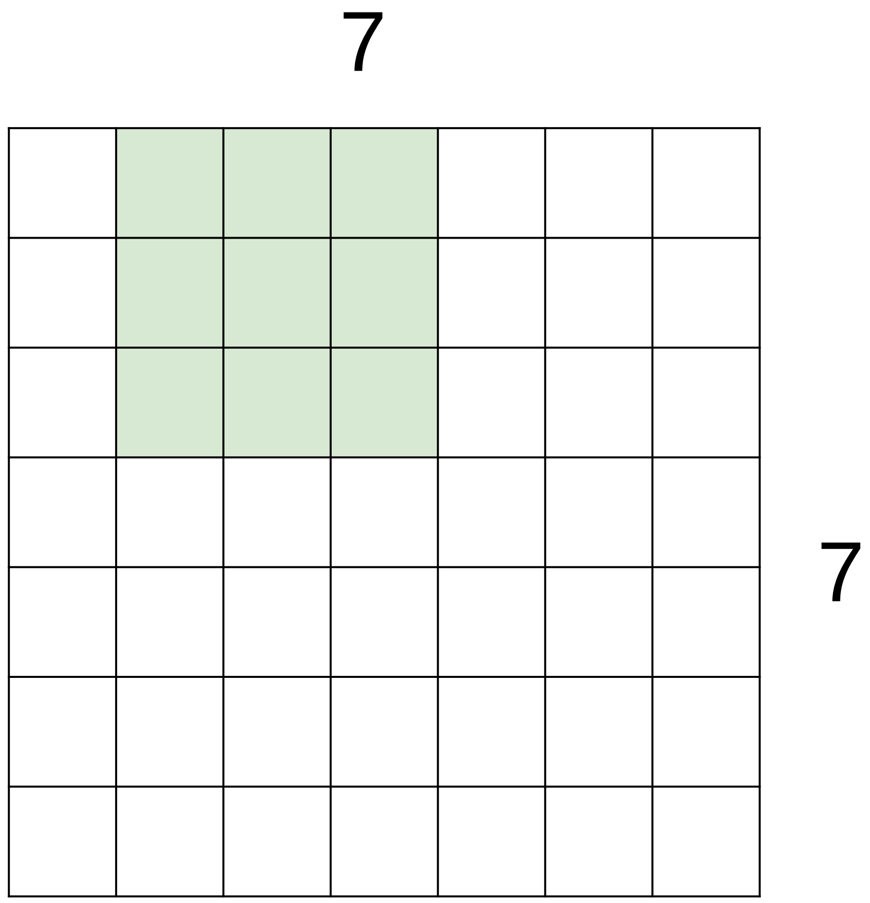


A CLOSER LOOK AT SPATIAL DIMENSIONS:



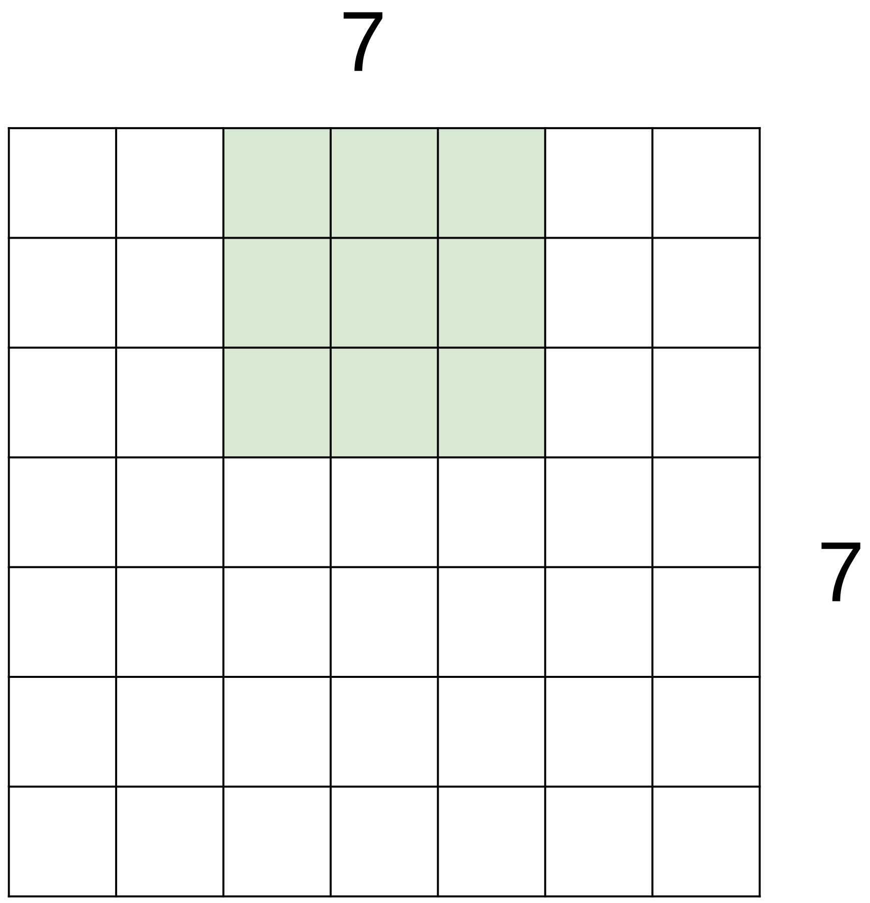
7x7 input (spatially)
assume 3x3 filter

A CLOSER LOOK AT SPATIAL DIMENSIONS:



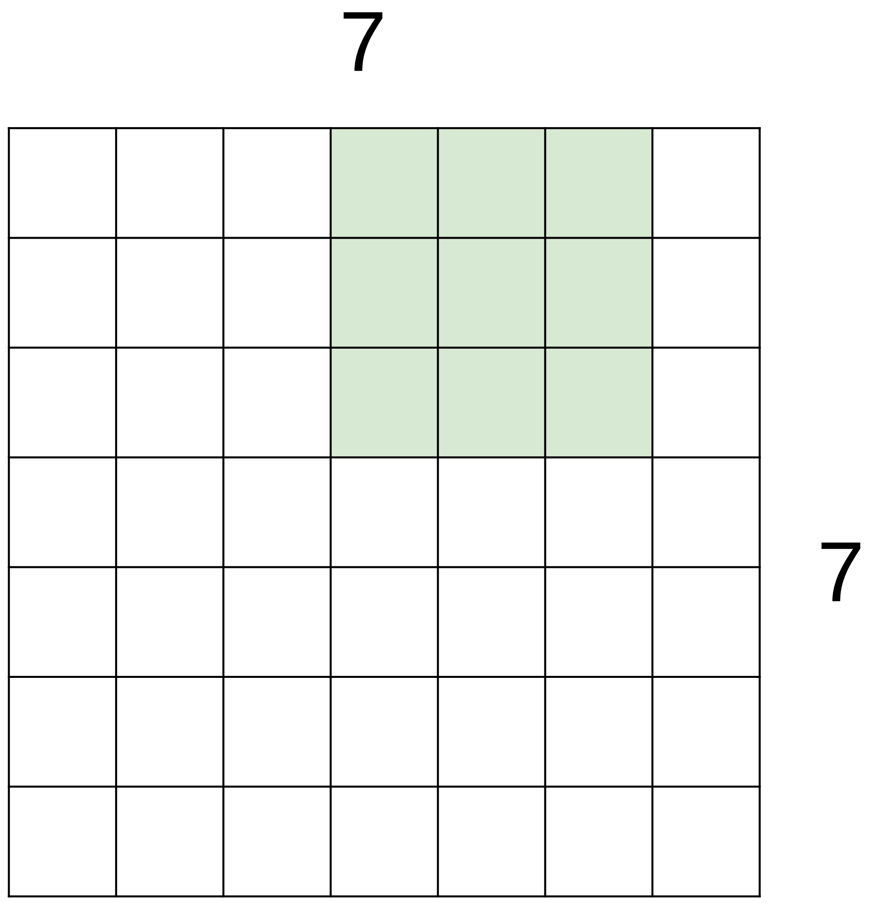
7x7 input (spatially)
assume 3x3 filter

A CLOSER LOOK AT SPATIAL DIMENSIONS:



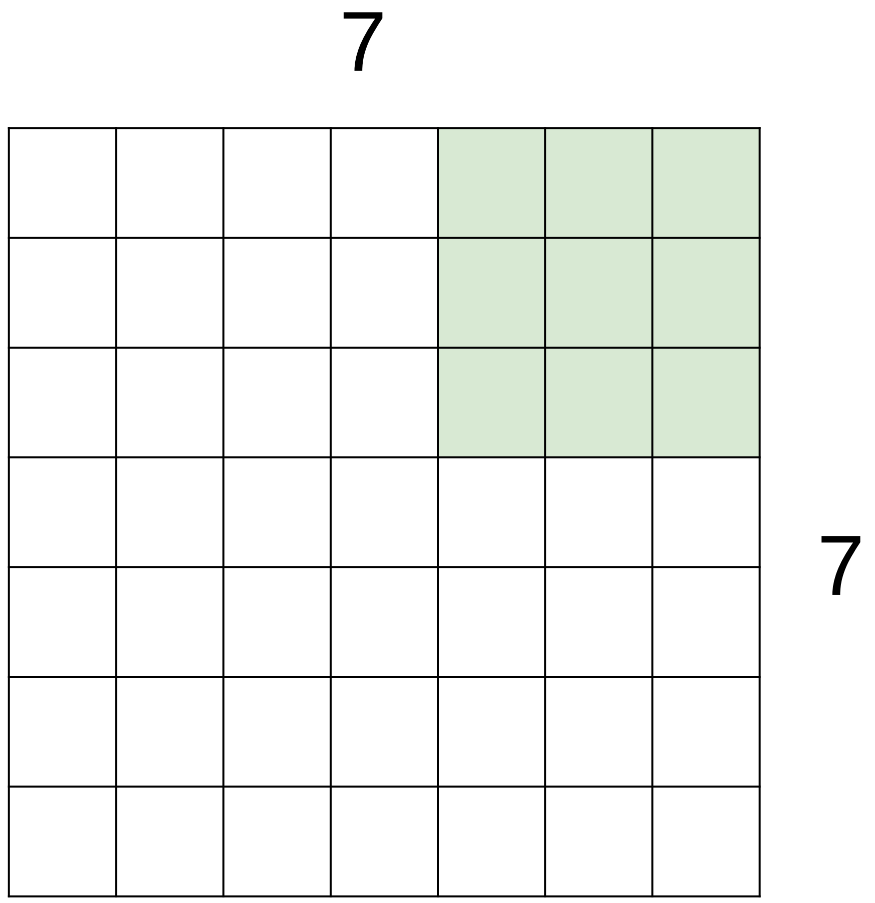
7x7 input (spatially)
assume 3x3 filter

A CLOSER LOOK AT SPATIAL DIMENSIONS:



7x7 input (spatially)
assume 3x3 filter

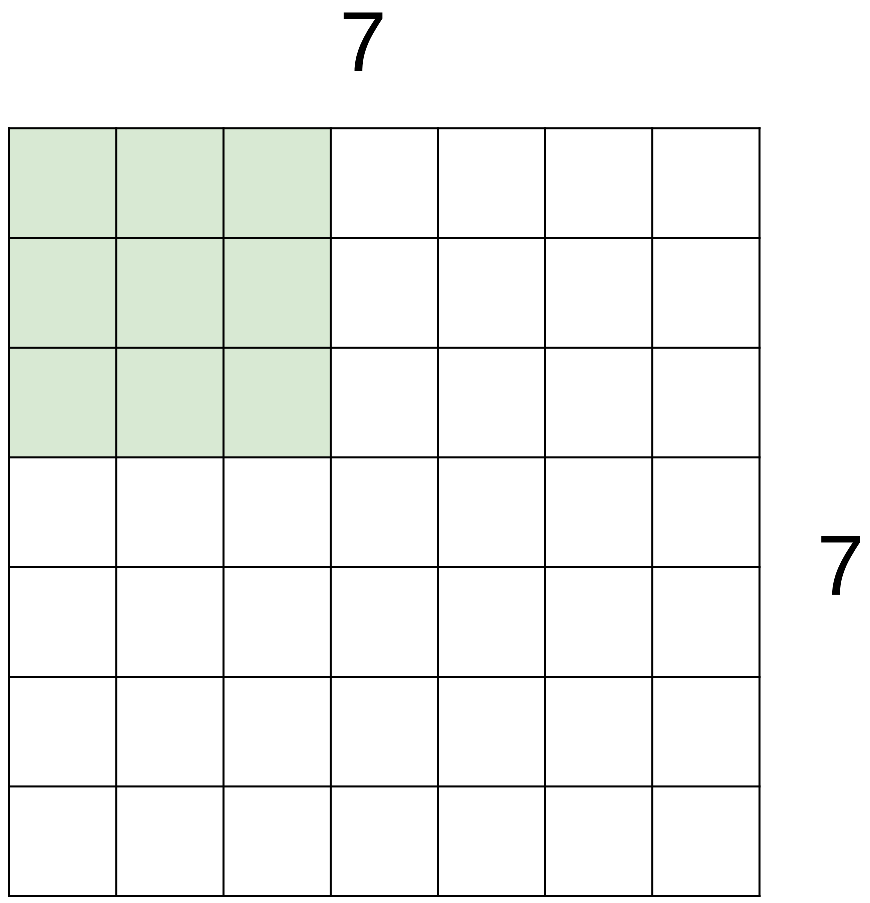
A CLOSER LOOK AT SPATIAL DIMENSIONS:



7x7 input (spatially)
assume 3x3 filter

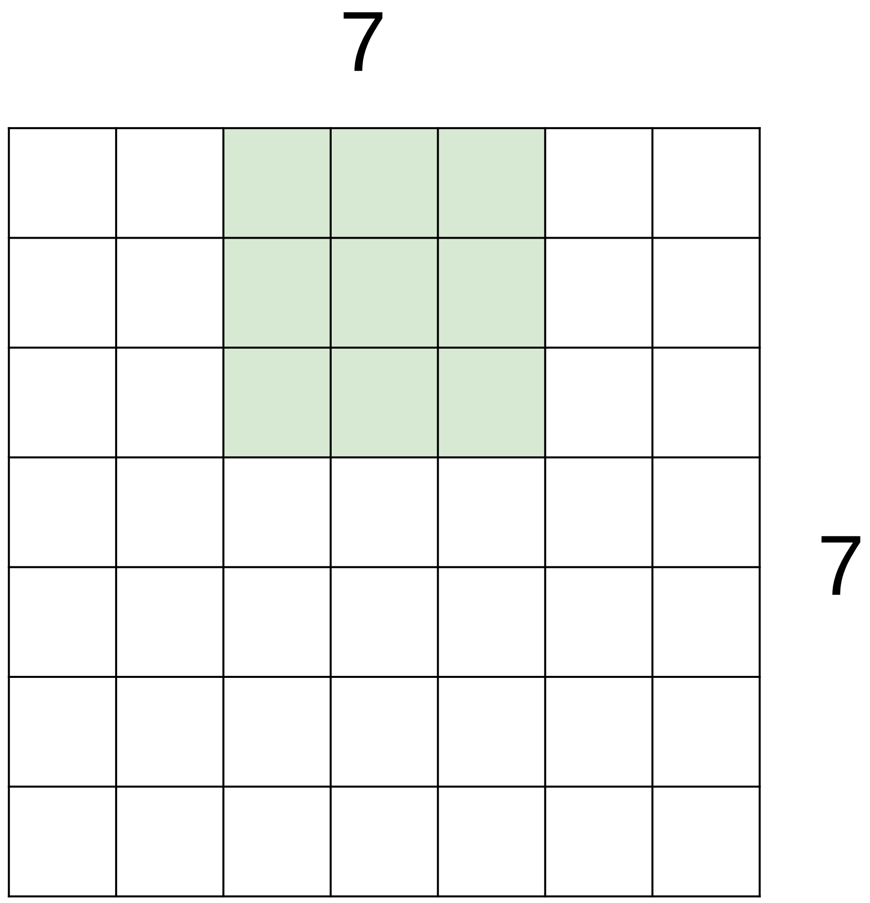
=> 5x5 output

A CLOSER LOOK AT SPATIAL DIMENSIONS:



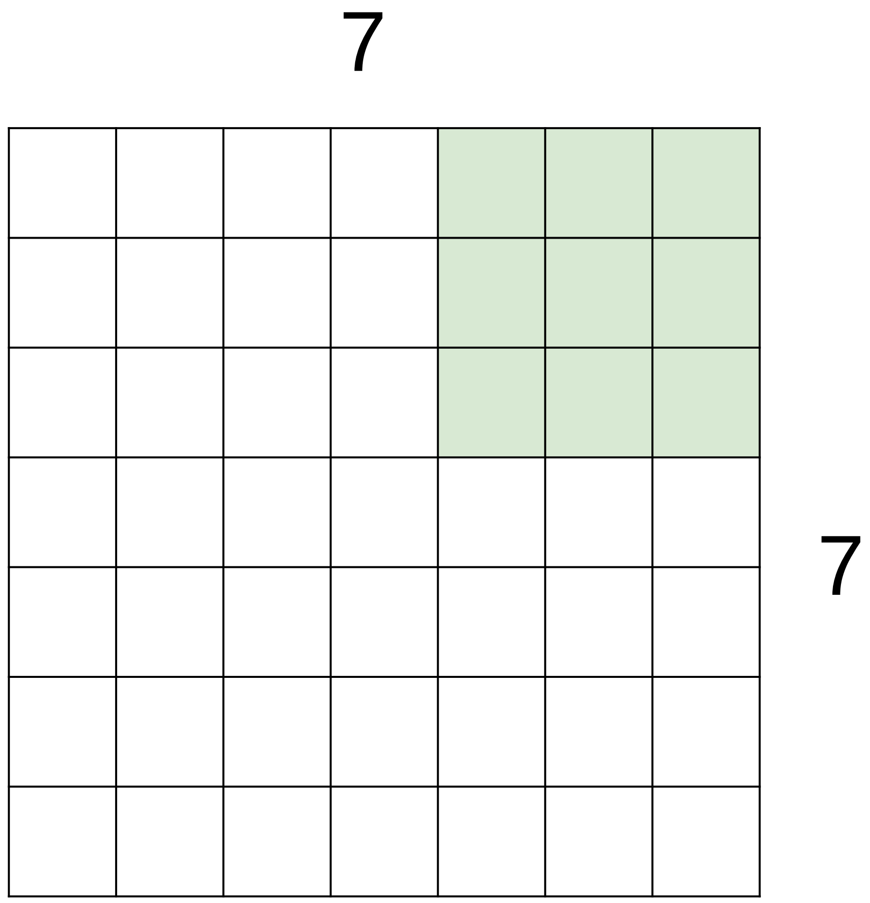
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A CLOSER LOOK AT SPATIAL DIMENSIONS:



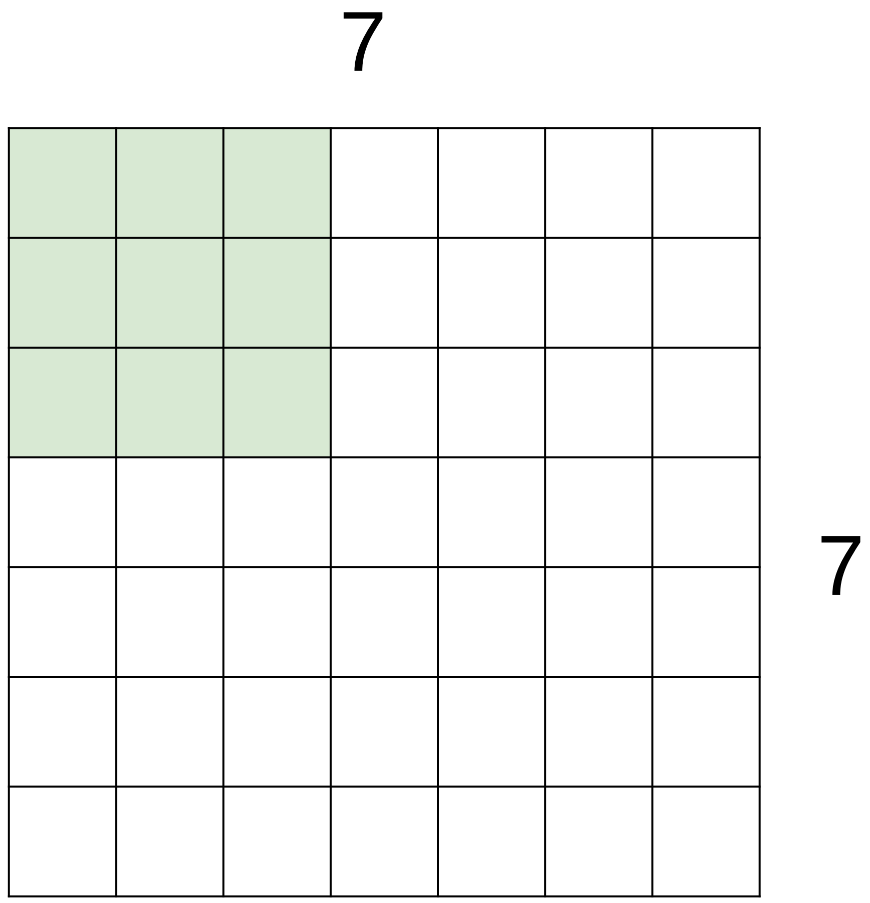
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A CLOSER LOOK AT SPATIAL DIMENSIONS:



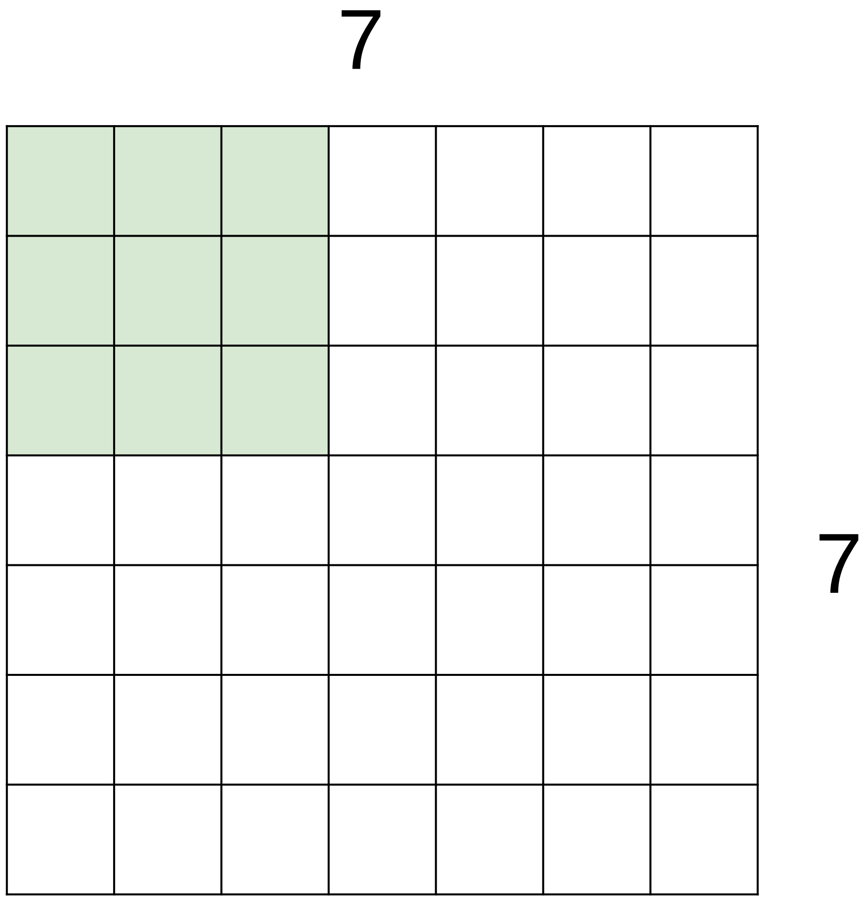
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A CLOSER LOOK AT SPATIAL DIMENSIONS:



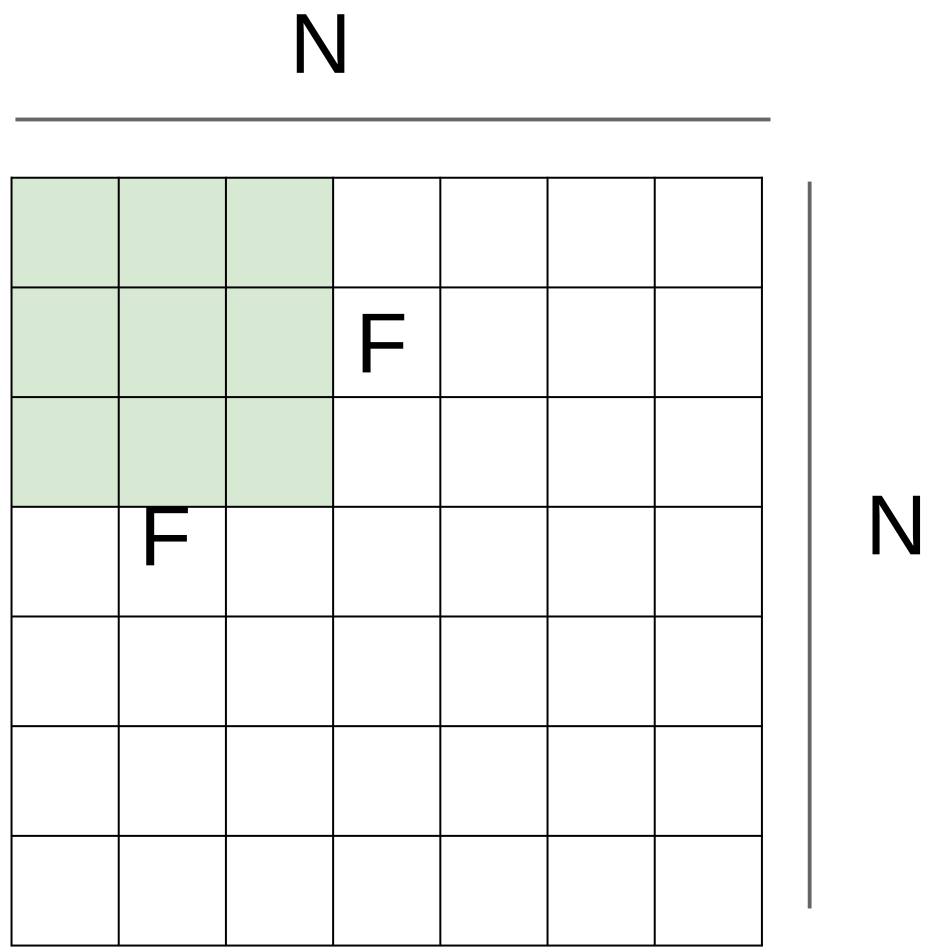
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A CLOSER LOOK AT SPATIAL DIMENSIONS:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

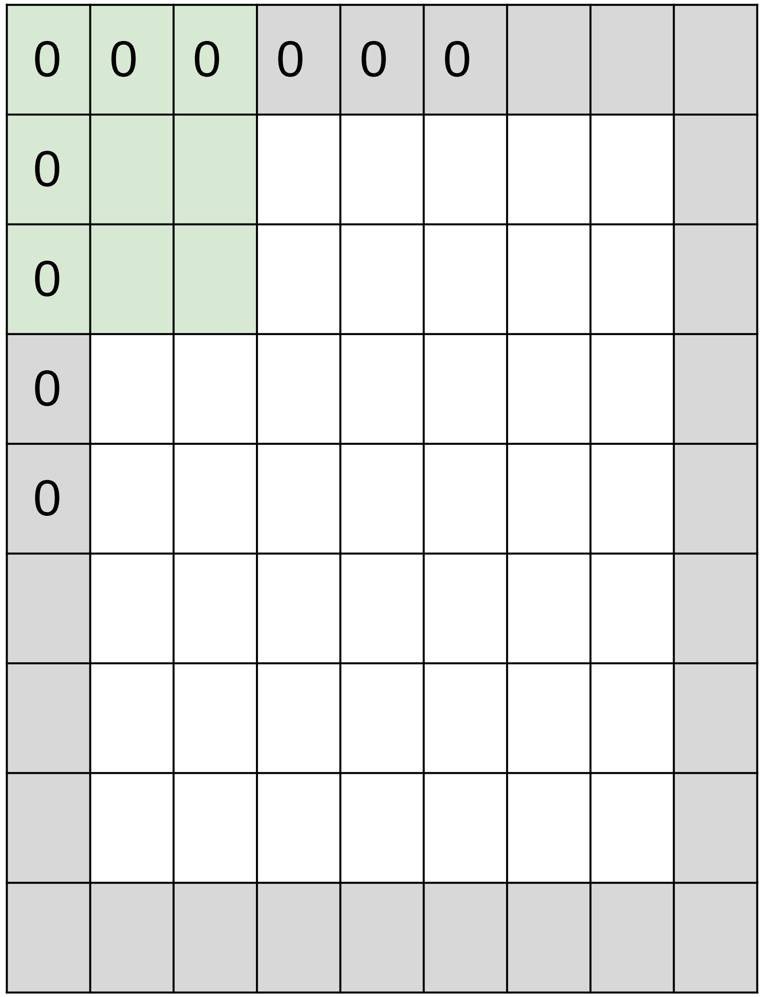
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

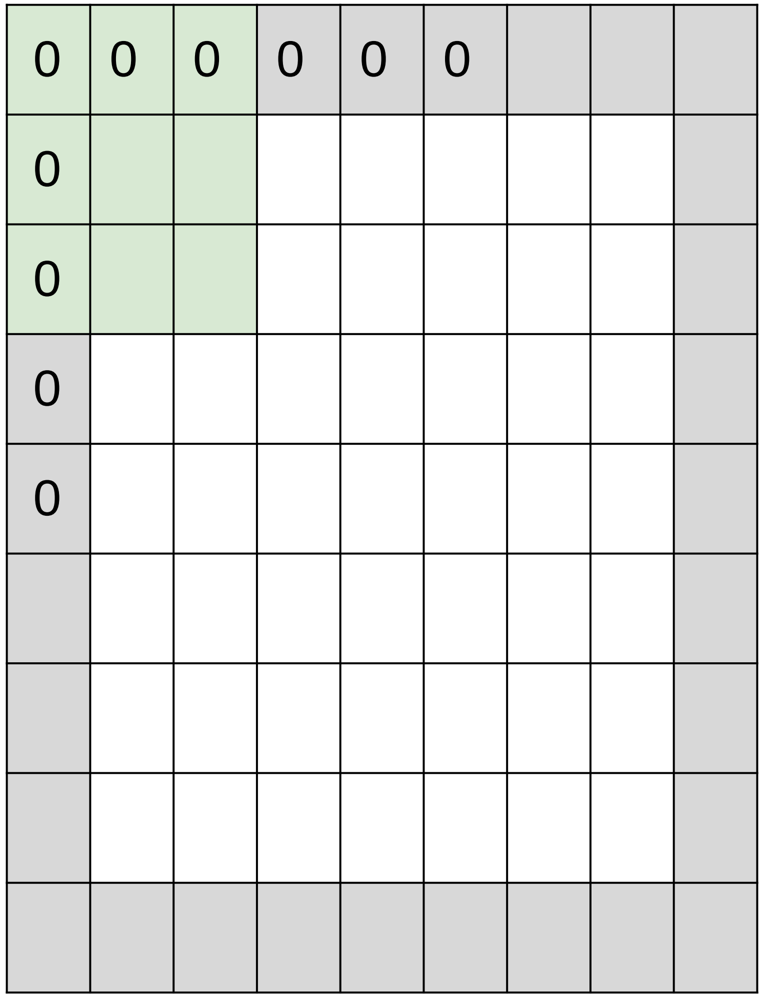
IN PRACTICE: COMMON TO ZERO PAD THE BORDER



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

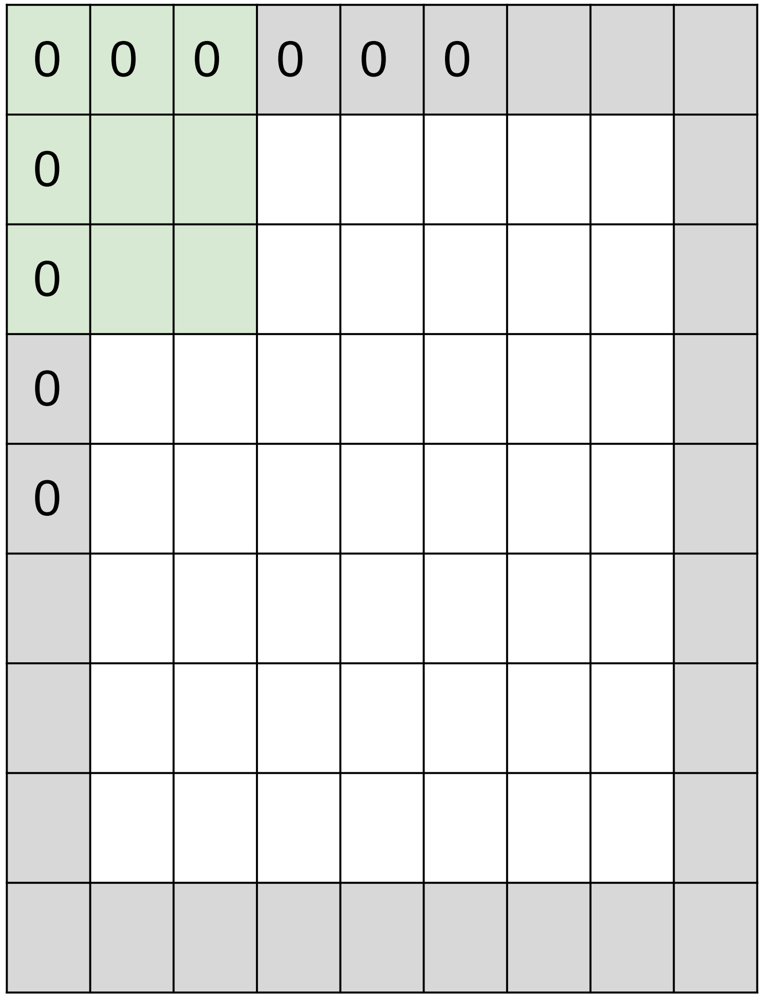
IN PRACTICE: COMMON TO ZERO PAD THE BORDER



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

IN PRACTICE: COMMON TO ZERO PAD THE BORDER

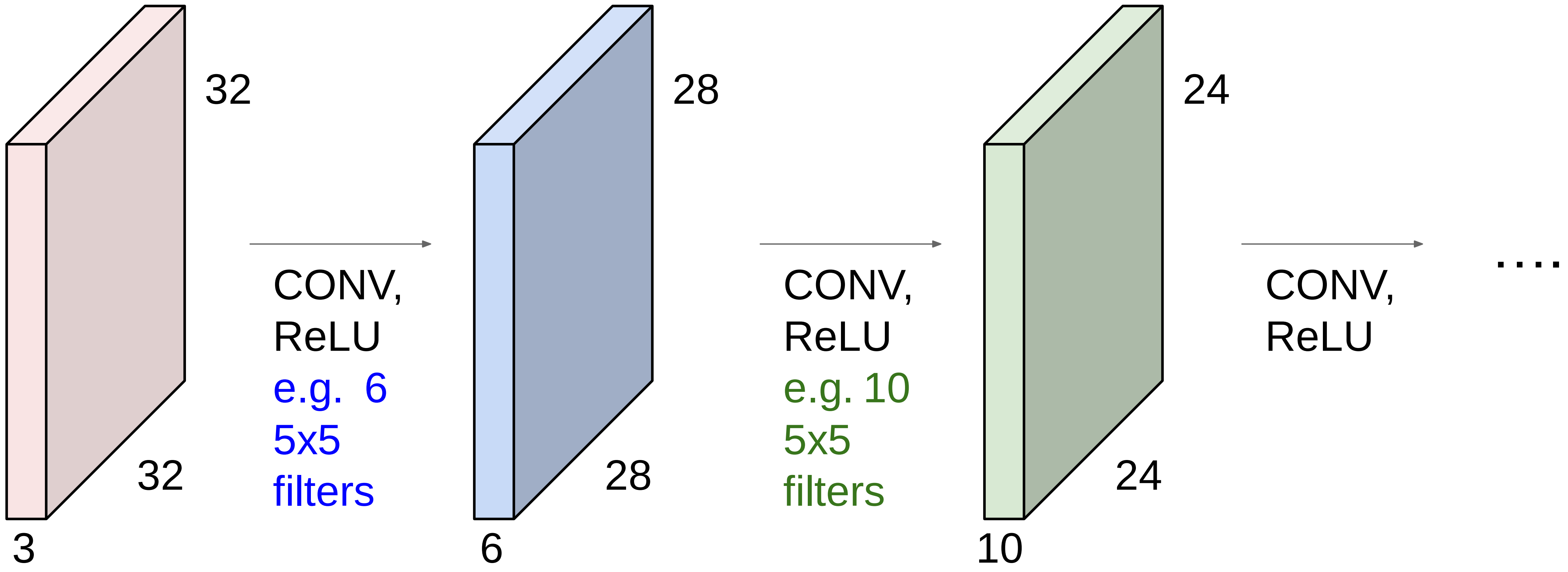


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

REMEMBER BACK TO…

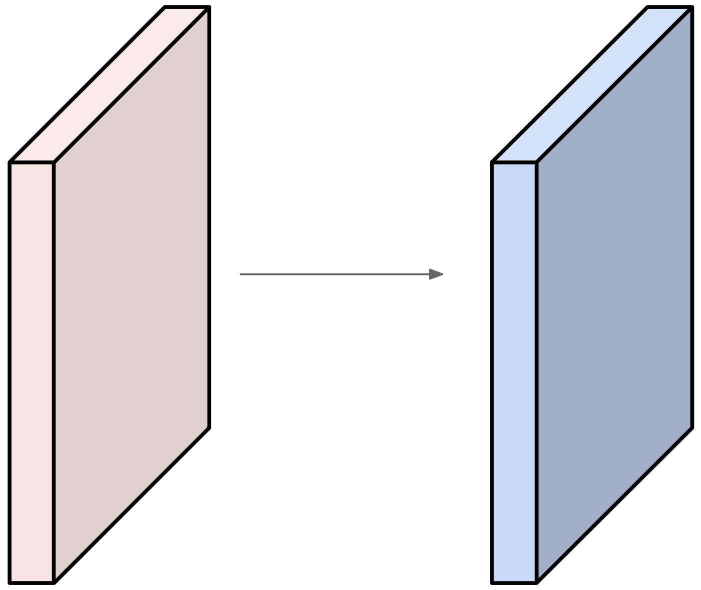
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



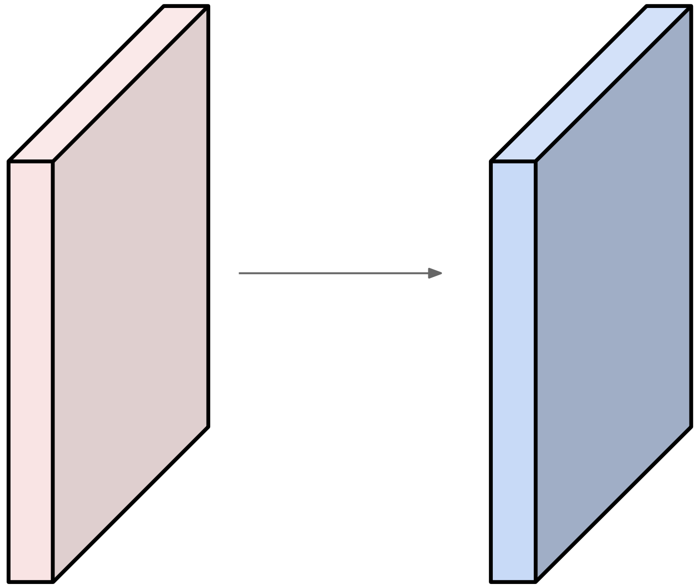
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Examples time:



Input volume: **32x32x3**

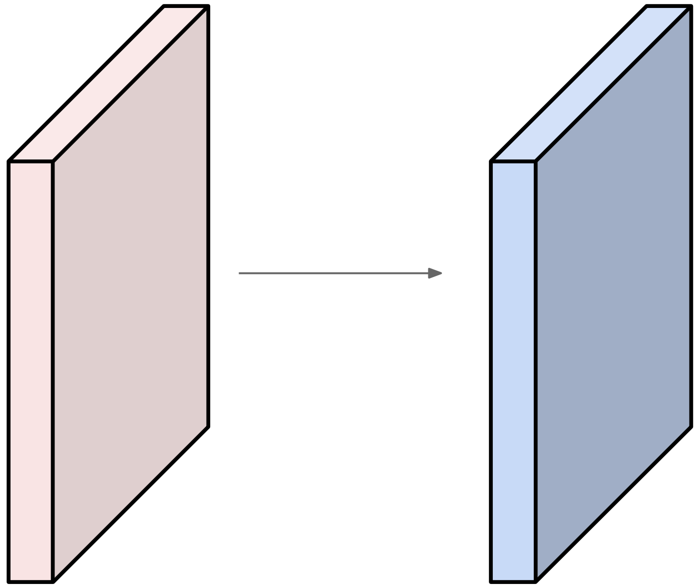
10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

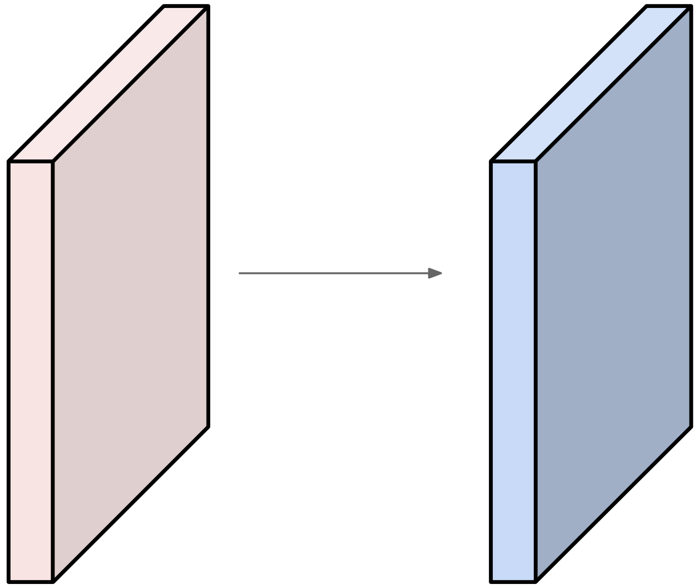
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

EXAMPLES TIME:

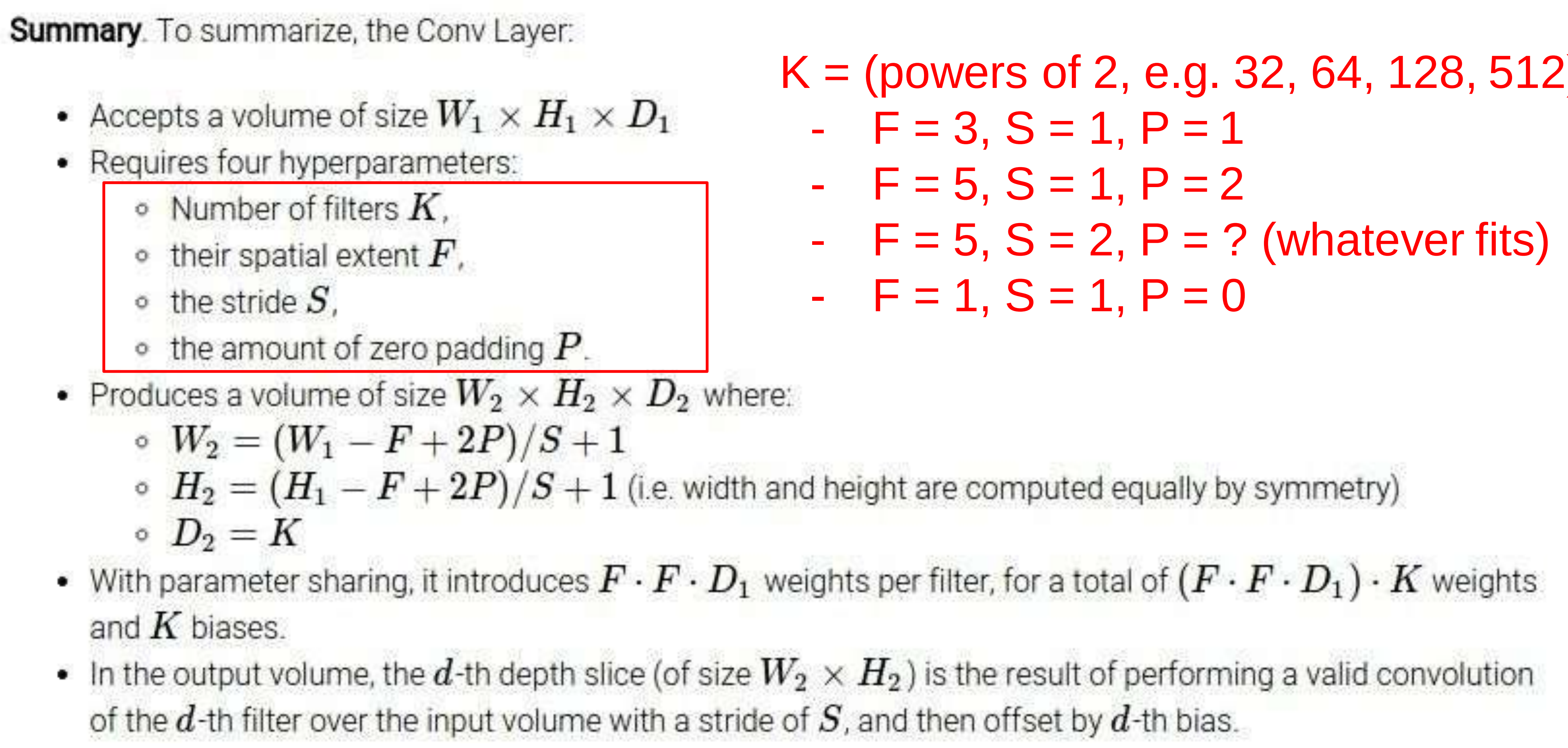


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5 + 1 = 26 params (+1 for bias)
=> 26*10 = **260**

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.



Summary. To summarize, the Conv Layer:

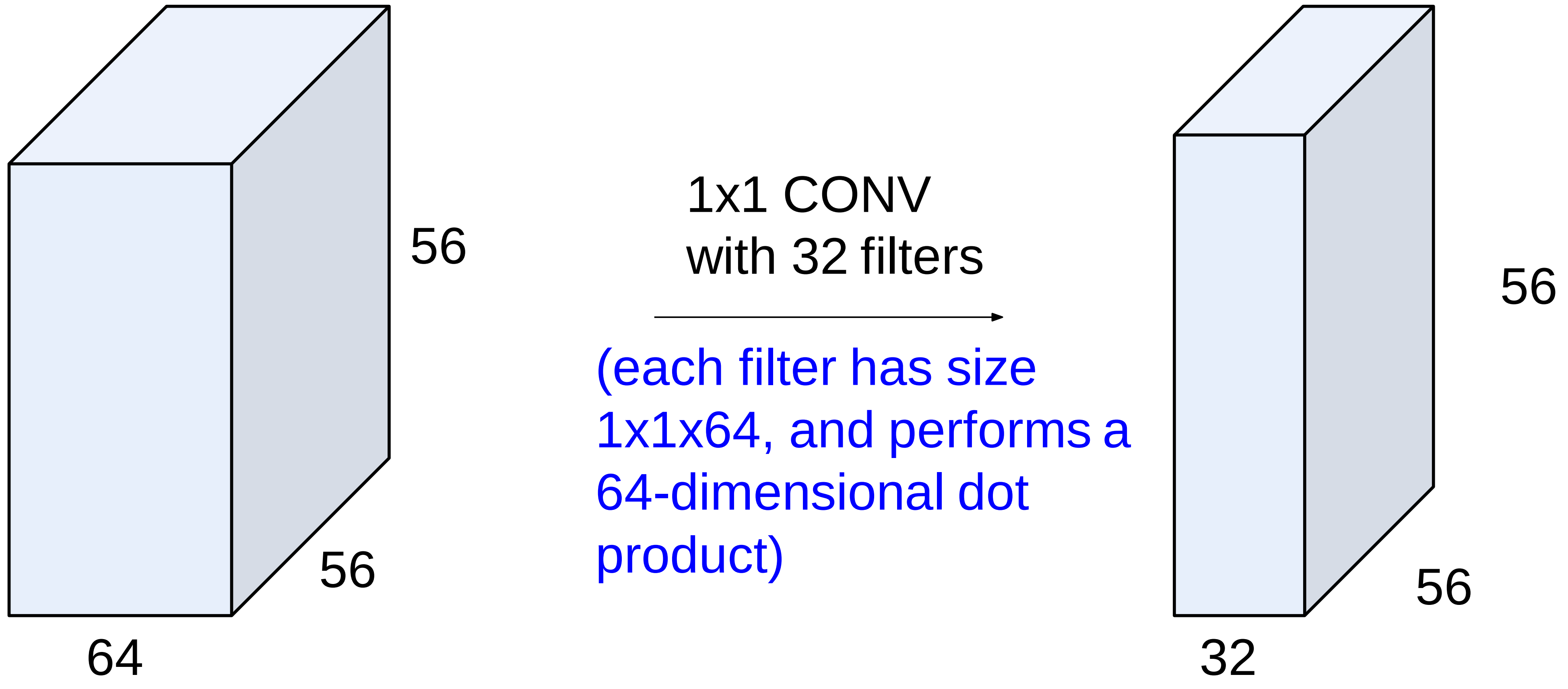
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

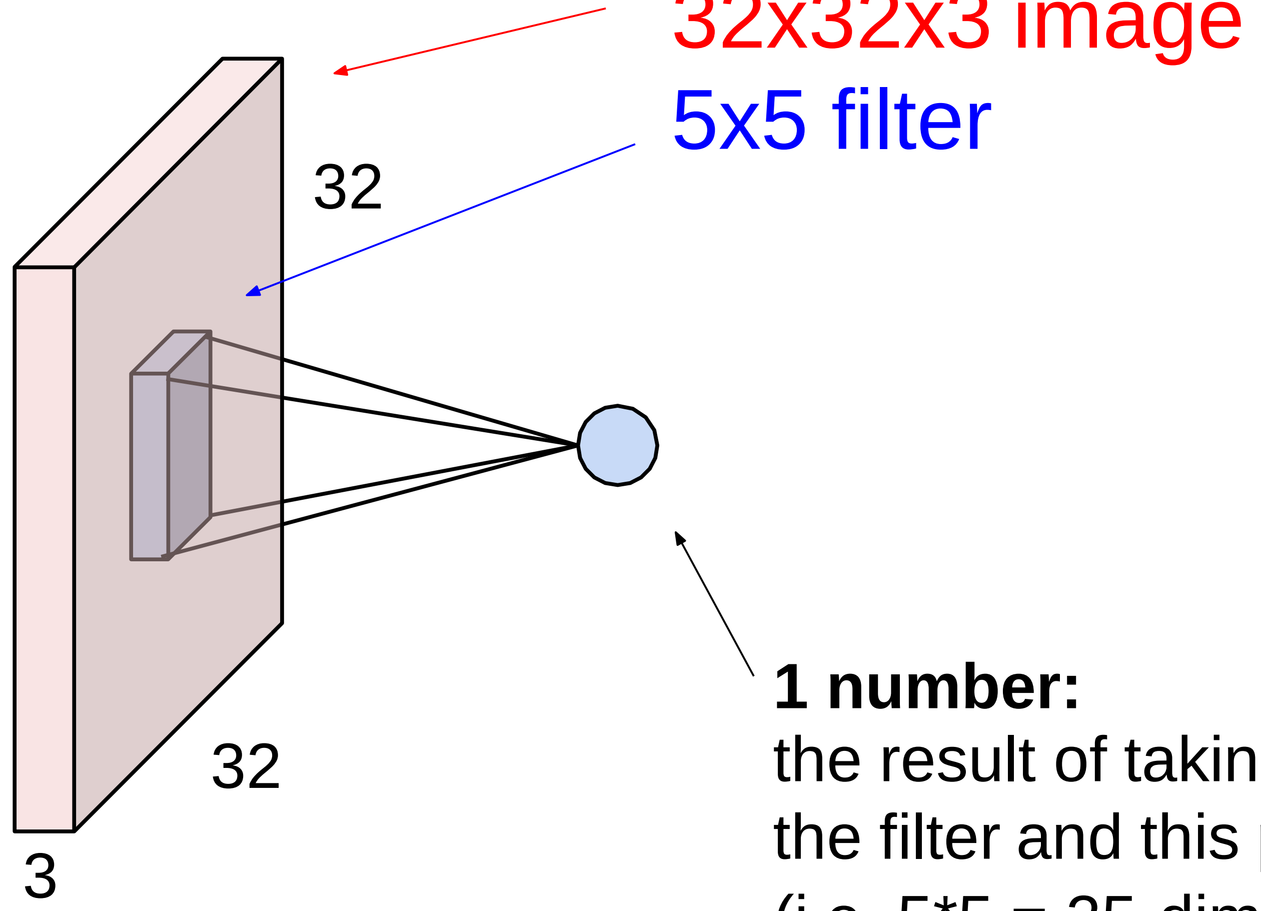
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

(BTW, 1X1 CONVOLUTION LAYERS MAKE PERFECT SENSE)



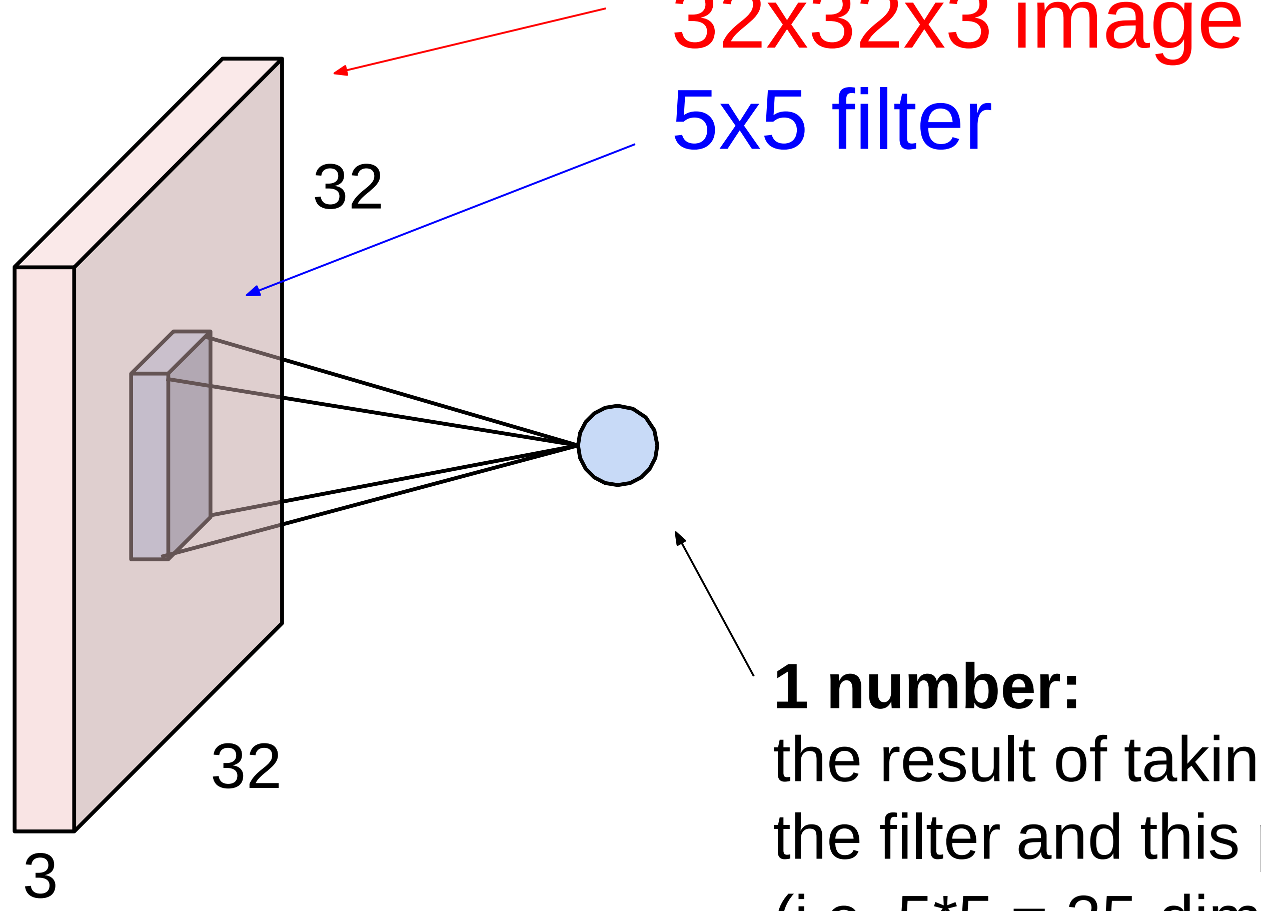
(each filter has size 1x1x64, and performs a 64-dimensional dot product)

THE BRAIN/NEURON VIEW OF CONV LAYER

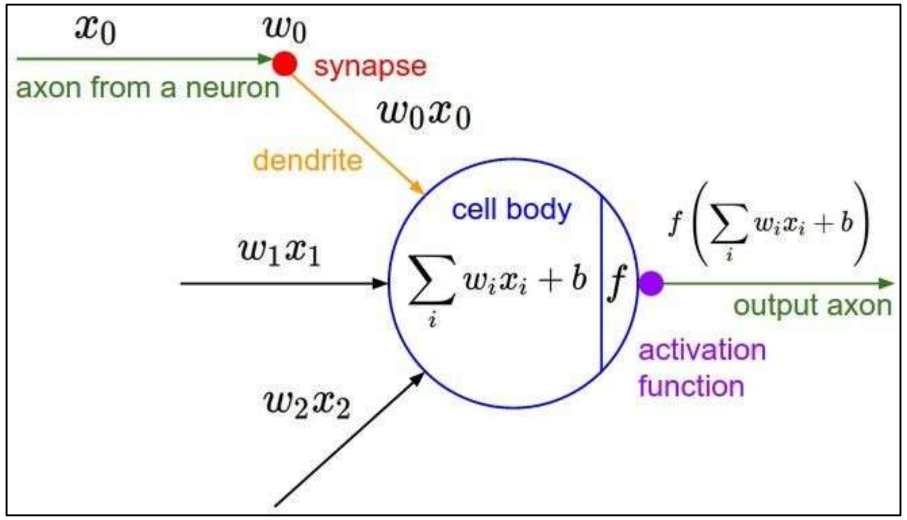


1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5 = 25-dimensional dot product)

THE BRAIN/NEURON VIEW OF CONV LAYER

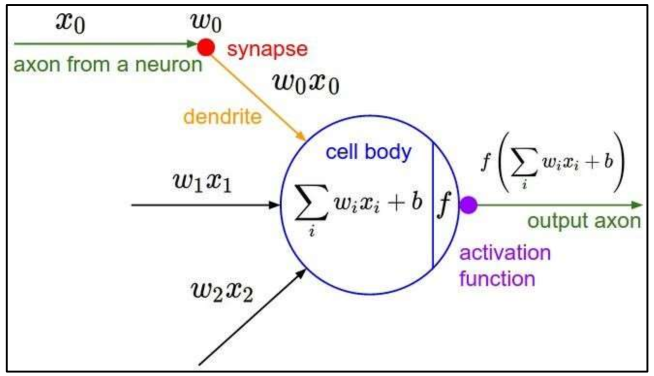


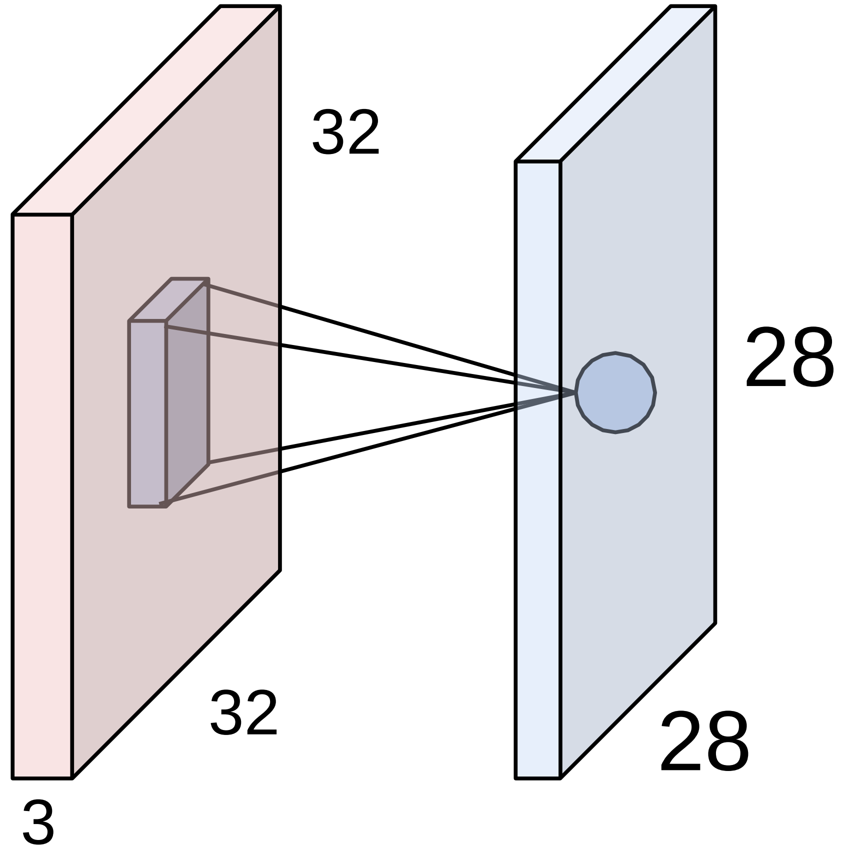
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5 = 25-dimensional dot product)



It’s just a neuron with local connectivity...

THE BRAIN/NEURON VIEW OF CONV LAYER



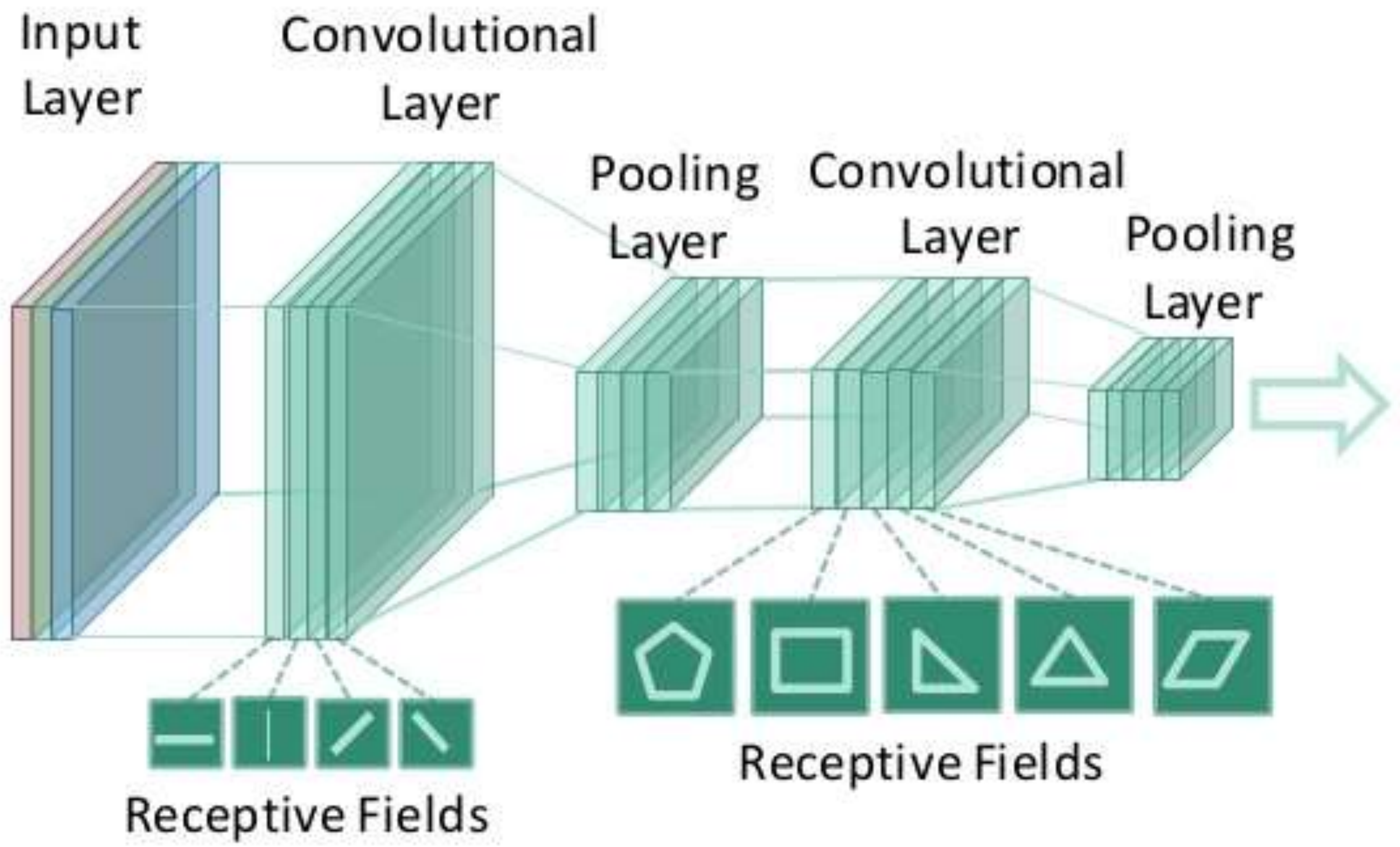


An activation map is a 28x28 sheet of neuron outputs:

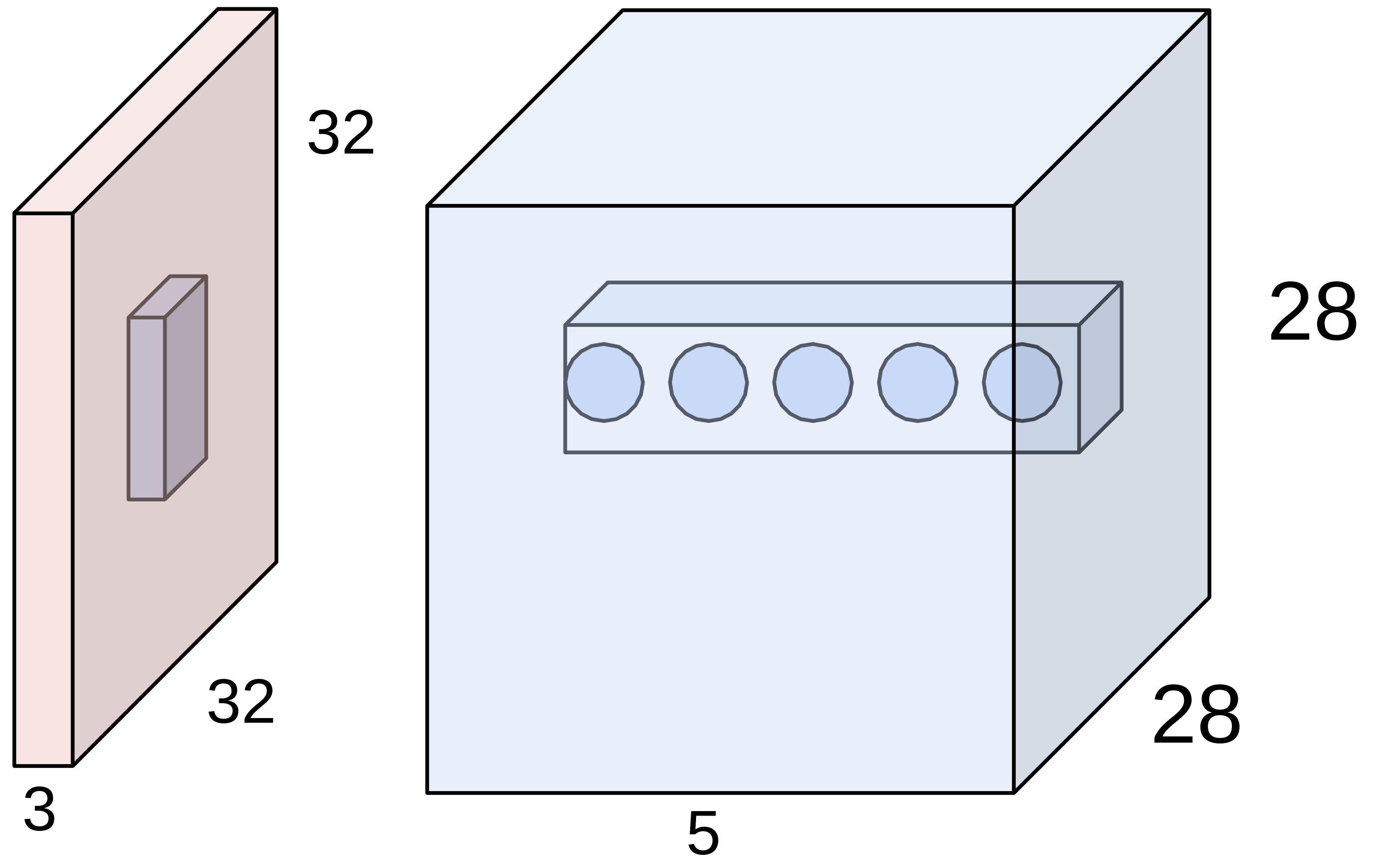
1. Each is connected to a small region in the input
2. All of them share parameters

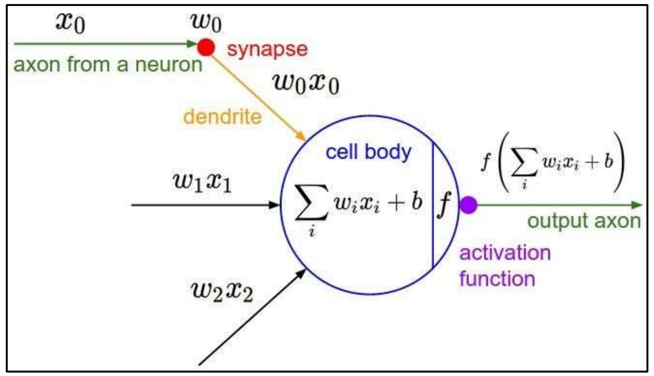
“5x5 filter” -> “5x5 receptive field for each neuron”

VISUAL PERCEPTION OF COMPUTER



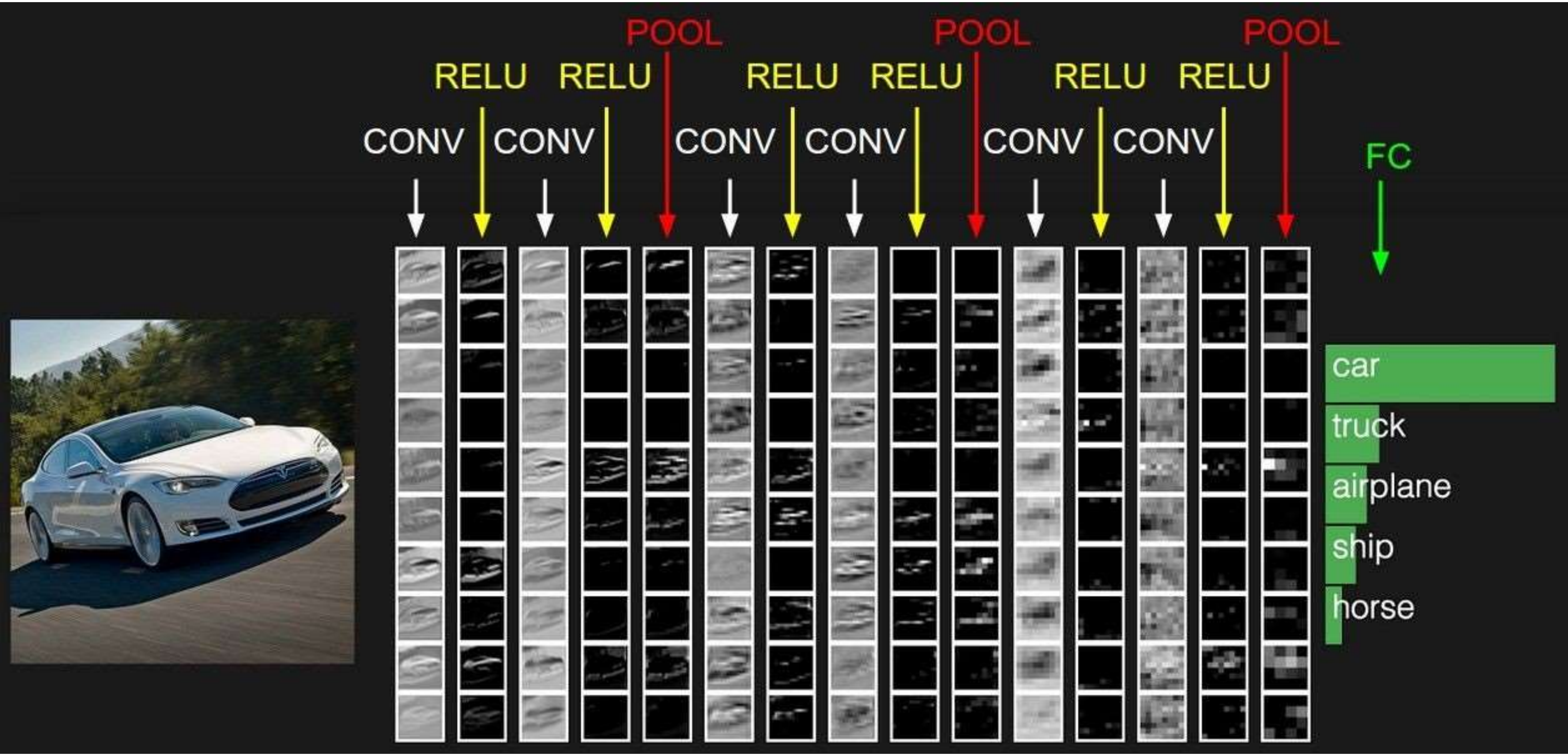
THE BRAIN/NEURON VIEW OF CONV LAYER





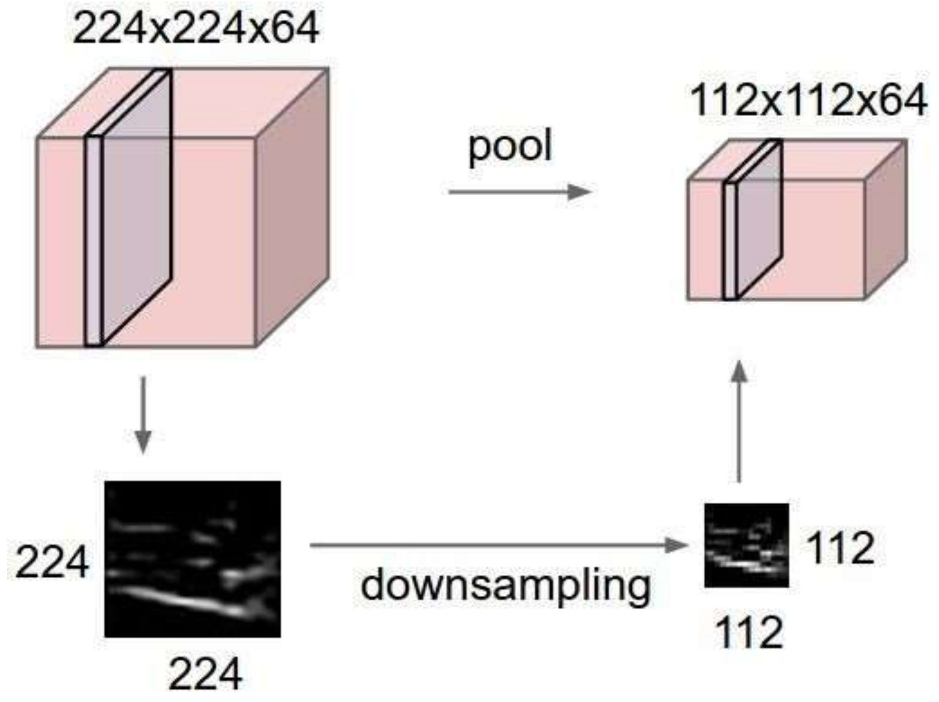
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume two more layers to go: POOL/FC

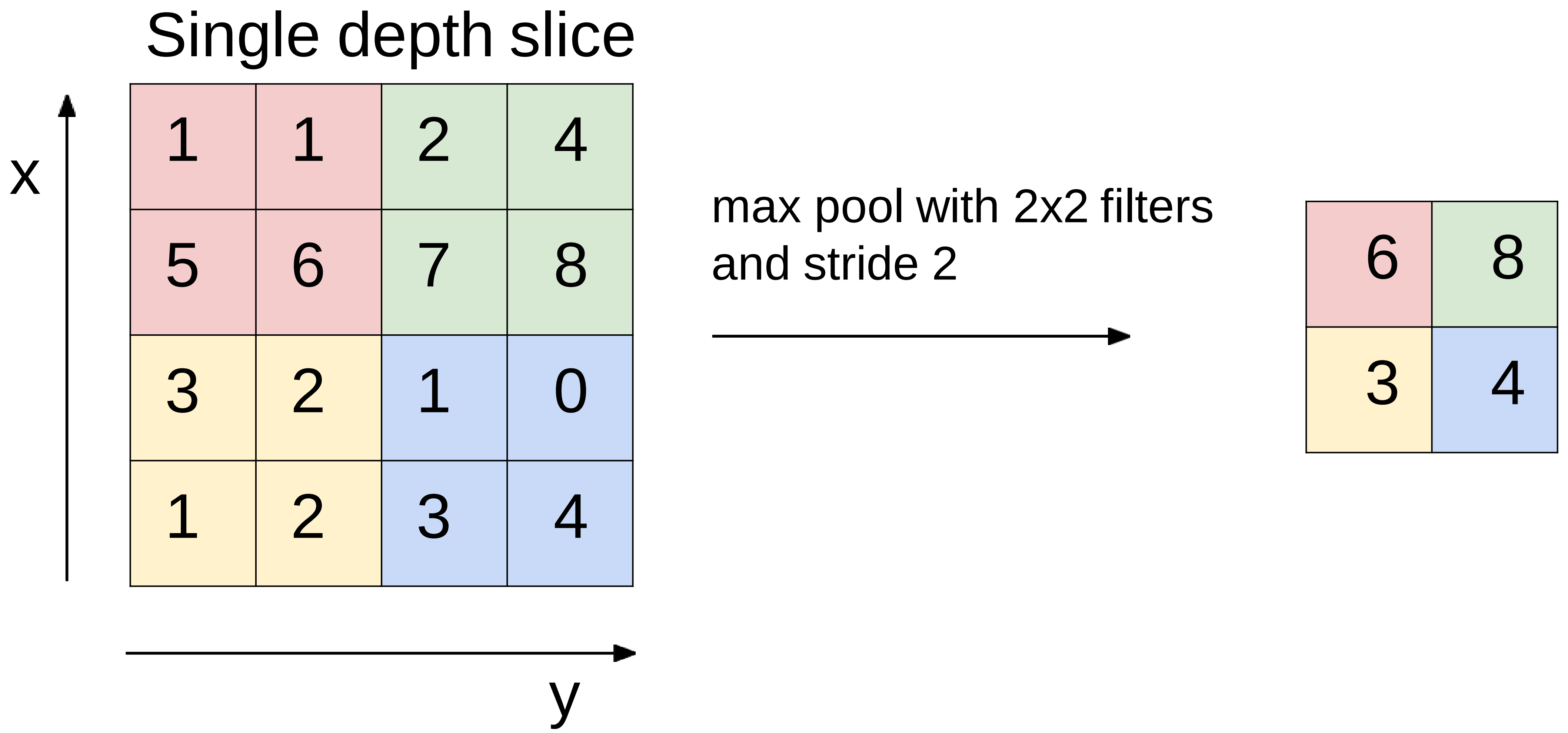


POOLING LAYER

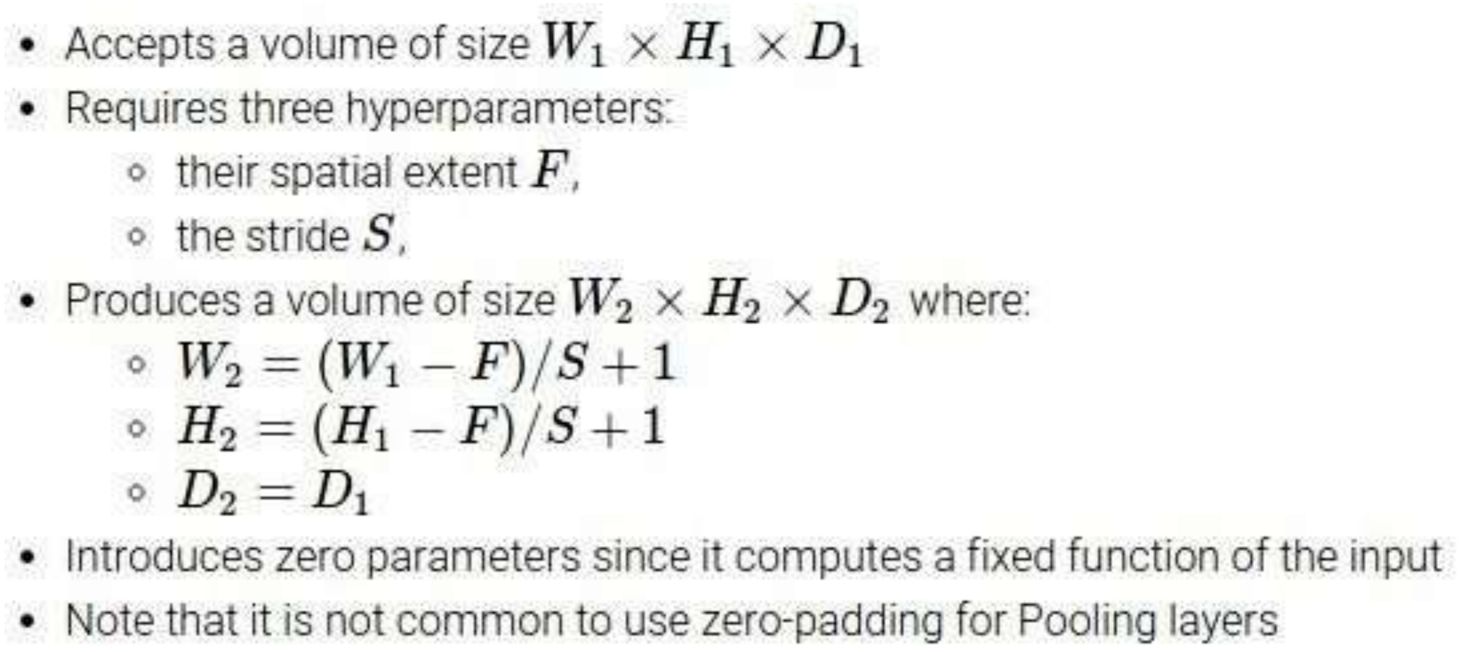
- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING


Single depth slice
x
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

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

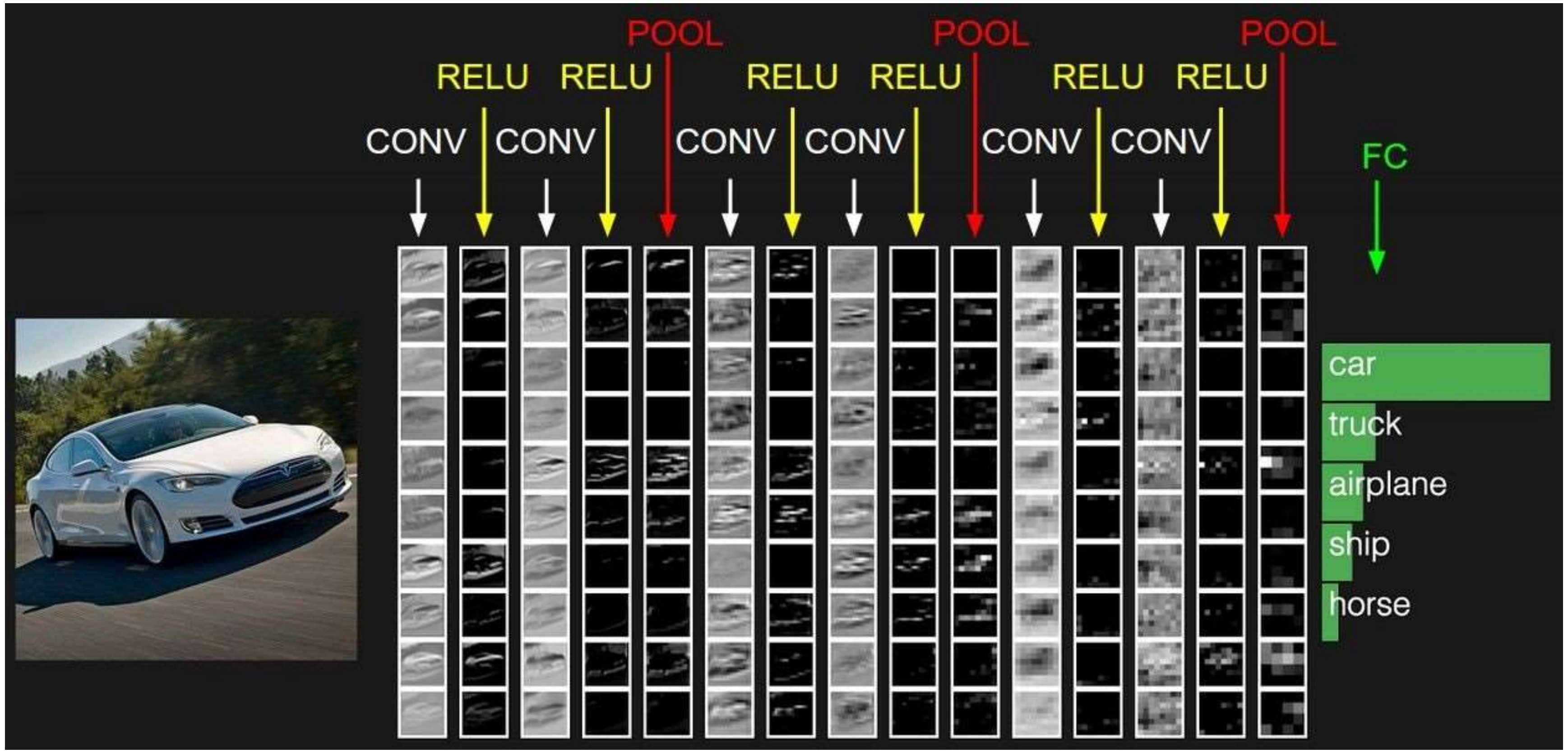
Common settings:

F = 2, S = 2

F = 3, S = 2

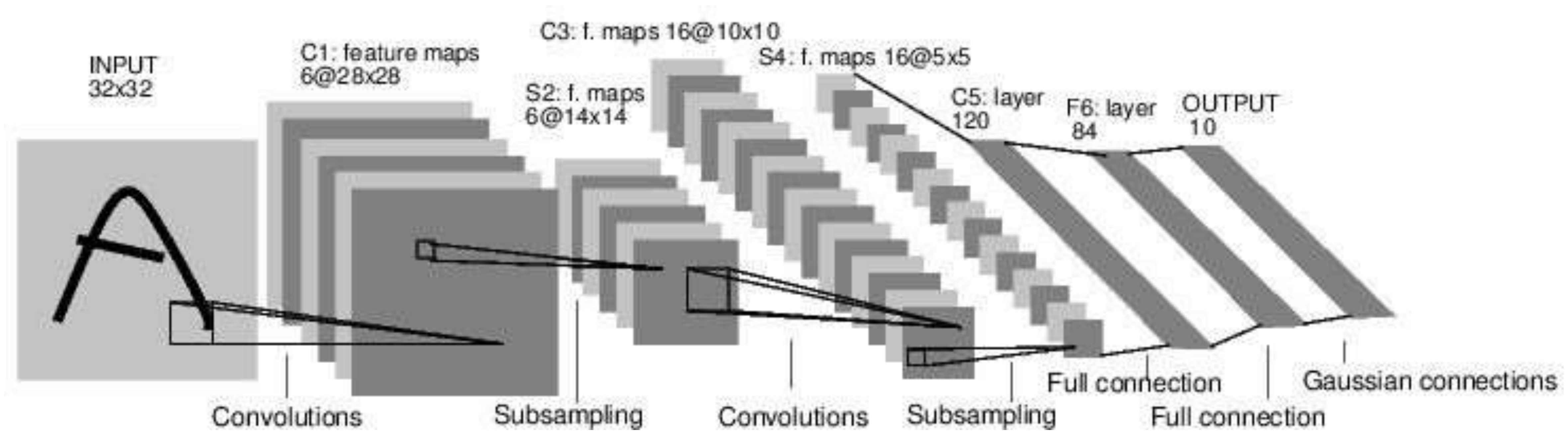
FULLY CONNECTED LAYER (FC LAYER)

- CONTAINS NEURONS THAT CONNECT TO THE ENTIRE INPUT VOLUME, AS IN ORDINARY NEURAL NETWORKS



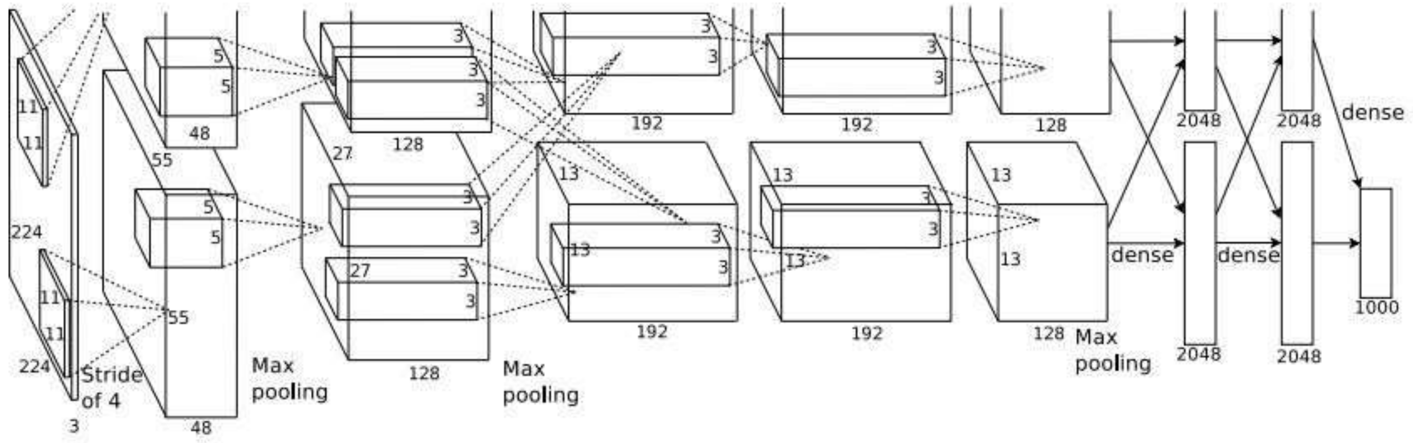
CASE STUDY: LENET-5

[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

CASE STUDY: ALEXNET

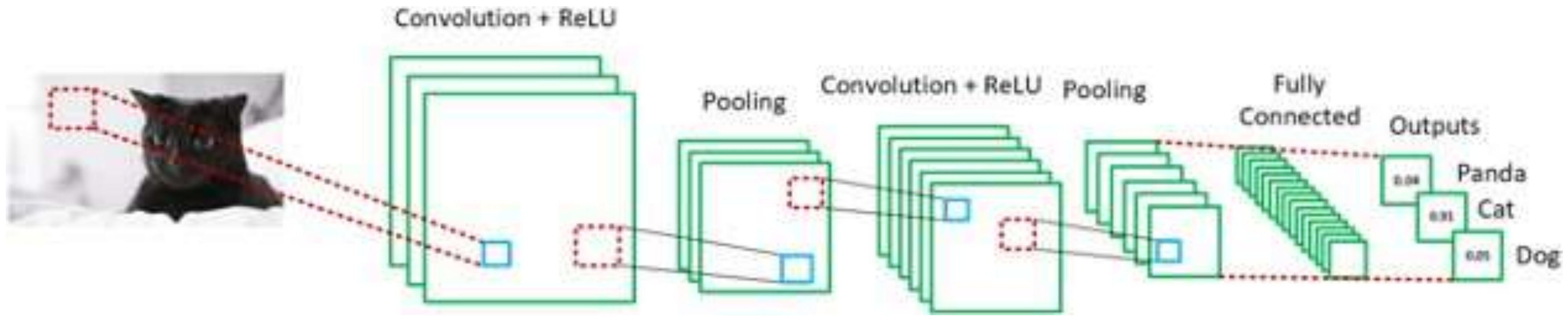


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

TRANSFER LEARNING - 2 MAJOR TYPES

Feature Extractor

Fine Tuning



TRANSFER LEARNING

- **Transfer learning** or inductive transfer is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.
- For example, knowledge gained while learning to recognize cars could apply when trying to recognize trucks. This area of research bears some relation to the long history of psychological literature on transfer of learning, although formal ties between the two fields are limited.

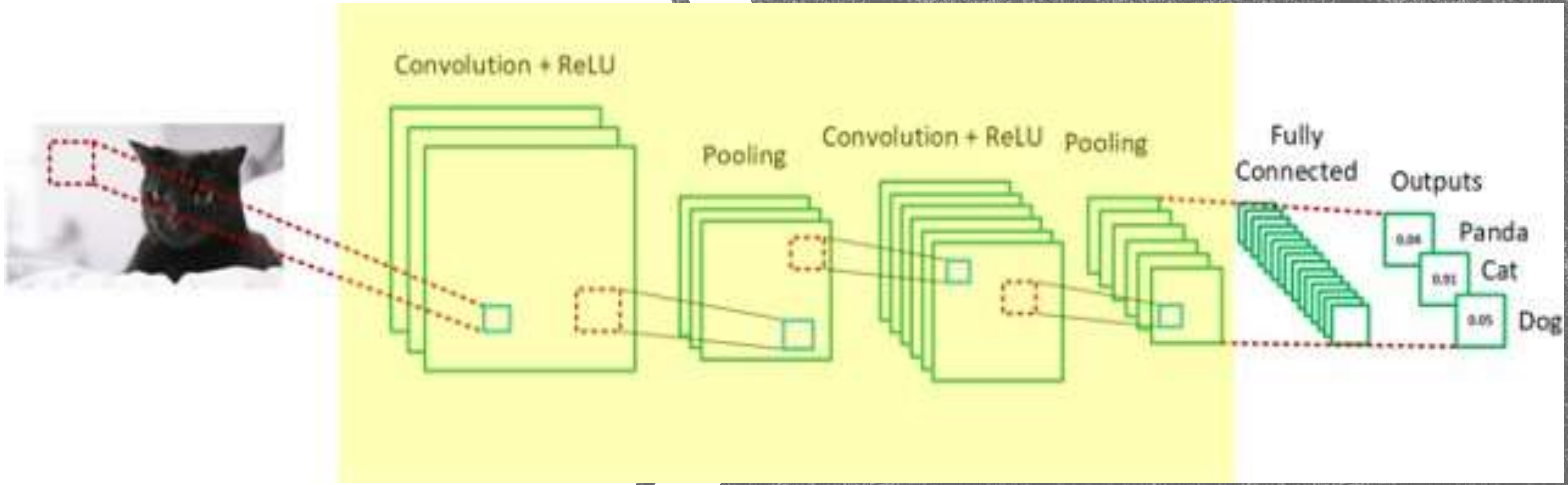
TRANSFER LEARNING

- The three major Transfer Learning scenarios look as follows:

1. ConvNet as fixed feature extractor.

- Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset.

1. CONVNET AS FIXED FEATURE EXTRACTOR.



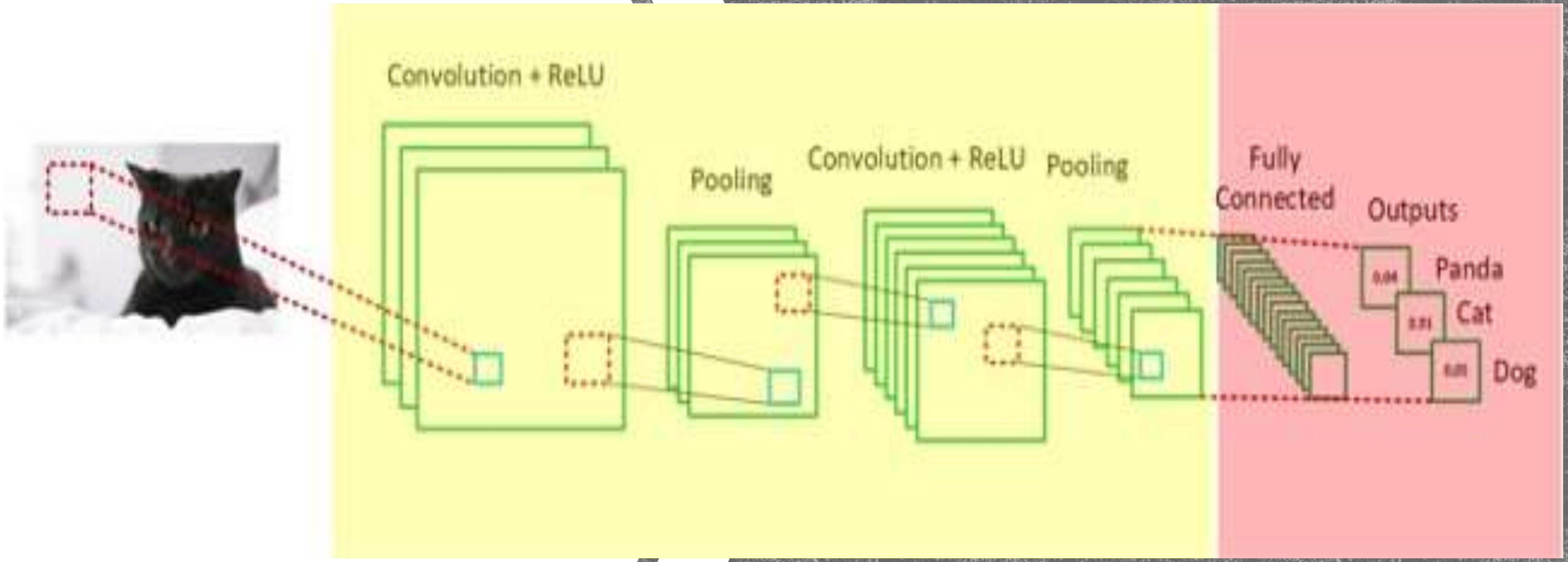
TRANSFER LEARNING

- The three major Transfer Learning scenarios look as follows:

2. Fine-tuning the ConvNet.

- The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network.

2. FINE-TUNING THE CONVNET

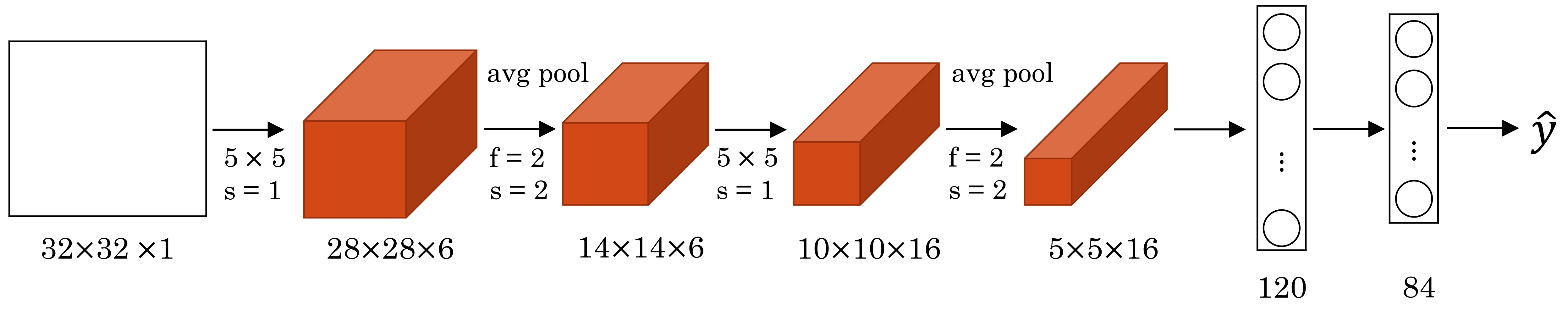


TRANSFER LEARNING - WHEN DO WE USE?

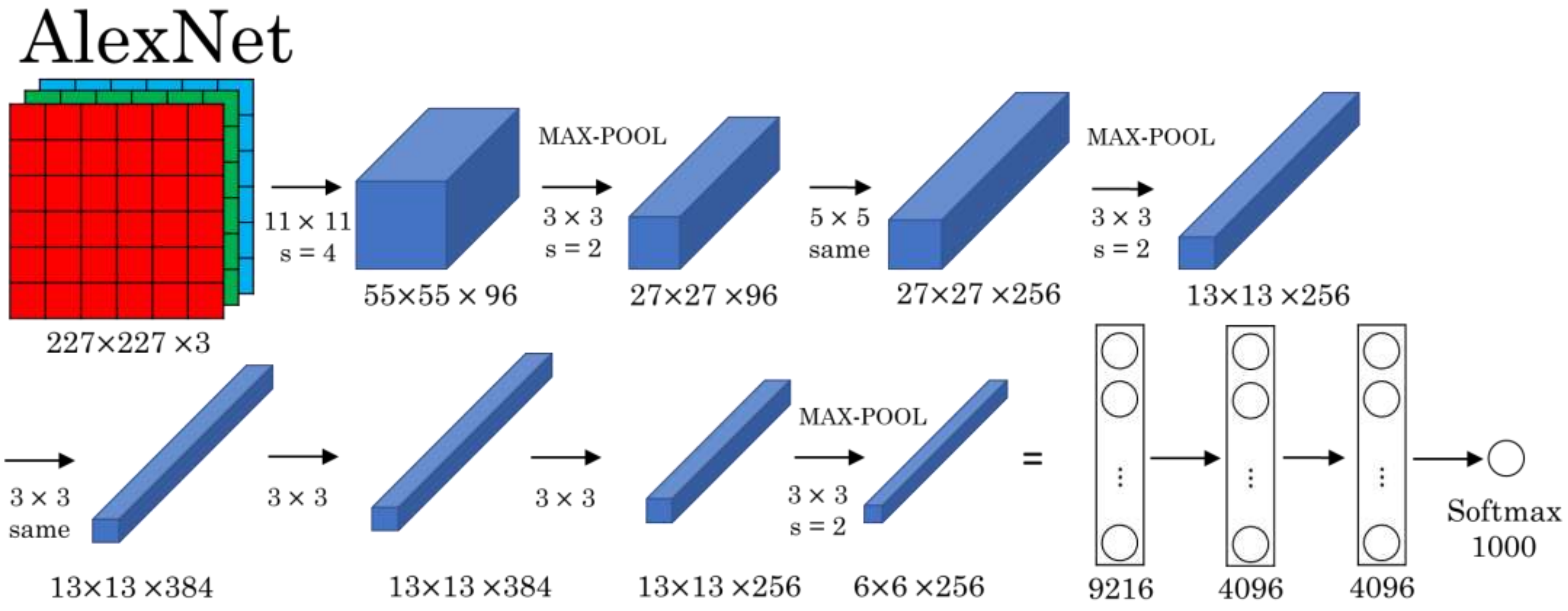
- Ideal scenario - New dataset is large and similar to the pre-trained original dataset. Models should not overfit.
- Not ideal but still recommended - New data is large but different.
- If data is small (even if similar) Transfer Learning and Fine Tuning can often overfit on the training data. A useful idea at times is to train a linear classier on the CNN outputs.

PLAIN NETWORKS

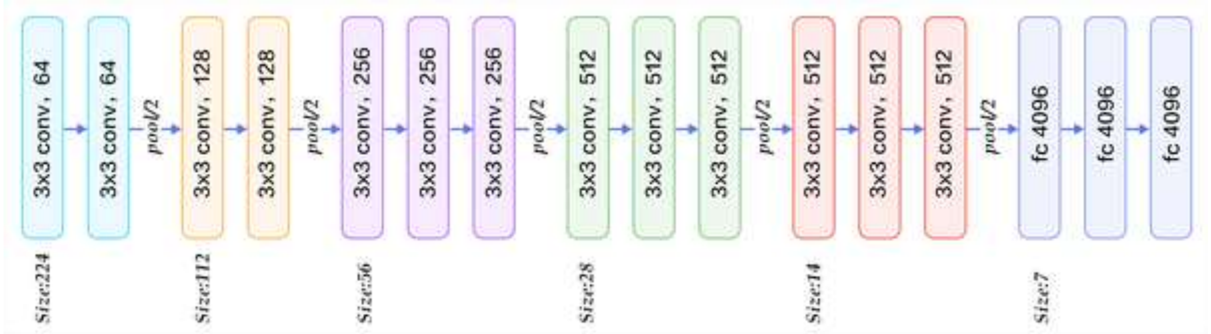
- Simple CNN architectures where connections among the layers follow simple straight-forward paths
- E.g. LeNet-5



PLAIN NETWORKS

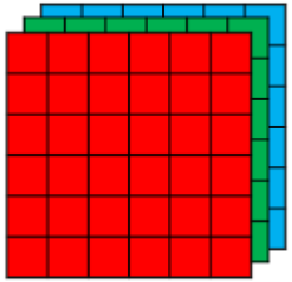


PLAIN NETWORKS



VGG - 16

CONV = 3×3 filter, s = 1, same   MAX-POOL = 2×2 , s = 2

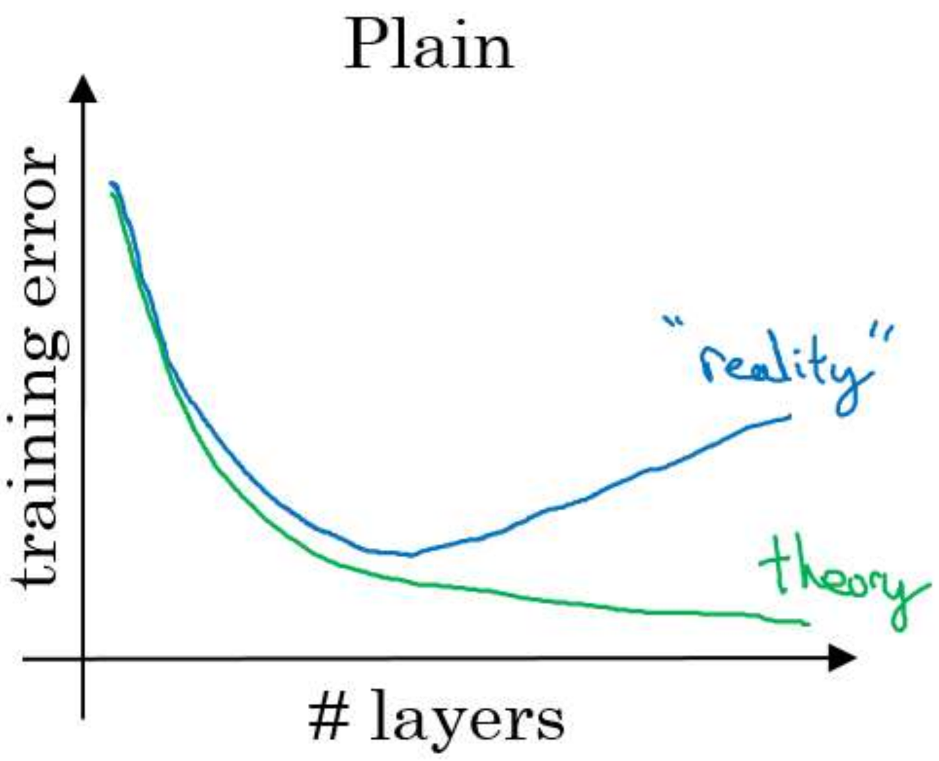


$$\begin{tikzcd}
224\times224\times3 \arrow[r, "\text{[CONV 64]}\ \times2"'] & 224\times224\times64 \arrow[r, "\text{POOL}"'] & 112\times112\times64 \arrow[r, "\text{[CONV 128]}\ \times2"'] & 112\times112\times128 \arrow[r, "\text{POOL}"'] & 56\times56\times128
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[r, "\text{[CONV 256]}\ \times3"'] & 56\times56\times256 \arrow[r, "\text{POOL}"'] & 28\times28\times256 \arrow[r, "\text{[CONV 512]}\ \times3"'] & 28\times28\times512 \arrow[r, "\text{POOL}"'] & 14\times14\times512
\end{tikzcd}$$

$$\begin{tikzcd}
{} \arrow[r, "\text{[CONV 512]}\ \times3"'] & 14\times14\times512 \arrow[r, "\text{POOL}"'] & 7\times7\times512 \arrow[r] & \text{FC } 4096 \arrow[r] & \text{FC } 4096 \arrow[r] & \text{Softmax } 1000
\end{tikzcd}$$

PROBLEMS WITH PLAIN NETWORKS



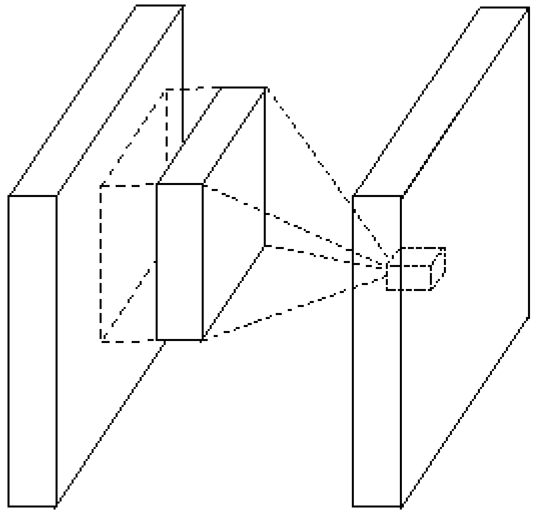
- Increasing number of layers increases computational complexity
- More layers does not always guarantee better performance

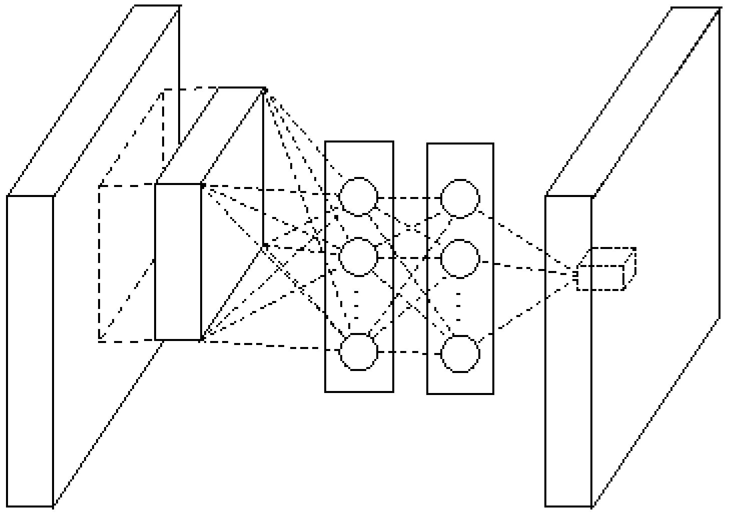
CNN MICROARCHITECTURES

- Modern CNNs are made up of various types of building blocks where each building block has a particular structure and function. E.g.
  - MLPConv (Network-in-Network)
  - Inception Module (GoogleNet)
  - Fire Module (SqueezeNet)
  - Residual Block (ResNet) etc.

MLPCONV (NETWORK IN NETWORK)

- Introduced series of 1x1 convolutions as a small network (MLP) inside the CNN


(a) Linear convolution layer


(b) Mlpconv layer

MLPCONV (NETWORK IN NETWORK)

- Introduced series of 1x1 convolutions as a small network (MLP) inside the CNN

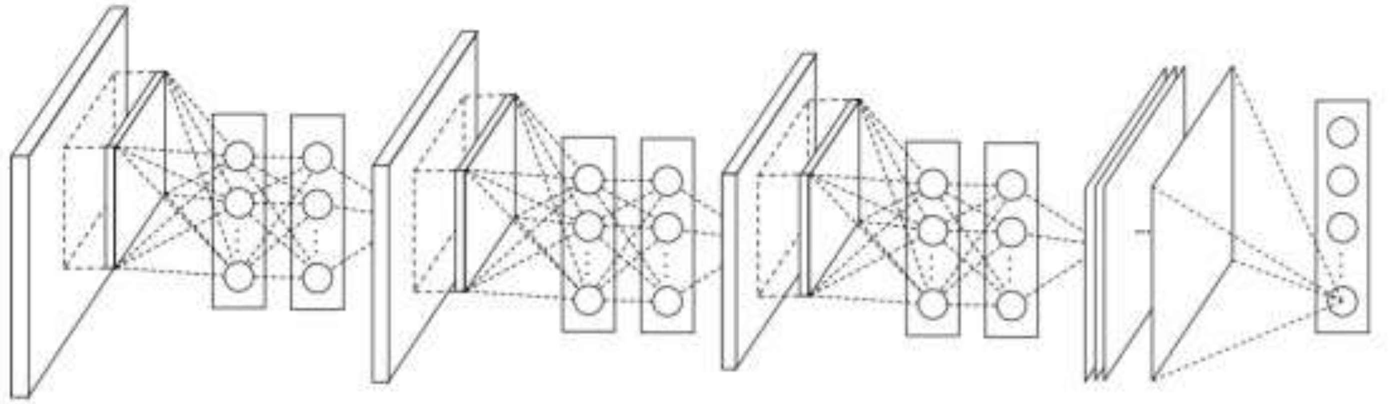
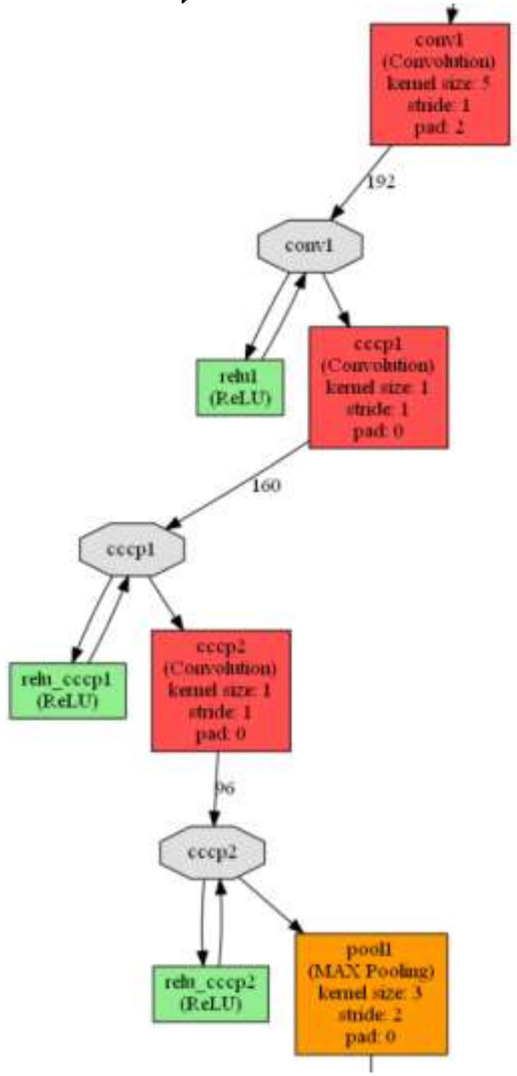
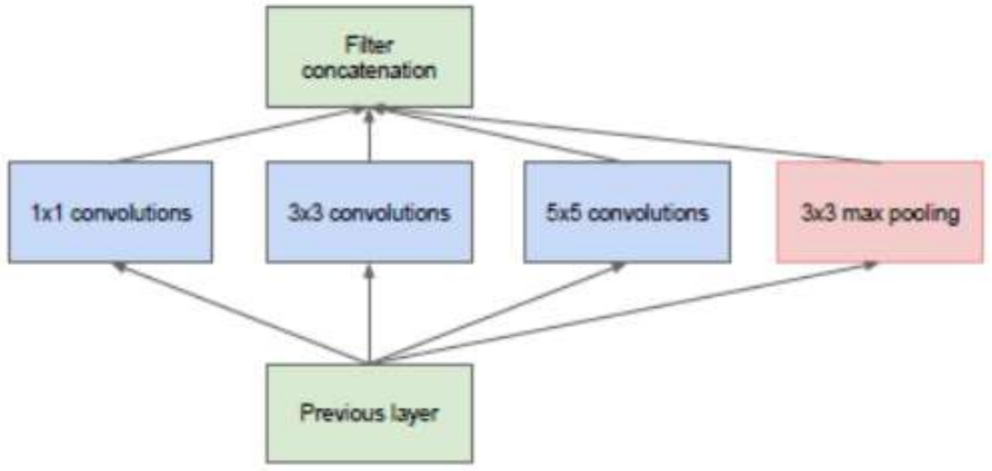


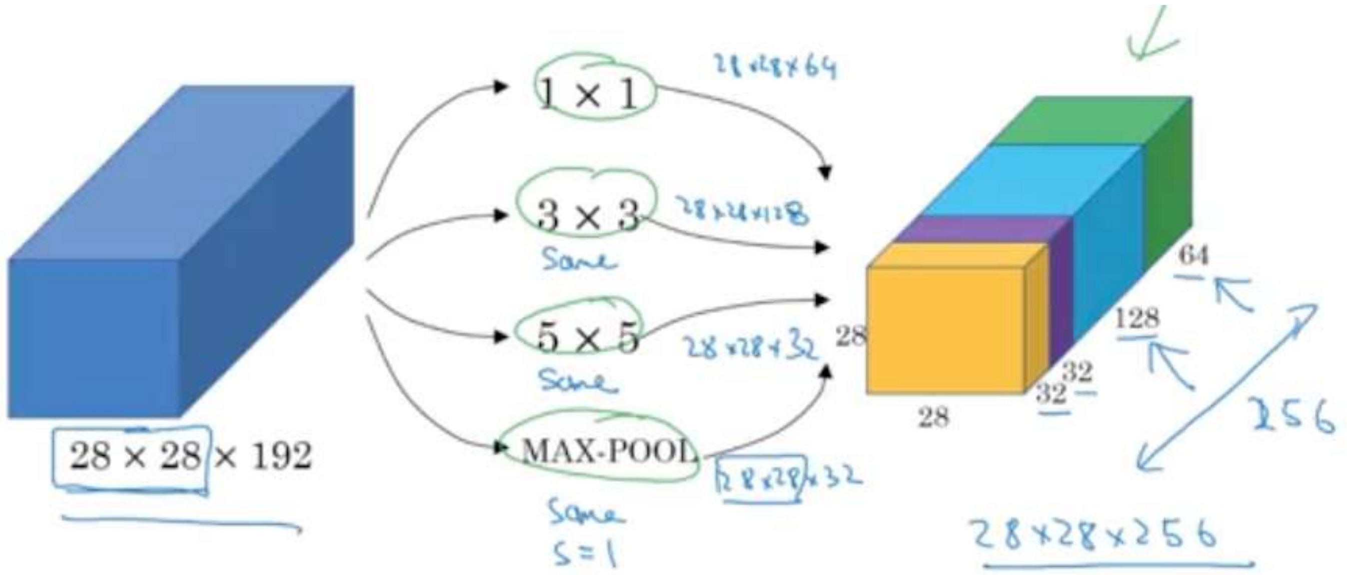
Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.


conv1 (Convolution) kernel size: 5 stride: 1 pad: 2
192
conv1
relu1 (ReLU)
cccp1 (Convolution) kernel size: 1 stride: 1 pad: 0
160
cccp1
relu_cccp1 (ReLU)
cccp2 (Convolution) kernel size: 1 stride: 1 pad: 0
96
cccp2
relu_cccp2 (ReLU)
pool1 (MAX Pooling) kernel size: 3 stride: 2 pad: 0

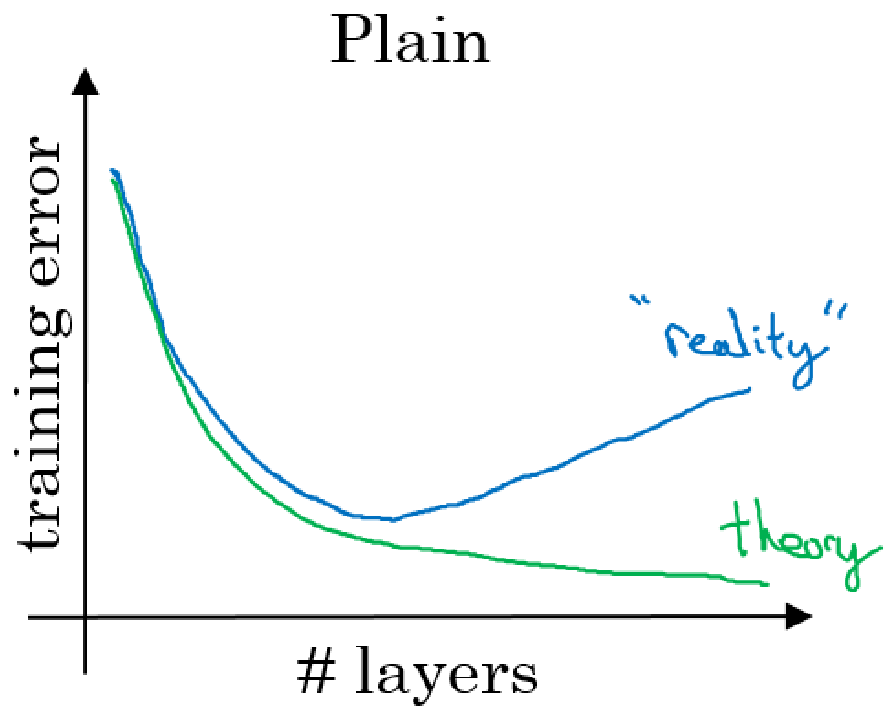
INCEPTION MODULE (GOOGLENET)

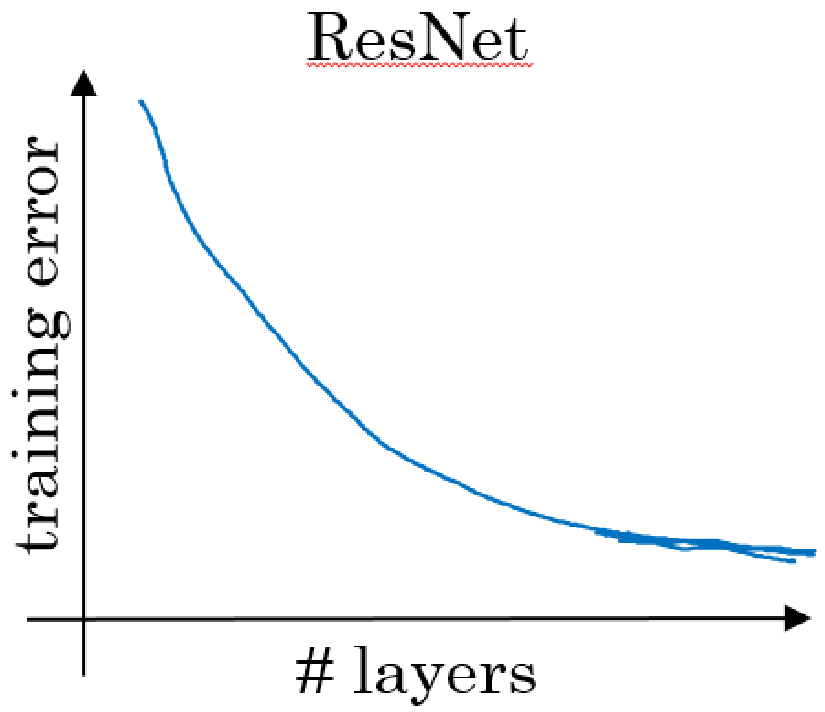


(a) Inception module, naïve version

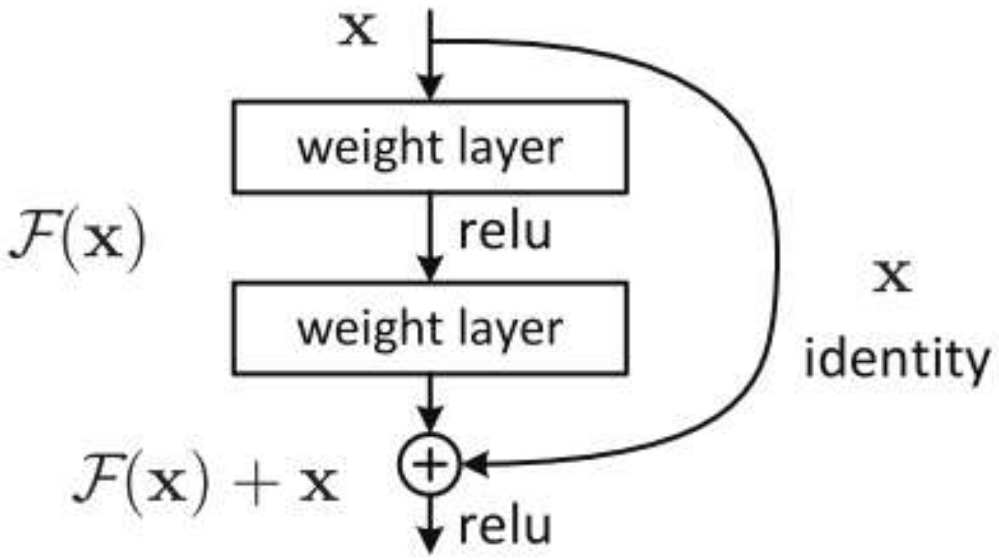


PROBLEMS WITH VERY DEEP NETWORKS

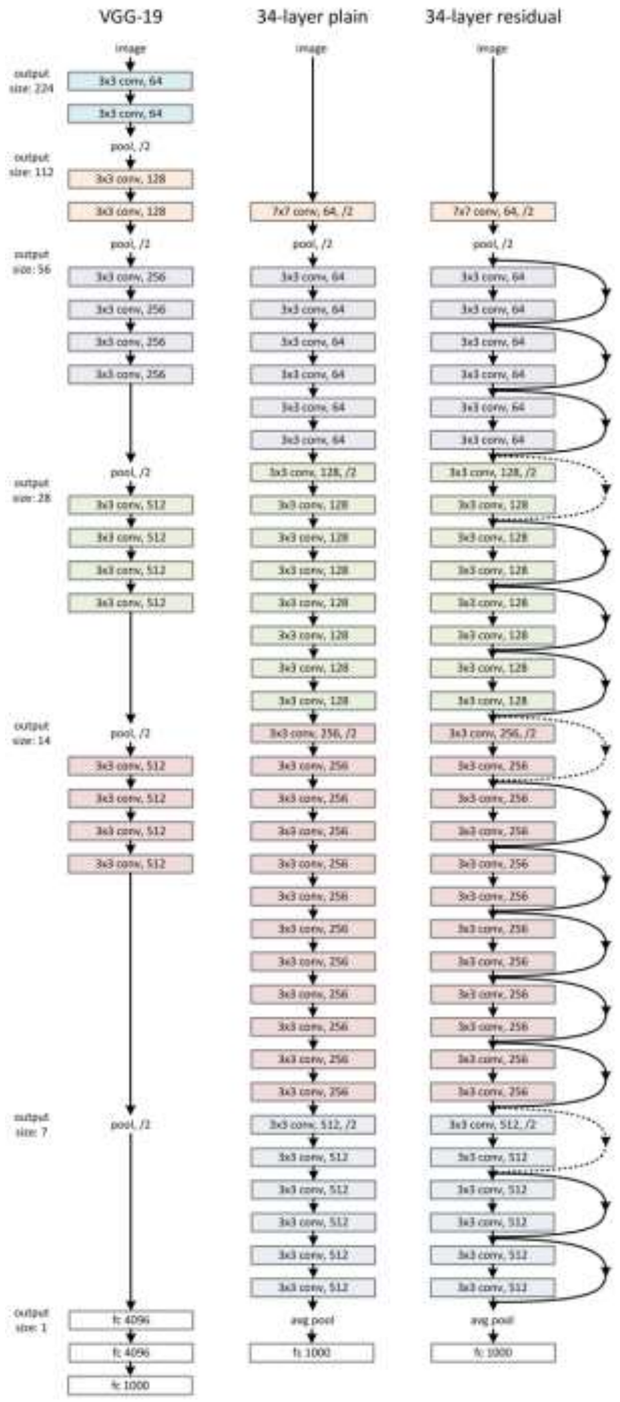




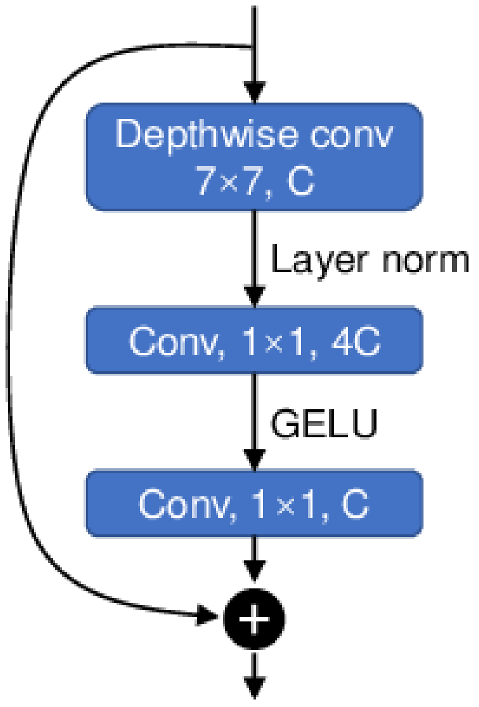
RESIDUAL BLOCK (RESNET)



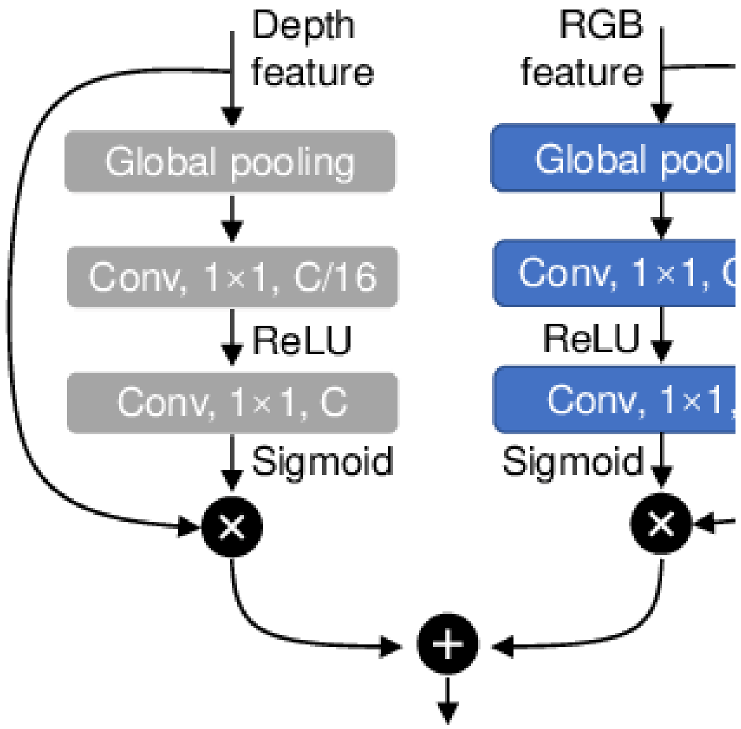
Residual Block



CONVNEXT (MULTI-MODULE)

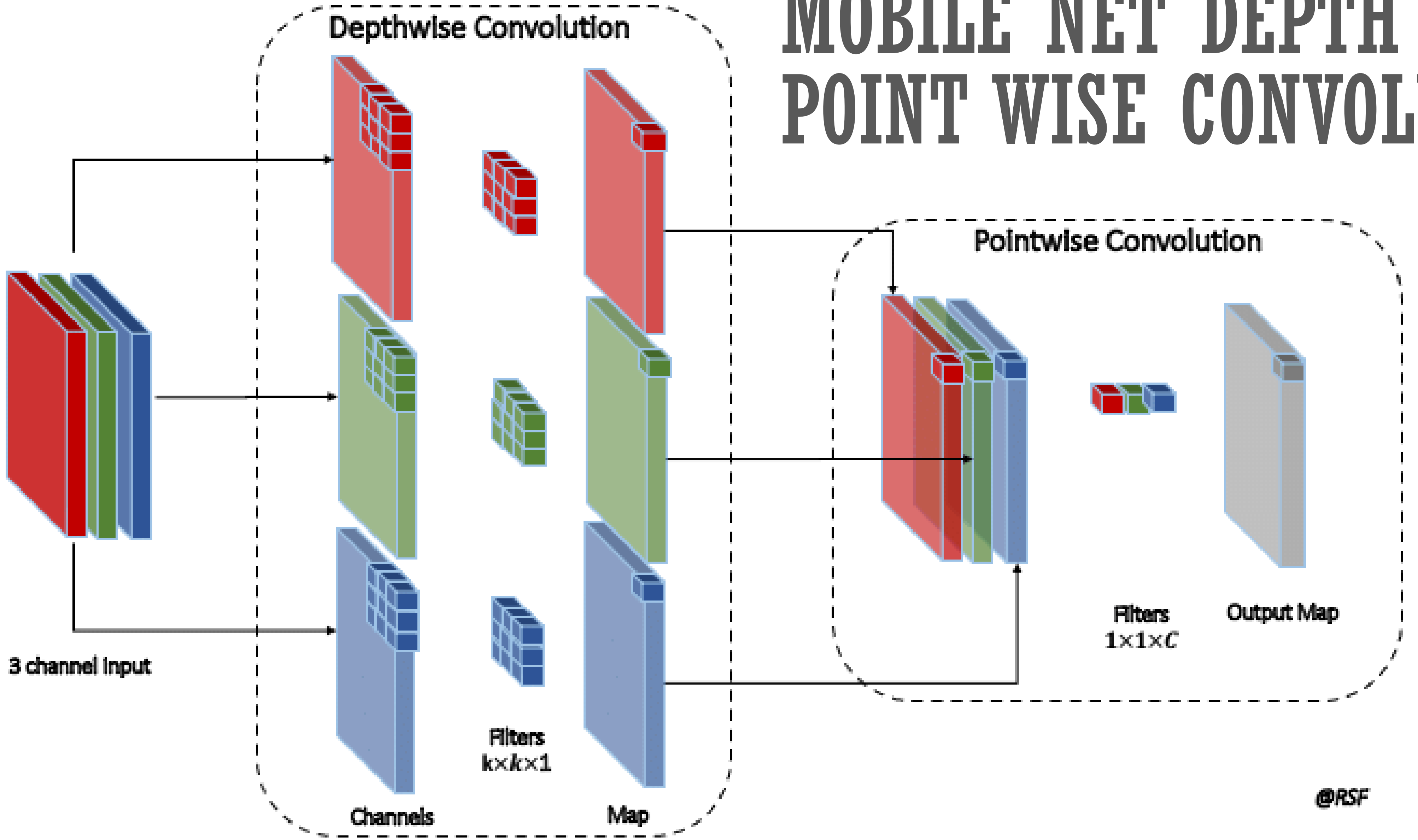


(a) ConvNext Block

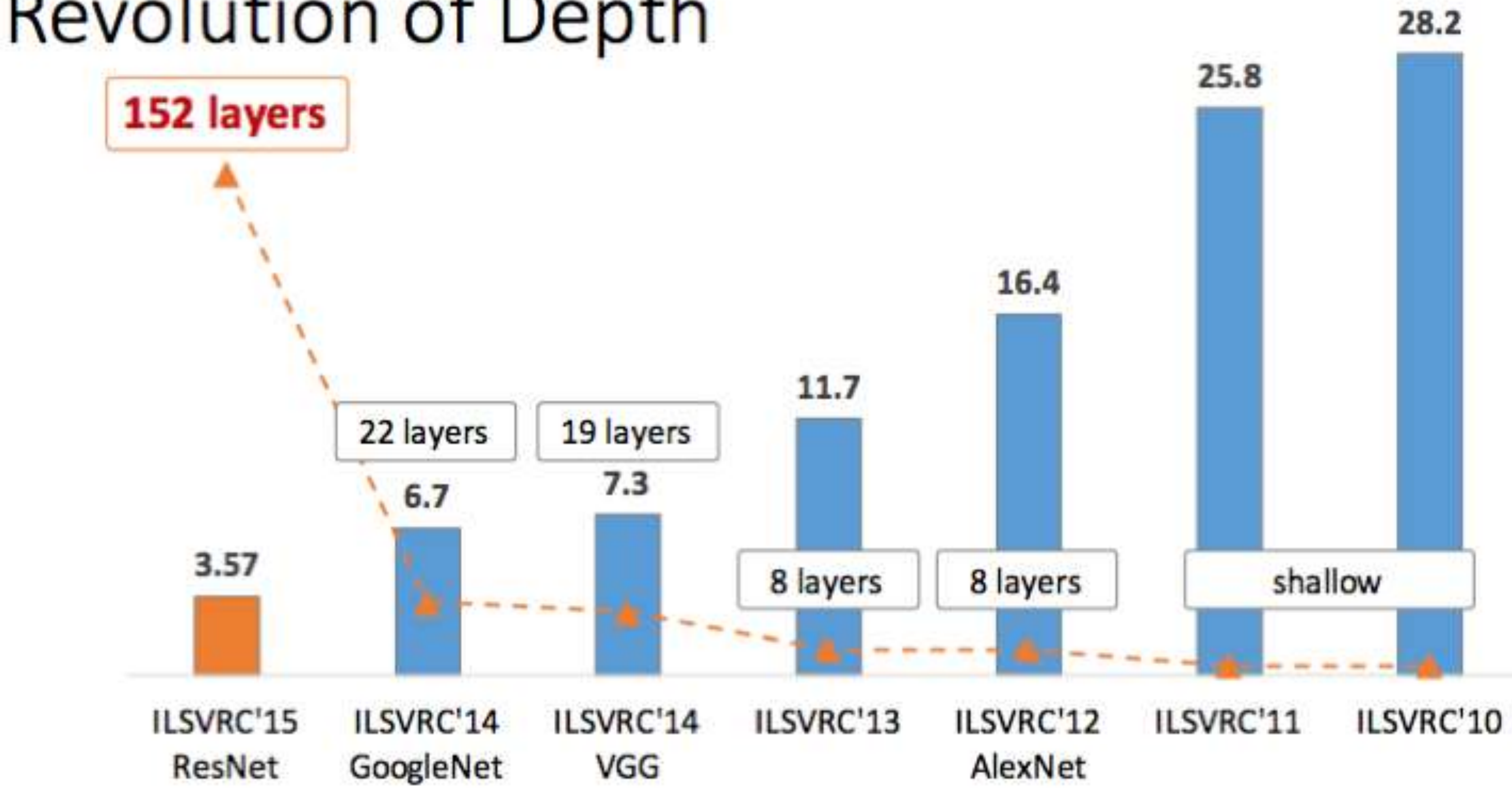


(b) Fusion Module

MOBILE NET DEPTH AND POINT WISE CONVOLUTION



Revolution of Depth



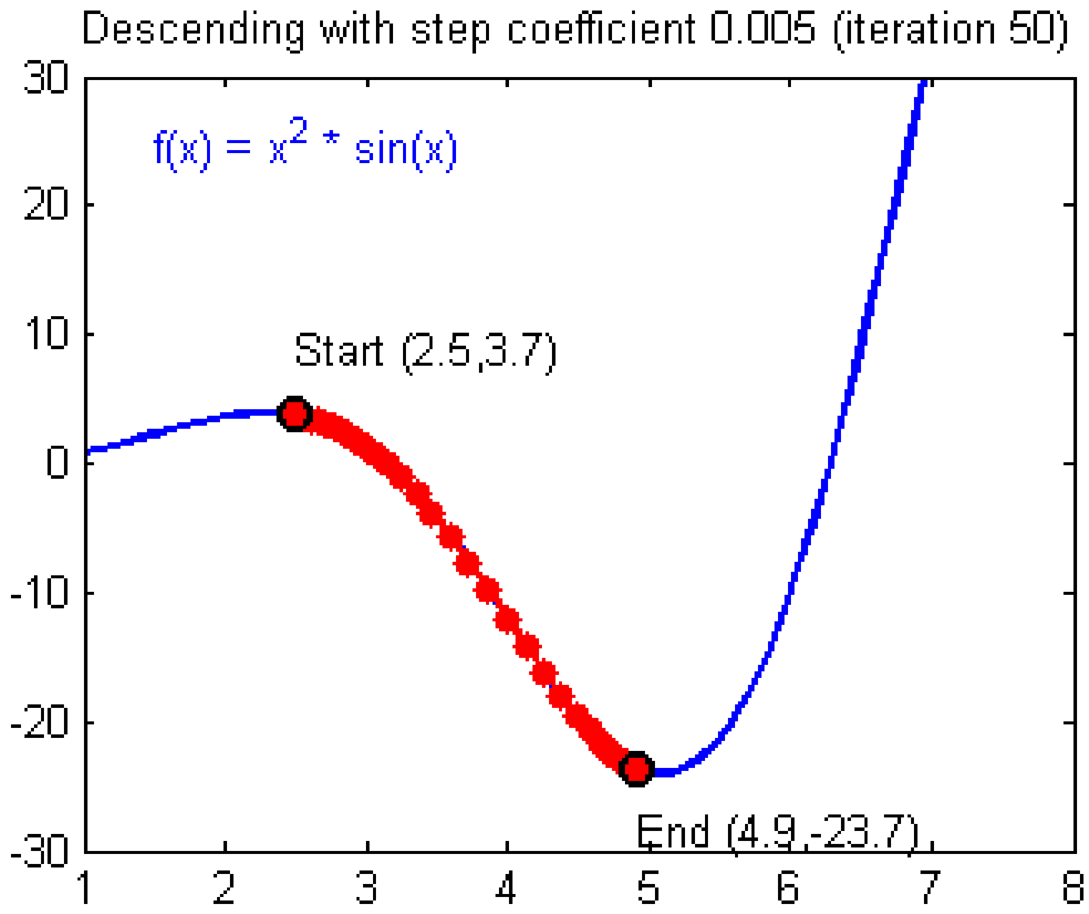
ImageNet Classification top-5 error (%)

GRADIENT DESCENT OPTIMIZATION

- gradient descent, computes the gradient of the cost function w.r.t. to the parameters θ for the entire training dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta).$$

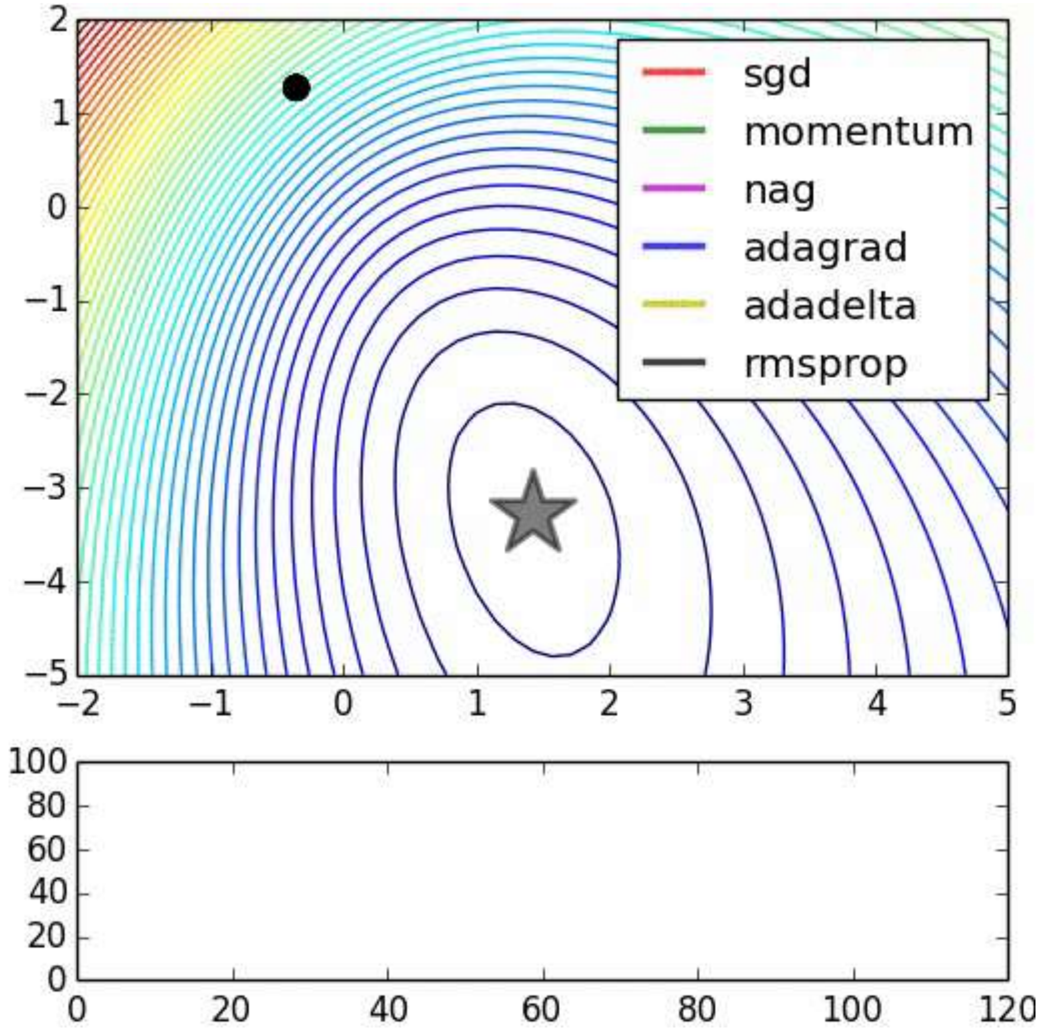
- Calculate the gradients for the whole dataset to perform just *one* update, batch gradient descent can be very slow



STOCHASTIC GRADIENT DESCENT

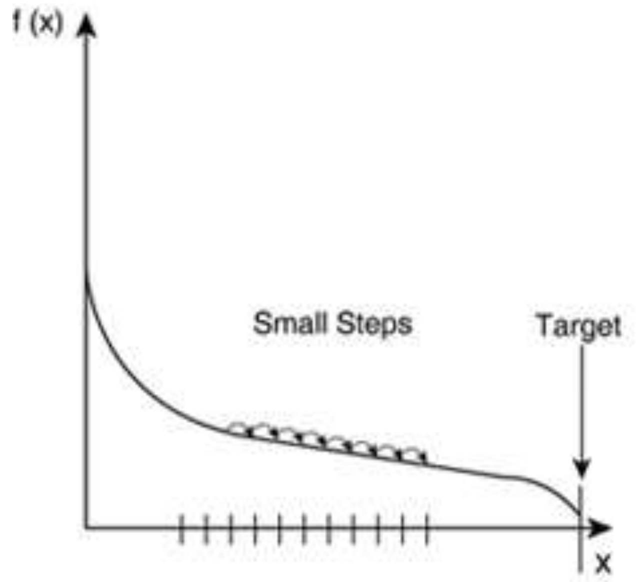
- Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example $x^{(i)}$ and label $y^{(i)}$:

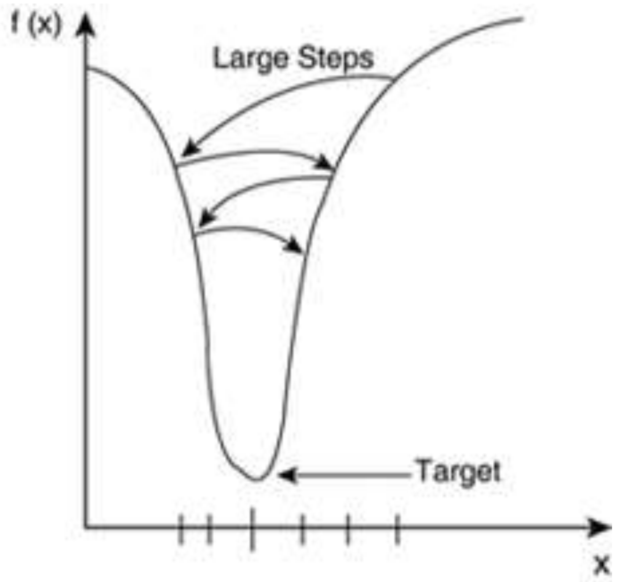
$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$



OPTIMIZATION CHALLENGES

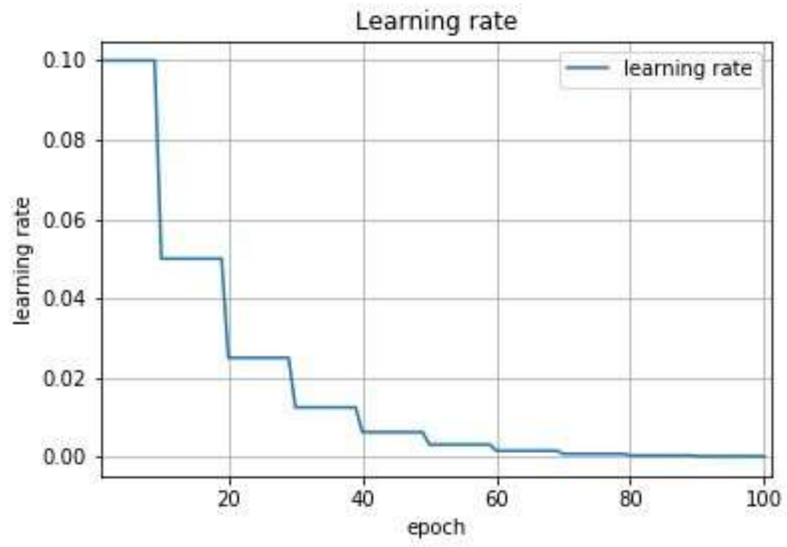
- Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.



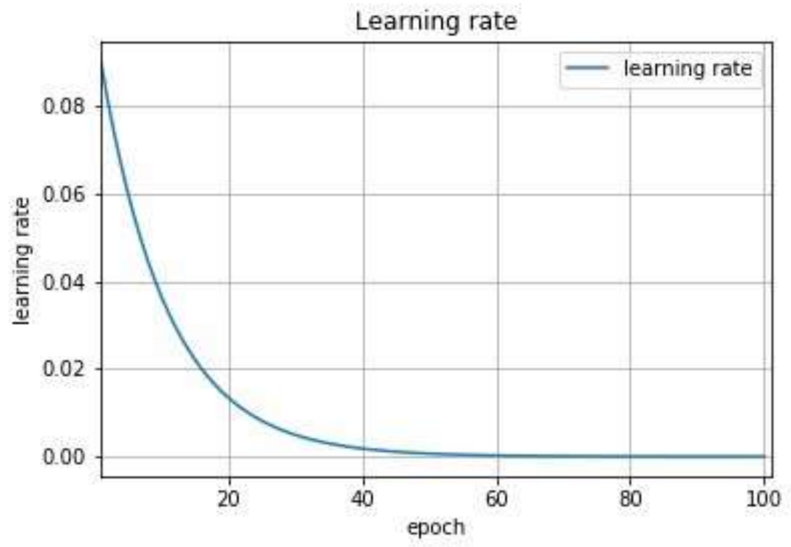


OPTIMIZATION CHALLENGES

- Learning rate schedules try to adjust the learning rate during training by e.g. annealing, i.e. reducing the learning rate according to a pre-defined schedule or when the change in objective between epochs falls below a threshold. These schedules and thresholds, however, have to be defined in advance and are thus unable to adapt to a dataset's characteristics



Step Decay Schedule

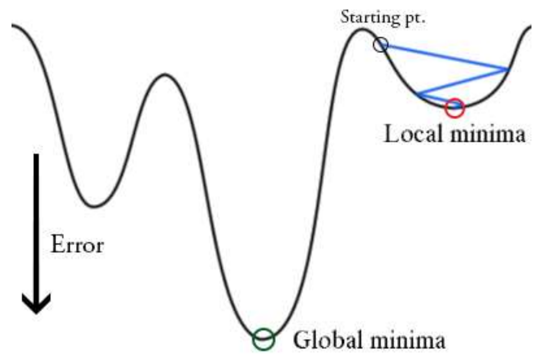


Exponential Decay Schedule

OPTIMIZATION CHALLENGES

- Additionally, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.

- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima.



GRADIENT DESCENT OPTIMIZATION ALGORITHMS (SOLVERS)

- **SGD with Momentum**
- SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima.

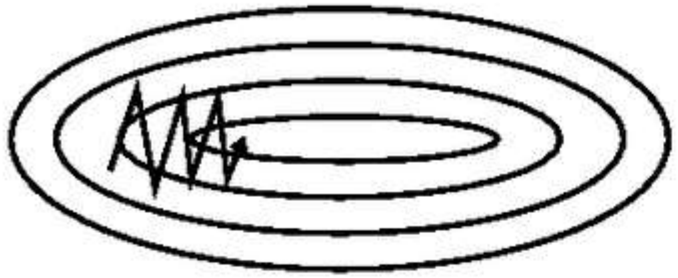


Image 2: SGD without momentum

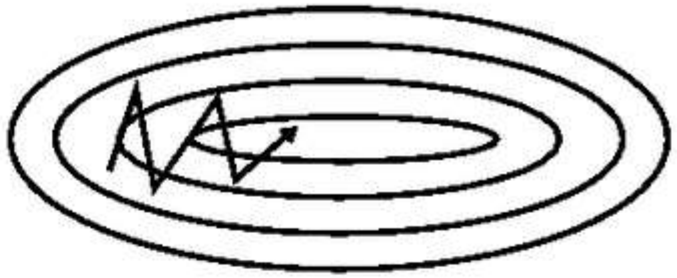


Image 3: SGD with momentum

- Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations as can be seen in Image 3. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

The momentum term γ is usually set to 0.9 or a similar value.

GRADIENT DESCENT OPTIMIZATION ALGORITHMS (SOLVERS)

- **SGD with Momentum**
- Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity if there is air resistance, i.e. $\gamma<1$).

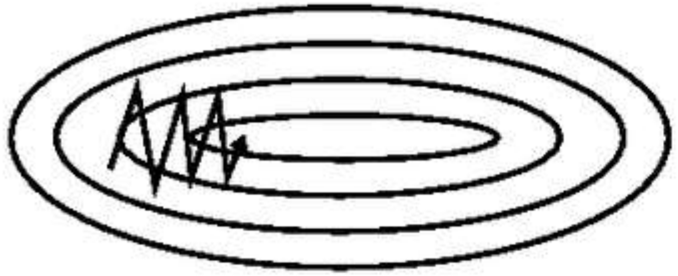


Image 2: SGD without momentum

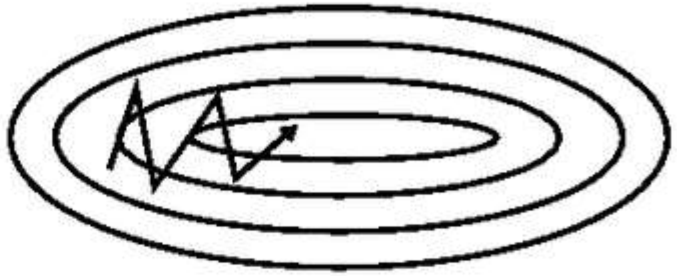
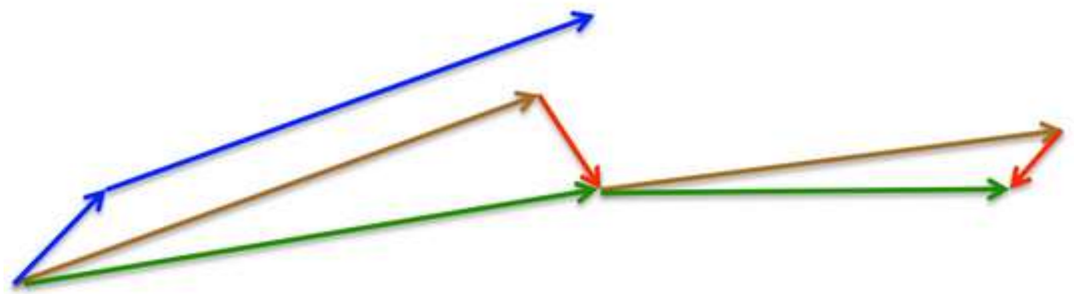


Image 3: SGD with momentum

NESTEROV ACCELERATED GRADIENT

- A ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. We'd like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.
- Nesterov accelerated gradient (NAG) is a way to give our momentum term this kind of prescience.
- We can effectively look ahead by calculating the gradient not w.r.t. to our current parameters θ but w.r.t. the approximate future position of our parameters:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$


Nesterov update

ADAGRAD

- It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data.
- Adagrad uses a different learning rate for every parameter θ_i at every time step t
- $$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$
- $g_{t,i}$ to be the gradient of the objective function w.r.t. to the parameter θ_i at time step t
- $G_t \in R^{d\times d}$ here is a diagonal matrix where each diagonal element i,i is the sum of the squares of the gradients w.r.t. θ_i up to time step t
- ϵ is a smoothing term that avoids division by zero
- One of Adagrad's main benefits is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that.

ADAGRAD

- Adagrad's main weakness is its accumulation of the squared gradients in the denominator:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t.$$

- Since every added term is positive, the accumulated sum keeps growing during training.
- This in turn causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.
- This flaw has been resolved in AdaDelta.

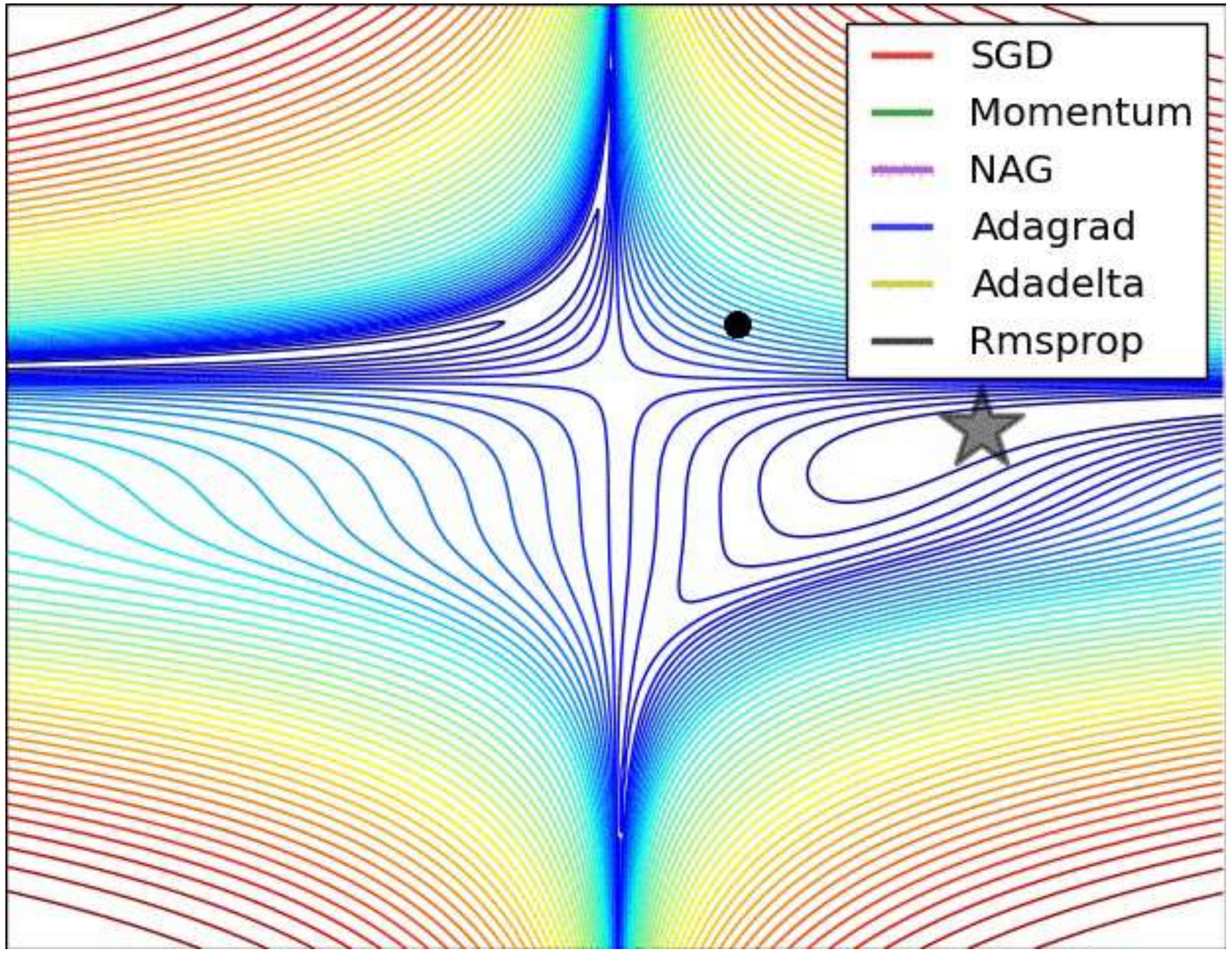
OTHER VARIANTS

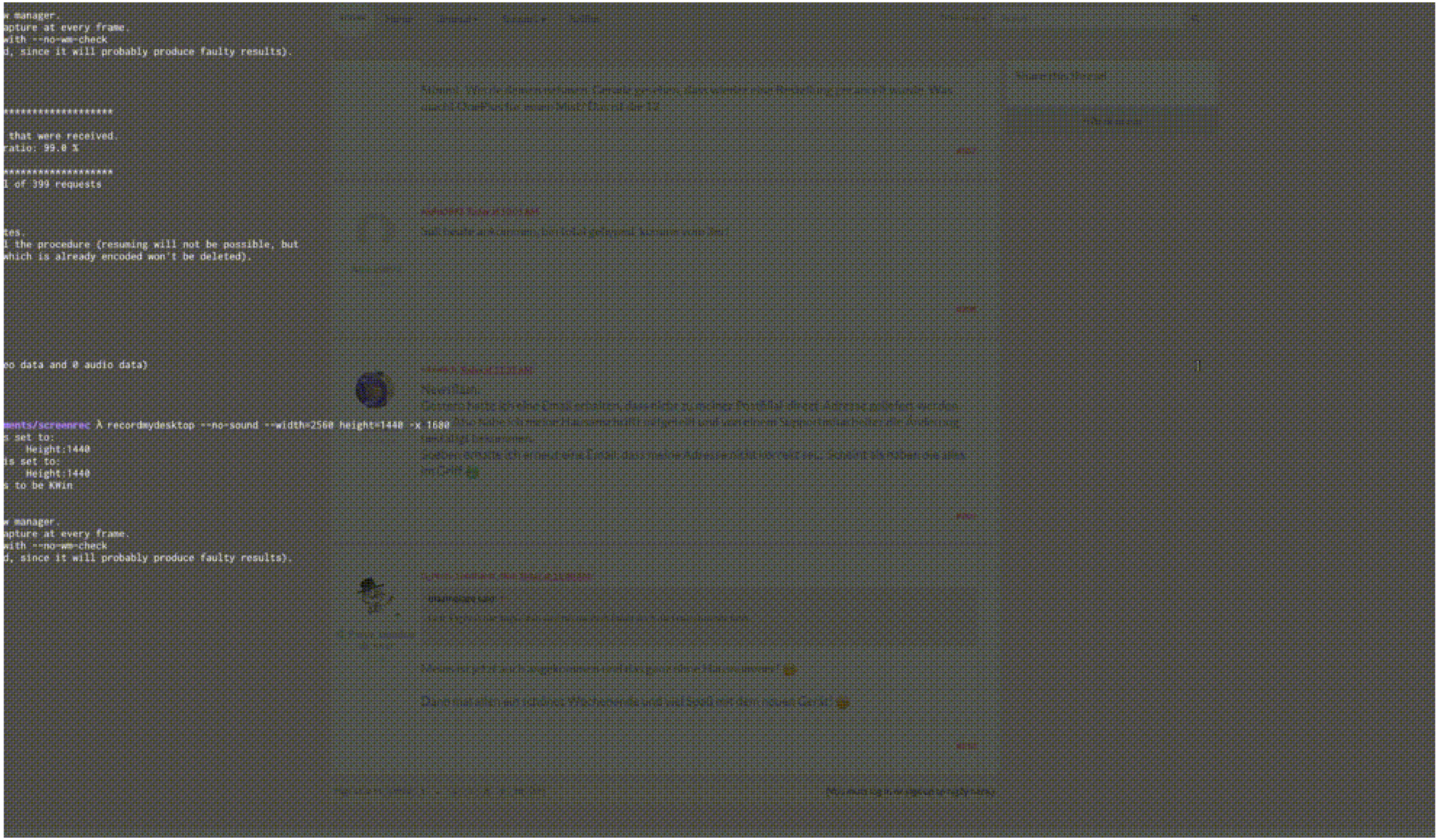
- **Adadelta**
- It is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate.
- Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w.
- With Adadelta, we do not even need to set a default learning rate

- **RMSprop**
- It is an unpublished, adaptive learning rate method proposed by Geoff Hinton
- RMSprop divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests γ to be set to 0.9, while a good default value for the learning rate η is 0.001.

OTHER VARIANTS

- **Adam**
- Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter.
- momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface





```
w manager.
apture at every frame.
with --no-wm-check
d, since it will probably produce faulty results).



********************

 that were received.
ratio: 99.0 %

********************
1 of 399 requests



tes.
l the procedure (resuming will not be possible, but
which is already encoded won't be deleted).








eo data and 0 audio data)




ments/screenrec λ recordmydesktop --no-sound --width=2560 height=1440 -x 1600
s set to:
     Height:1440
is set to:
     Height:1440
s to be KWin


w manager.
apture at every frame.
with --no-wm-check
d, since it will probably produce faulty results).
```

ASSIGNMENT-1

1. Collect and label your own dataset for image classification
2. Construct a CNN model for your data (use existing nets as reference)
3. Report classification accuracy on the validation set

REFERENCES

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. "Caffe: Convolutional architecture for fast feature embedding." In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675-678. ACM, 2014.

http://ruder.io/optimizing-gradient-descent/