Deep Learning Bootcamp Day 3

Agenda

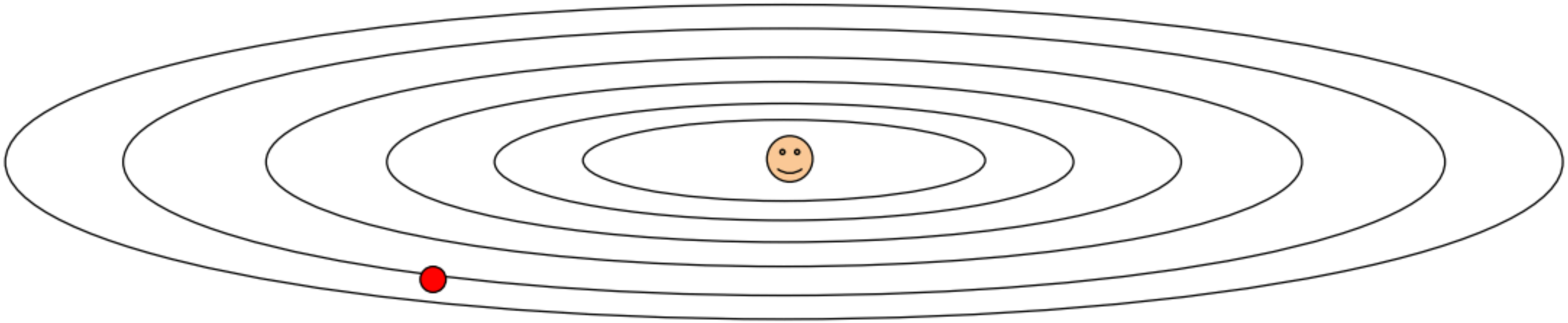
- Optimization
- Architectures

3 Pieces

- **Score**: $f(\vec{x}_i; W) = W\vec{x}_i$
- **Loss**: $L = \dfrac{1}{n}\sum_{i=1}^{n} L_i + \lambda R(W)$
- Use training data to find a W that minimizes L
- **Optimization**: change W in the direction of $-\partial L / \partial W$ to find the optimal W

SGD and bells and whistles

```
# Vanilla update
x += - learning_rate * dx
```

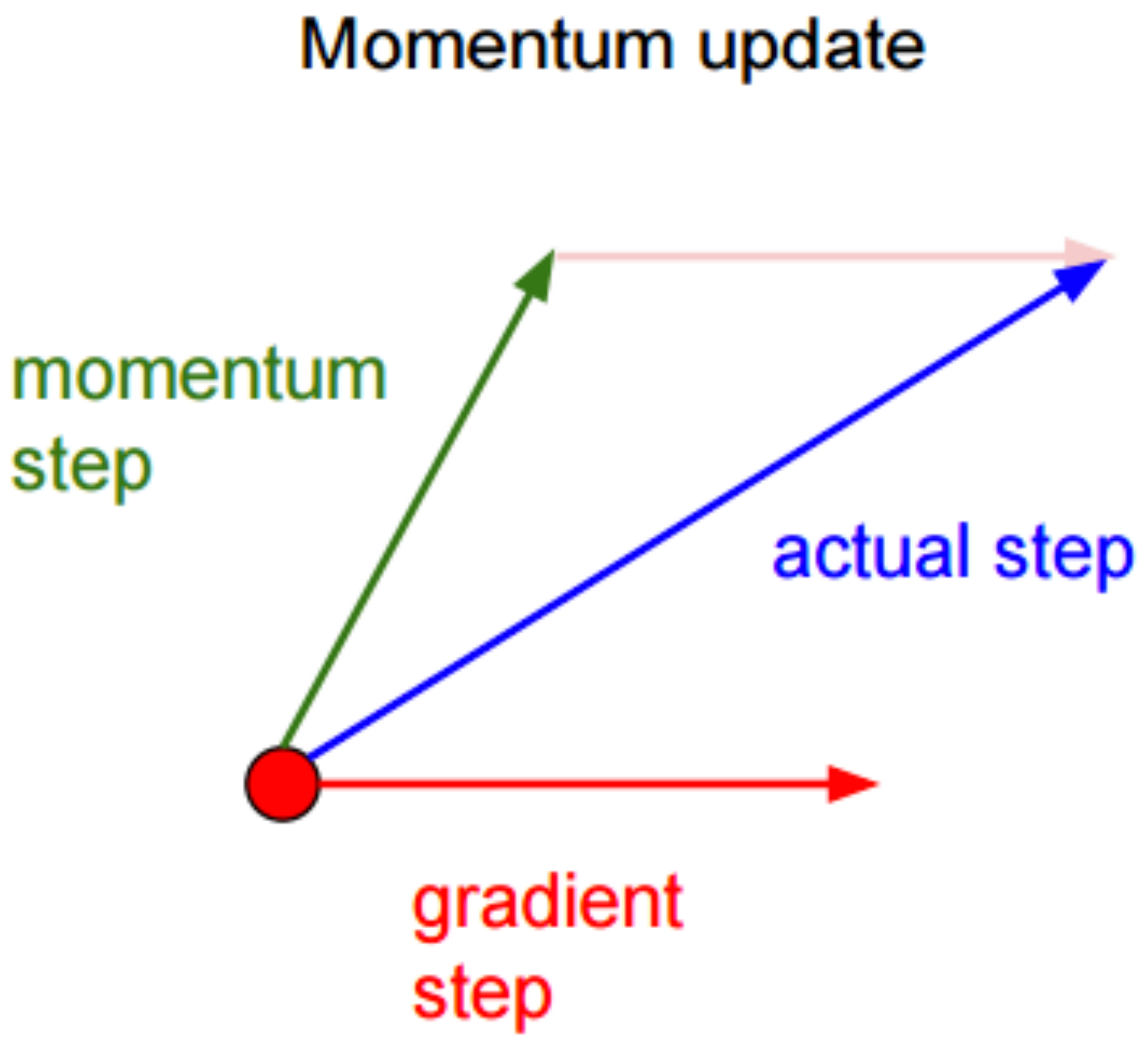


Mini-bath SGD Issues

- Condition number
- Saddle points
- Local minima
- Noisy gradients

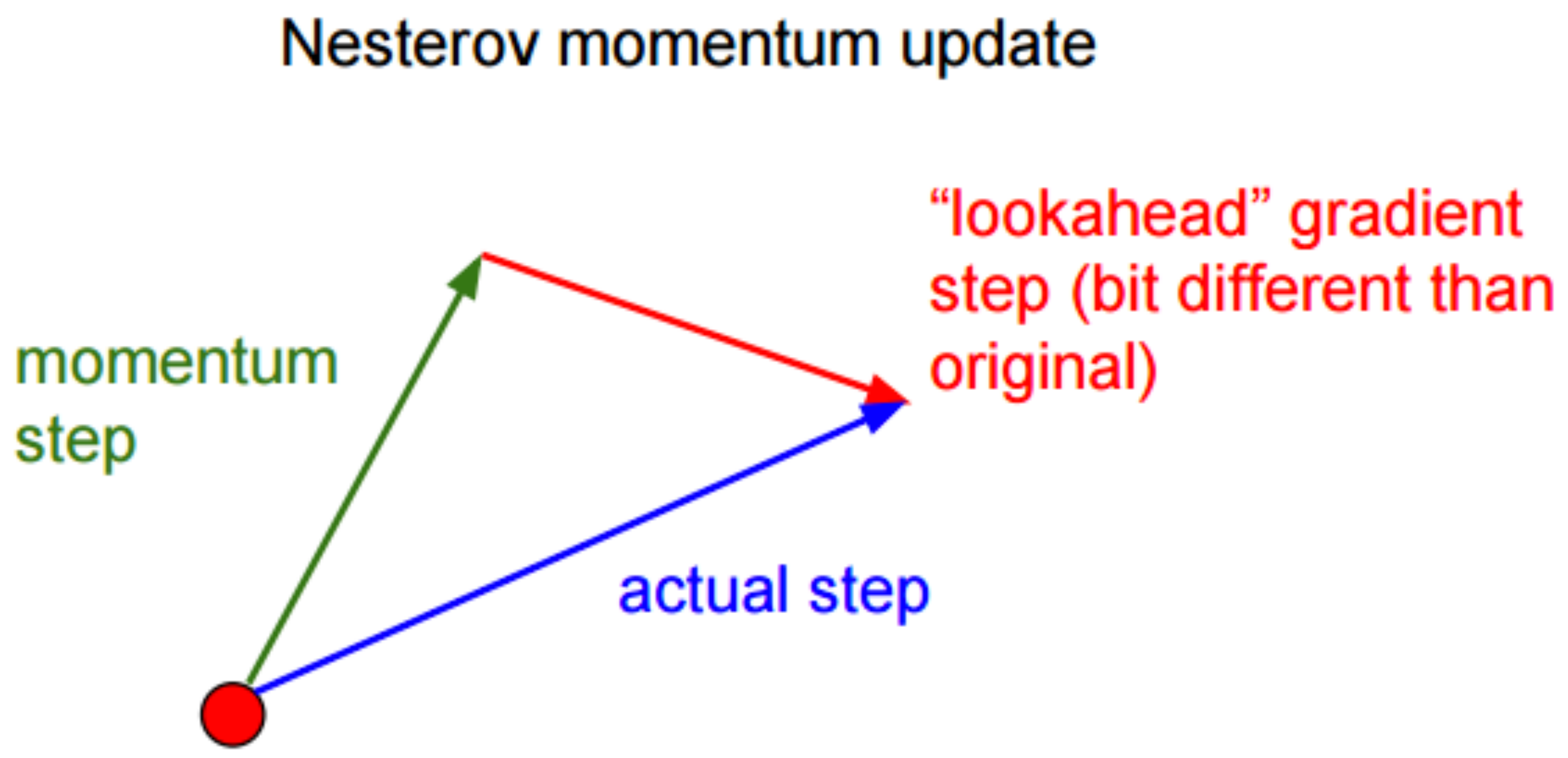
Momentum update

```
# integrate velocity
v = mu * v - learning_rate * dx

# integrate position
x += v
```



Nesterov Momentum (2012)

```
x_ahead = x + mu * v
# evaluate dx_ahead
# (the gradient at x_ahead
# instead of at x)
v = mu * v - learning_rate * dx_ahead
x += v

# This alternative preferred
v_prev = v # back this up
# velocity update stays the same
v = mu * v - learning_rate * dx
# position update changes form
x += -mu * v_prev + (1 + mu) * v
```



Per-parameter adaptive learning rate methods

- Adagrad
- RMSprop
- Adam

Adagrad (2011)

```
# Assume the gradient dx and parameter vector x
cache += dx**2
x += - learning_rate * dx / np.sqrt(cache + 1e-7)
```

- cache has size equal to gradient `dx`
- Weights that receive high gradients will have their effective learning rate reduced

RMSprop

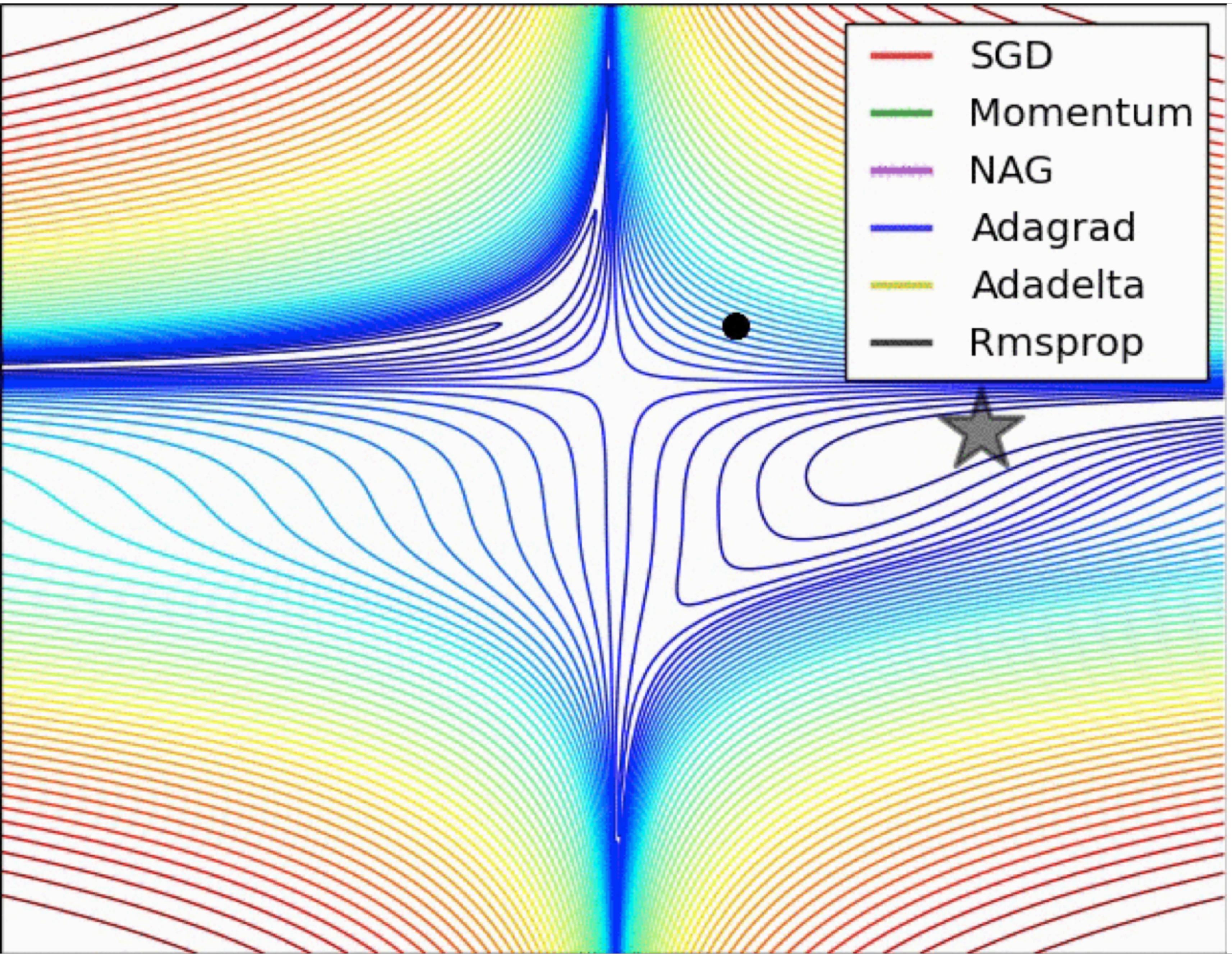
Very effective, but currently unpublished (most currently cite Lecture 6: slide 29 of Geoff Hinton's Coursera class!)

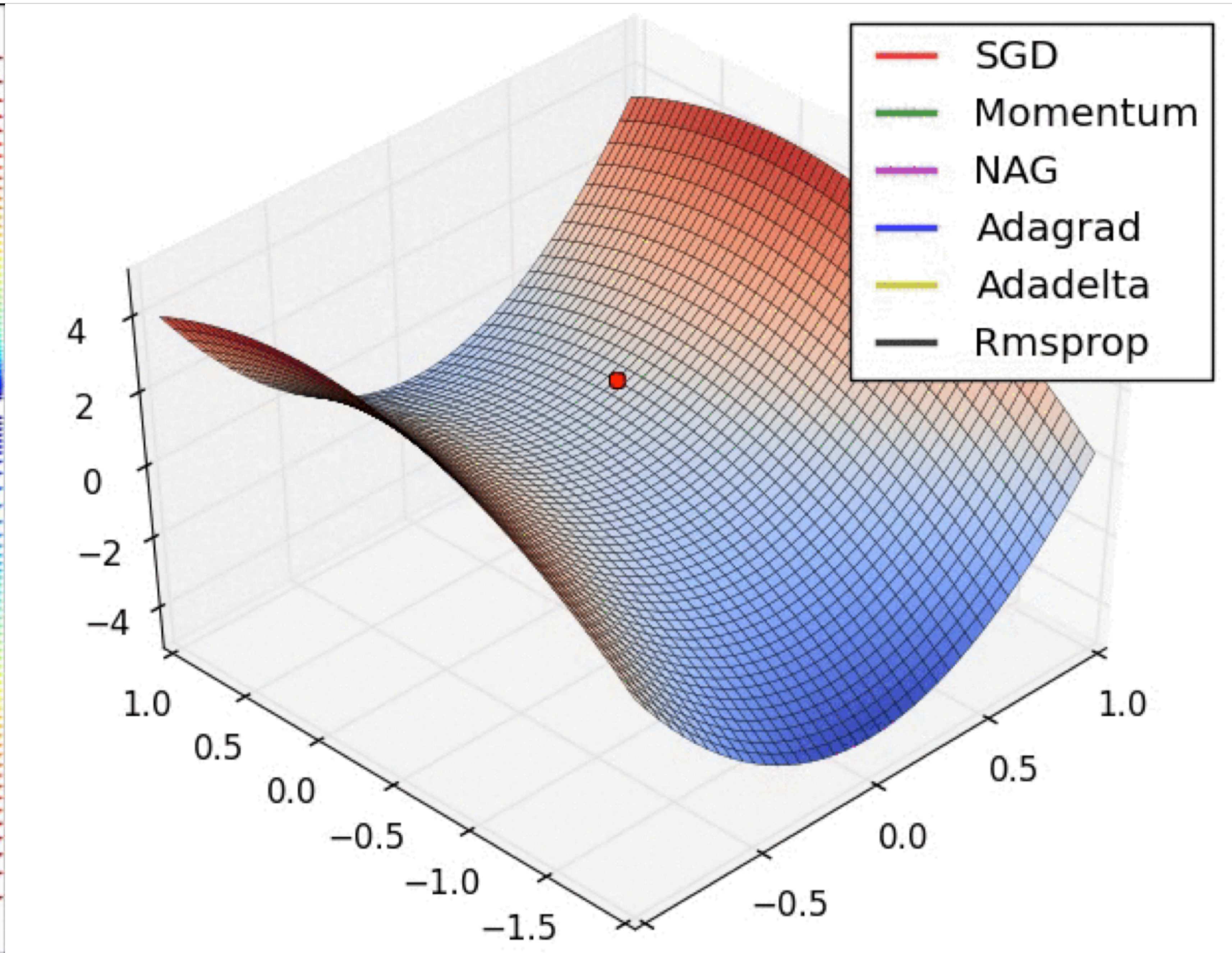
```
# Assume the gradient dx and parameter vector x
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / np.sqrt(cache + 1e-7)
```

- `decay_rate` is a hyperparameter and typical values are `[0.9, 0.99, 0.999]`
- cache variable is "leaky"

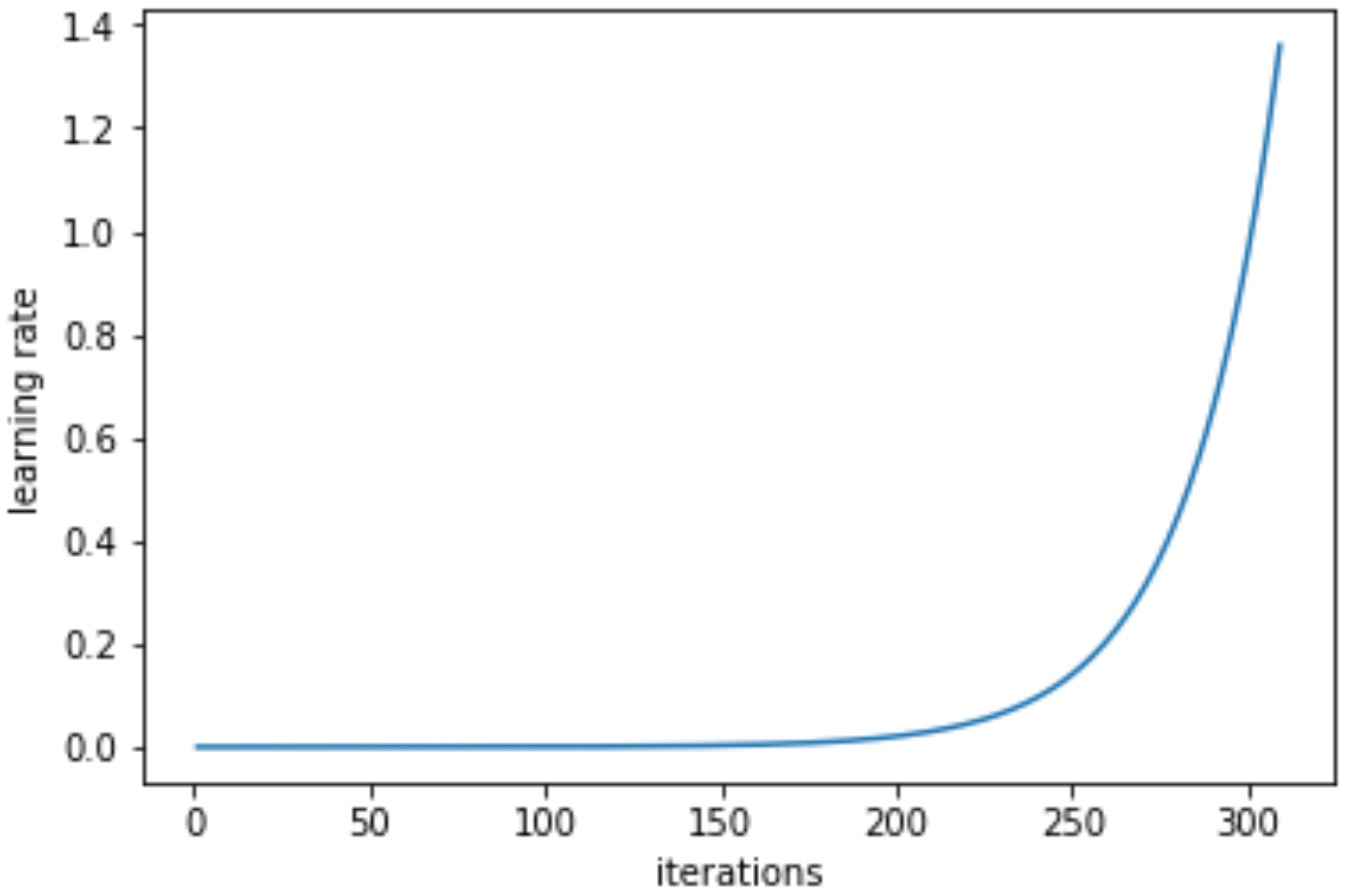
Adam (2014)

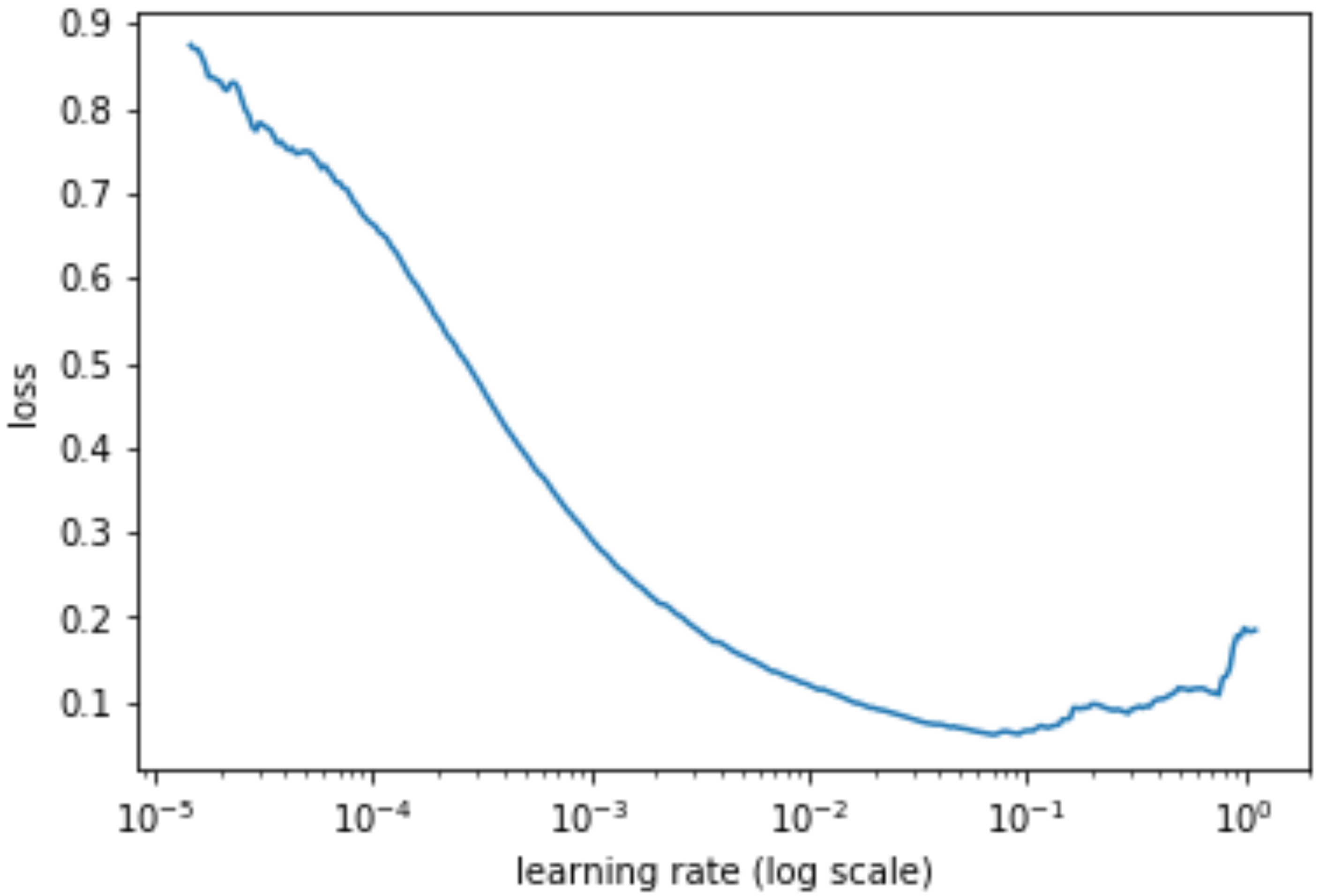
```
# Check paper for implementation

m = beta1*m + (1-beta1)*dx
v = beta2*v + (1-beta2)*(dx**2)

x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

[Cyclical Learning Rates for Training Neural Networks](#) (2015)





Annealing the learning rate

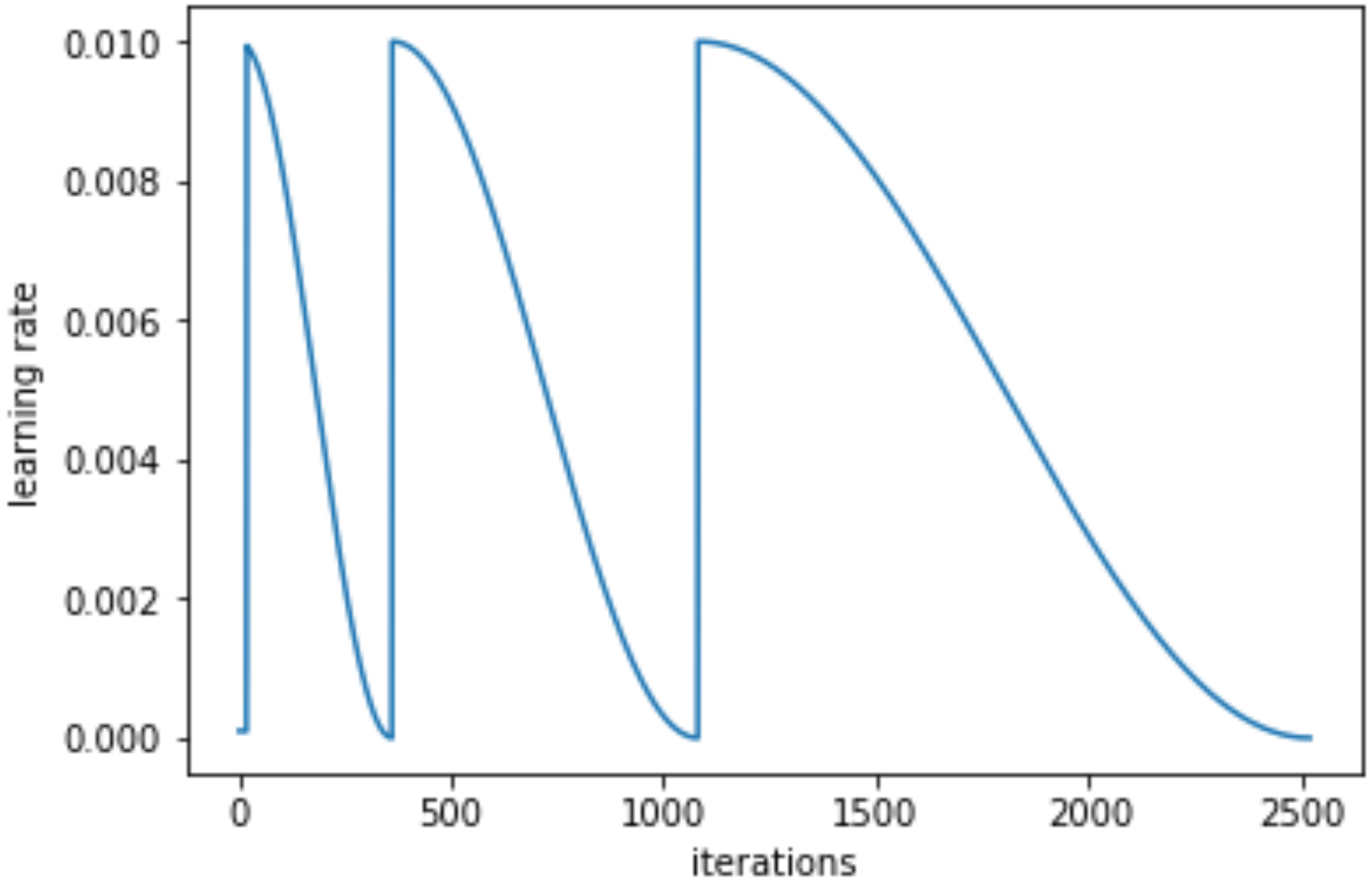
- High learning rate: too much kinetic energy, parameter vector bounces around chaotically
- Knowing when to decay the learning rate: tricky
- Three common ways:
  - Step decay
  - Exponential decay
  - $1/t$ decay

Step decay

- Reduce the learning rate by some factor every few epochs
- Typically: reduce learning rate by a half every 5 epochs or 0.1 every 20 epochs
- Depend on the type of problem and model
- Reduce rate by constant (e.g. 0.5) whenever the validation error stops improving

Others

- Exponential decay: $\alpha = \alpha_0 \exp(-kt)$ where α_0 and k are hyperparameters and t is the iteration number (but you can also use units of epochs)
- $1/t$ decay: $\alpha = \alpha_0/(1+kt)$ where again α_0, k and t are as before
- Cosine annealing



ConvNet Architectures

- CONV
- POOL
- FC
- RELU

Layer Patterns

- Stack a few CONV-RELU-POOL layers
- Repeat this pattern until the image has been merged spatially to a small size
- Then transition to fully-connected layers

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

- `INPUT -> FC`: a linear classifier where `N = M = K = 0`
- `INPUT -> CONV -> RELU -> FC`
- `INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC`: a single CONV layer between every POOL layer
- `INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC`: two CONV layers stacked before every POOL layer

Prefer a stack of small filter CONV to one large receptive field CONV layer.

Layer Sizing Patterns

- The input layer should be divisible by 2 many times: 32, 64, 96, 224, 384, and 512
- The conv layers should be using small filters: e.g. 3x3, using a stride of 1, and crucially, padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input
- The pool layer: use maximum and 2x2 with stride of 2

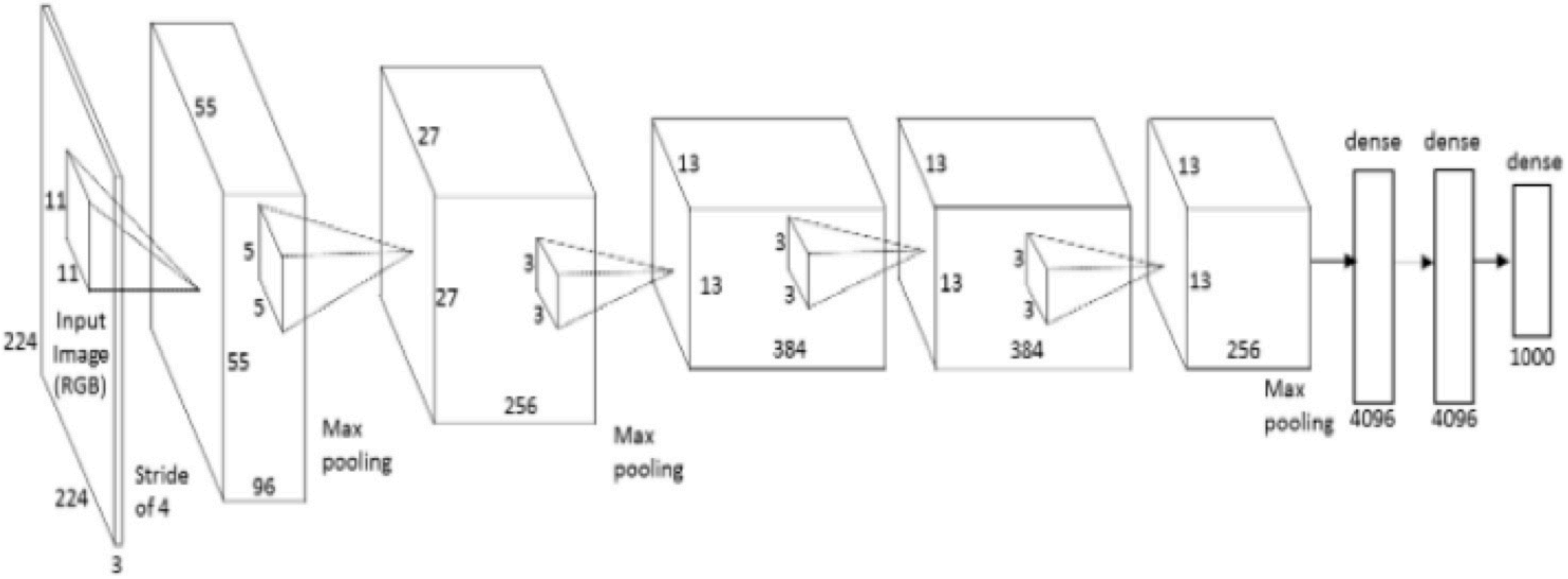
FC vs Conv Layer

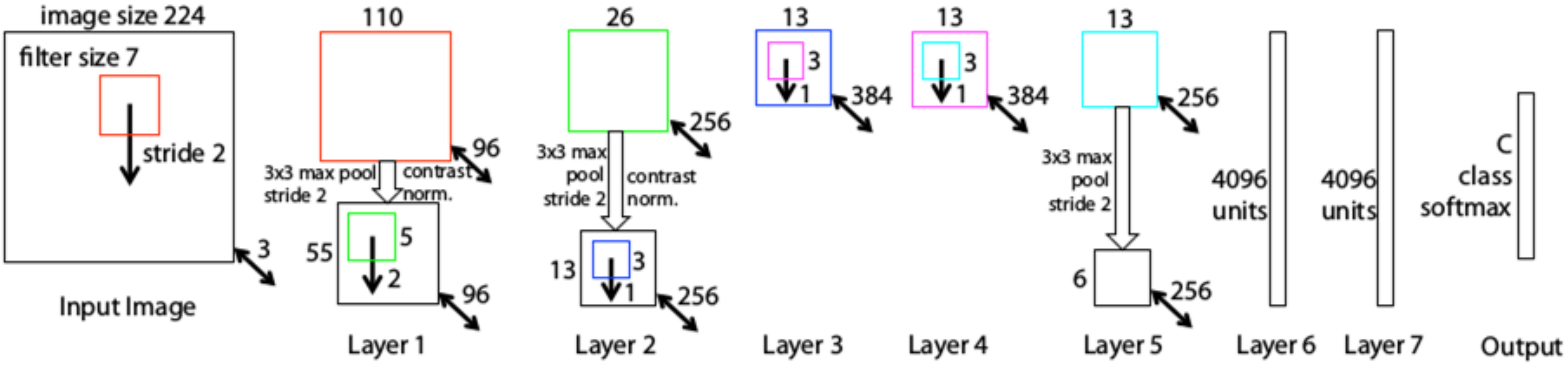
- Only difference:
  - neurons in the CONV layer are connected only to a local region in the input
  - many of the neurons in a CONV volume share neurons
- Neurons in both layers still compute dot products
- Possible to convert between FC and CONV layers

- For any CONV layer there is an FC layer that implements the same forward function
- Any FC layer can be converted to a CONV layer: setting the filter size to be exactly the size of the input volume

Convolution layer

- Input $W_1 \times H_1 \times D_1$
- Needs 4 parameters: K number of filters, F spatial extent, S stride and P zero padding
- Outputs volume: $W_2 = (W_1 - F + 2P)/S + 1$, $H_2 = (H_1 - F + 2P)/S + 1$ and $D_2 = K$
- Parameters: $(F \times F \times D_1) \times K$ weights and K biases


224
224
3
11
11
Input
Image
(RGB)
Stride
of 4
55
55
96
5
5
Max
pooling
27
27
256
3
3
Max
pooling
13
13
384
3
3
13
13
384
3
3
13
13
256
Max
pooling
dense
4096
dense
4096
dense
1000



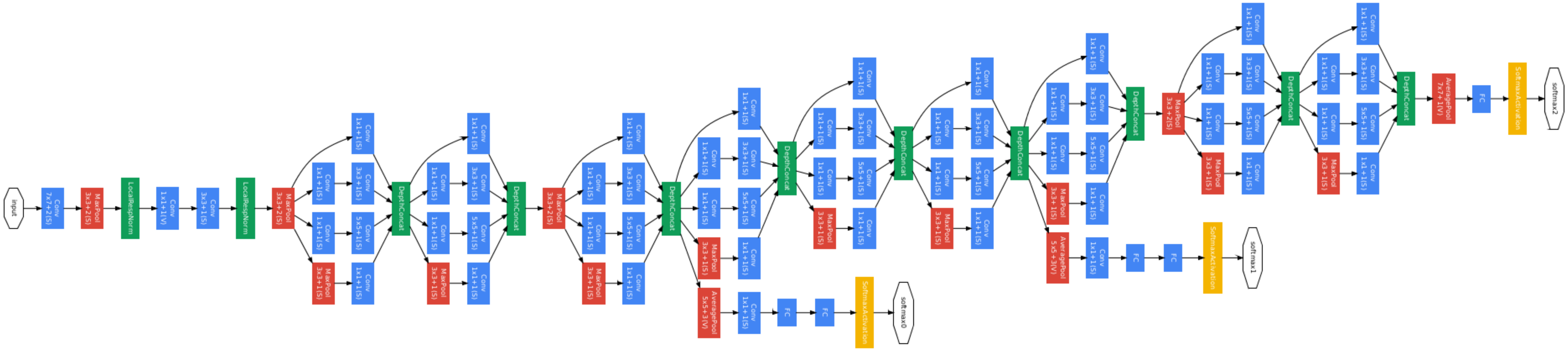
ZF Net Architecture

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

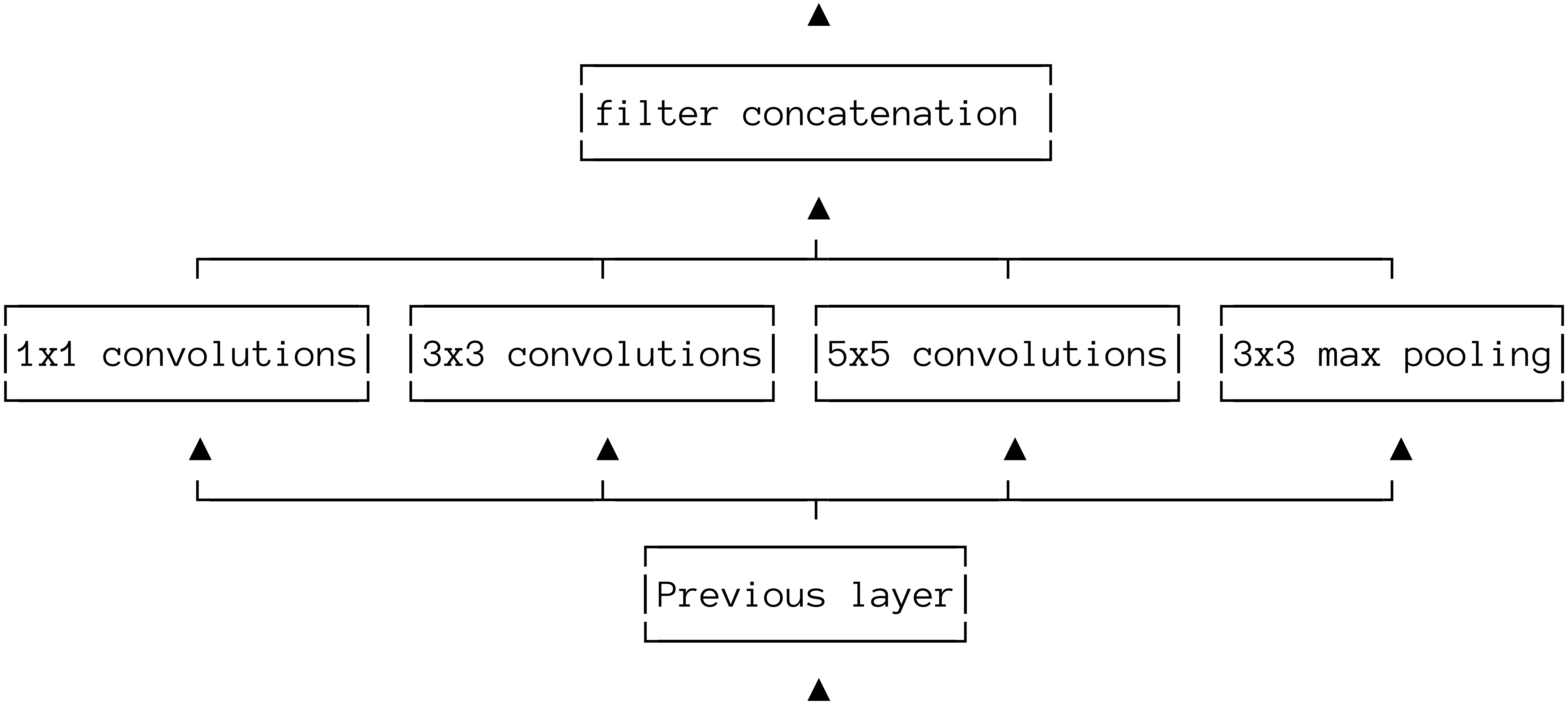
Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

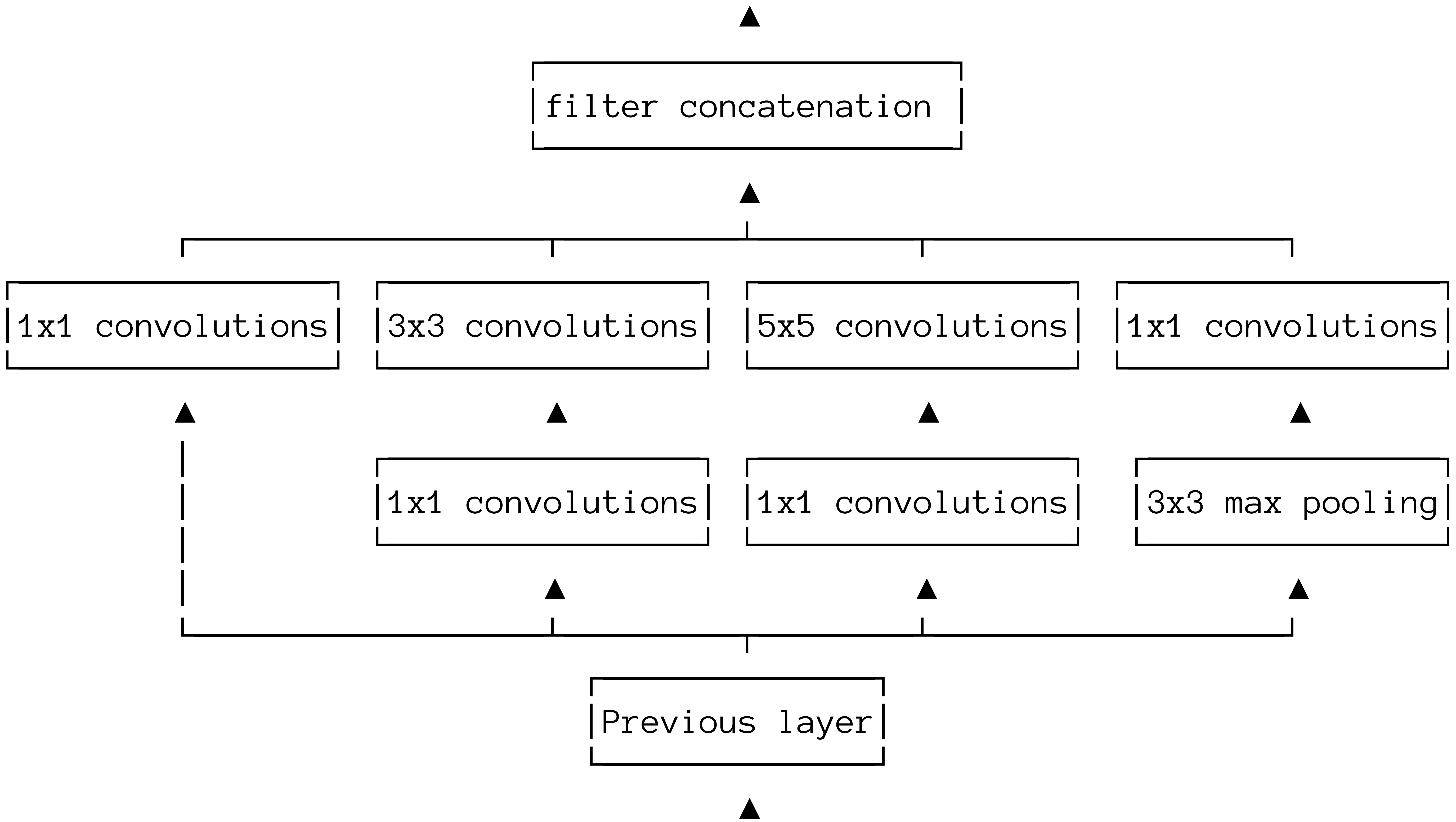
```
INPUT:    [224x224x3]   memory: 224*224*3=150K  weights: 0
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M weights: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M weights: (3*3*64)*64 = 36,864
POOL2:    [112x112x64]  memory: 112*112*64=800K weights: 0
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory:  112*112*128=1.6M   weights: (3*3*128)*128 = 147,456
POOL2: [56x56x128]  memory:  56*56*128=400K   weights: 0
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory:  56*56*256=800K   weights: (3*3*256)*256 = 589,824
POOL2: [28x28x256]  memory:  28*28*256=200K   weights: 0
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory:  28*28*512=400K   weights: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]  memory:  14*14*512=100K   weights: 0
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory:  14*14*512=100K   weights: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]  memory:  7*7*512=25K  weights: 0
FC: [1x1x4096]  memory:  4096  weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]  memory:  4096  weights: 4096*4096 = 16,777,216
FC: [1x1x1000]  memory:  1000 weights: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
```


input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
Conv 1x1+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
AveragePool 5x5+3(V)
FC
SoftmaxActivation
softmax0
softmax1
AveragePool 7x7+1(V)
softmax2


WE NEED TO GO
DEEPER

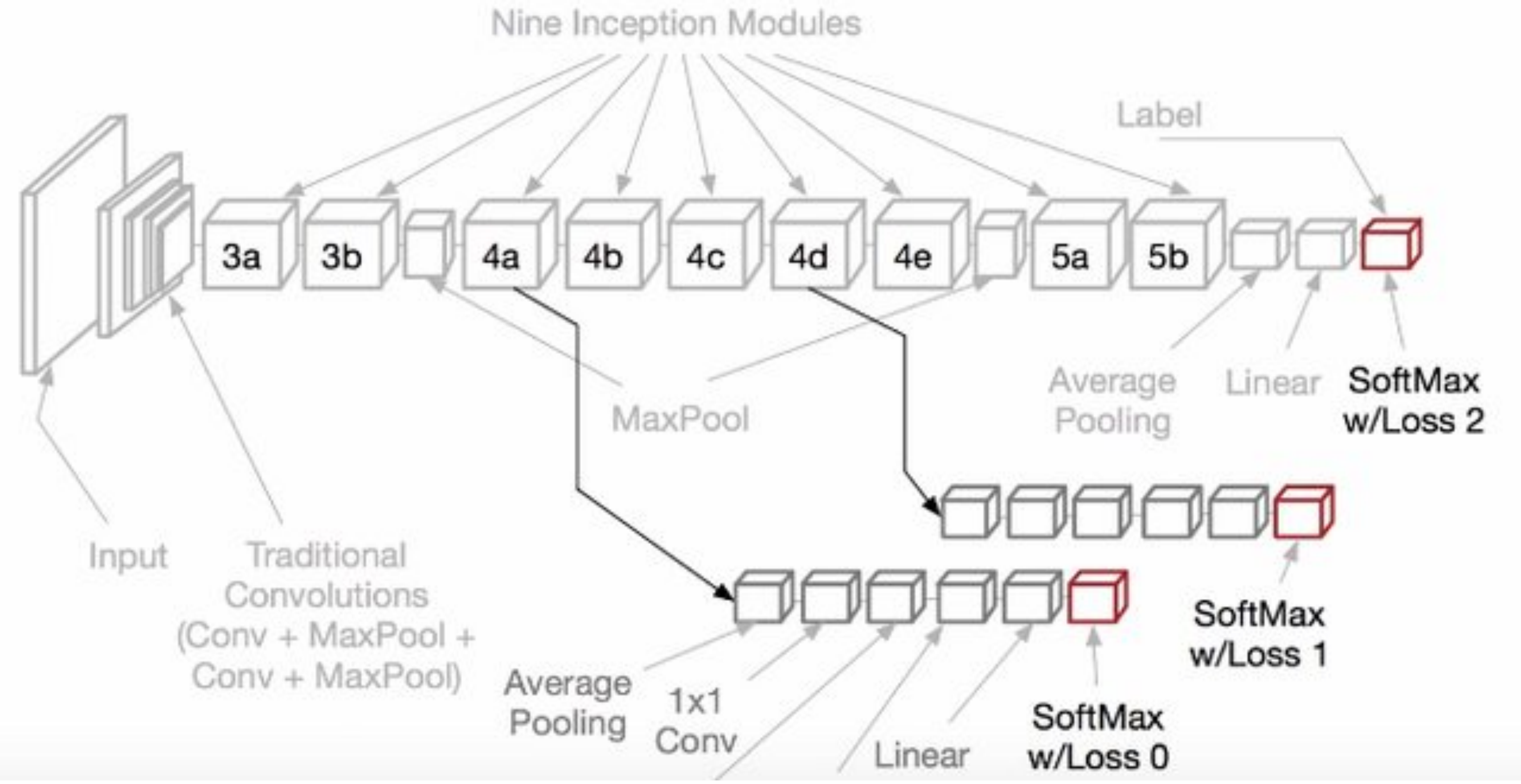
Inception module (2014)




filter concatenation
1x1 convolutions
3x3 convolutions
5x5 convolutions
1x1 convolutions
1x1 convolutions
1x1 convolutions
3x3 max pooling
Previous layer

Auxiliary classifiers

- Gradient carry less and less information the deeper we are (vanishing gradient problem)
- Perhaps intermediate feature have some discriminatory information
- Add auxiliary classifiers and the total loss is a weighted sum of all of them



Inception v2, v3 (2016)

- Use batch normalization!
- Auxiliary layers not really helping to push useful gradients into earlier layers
- Use factorized filters: e.g. $5 \times 5 \times c$ needs $25c$ params, two $3 \times 3 \times c$ needs $18c$ params

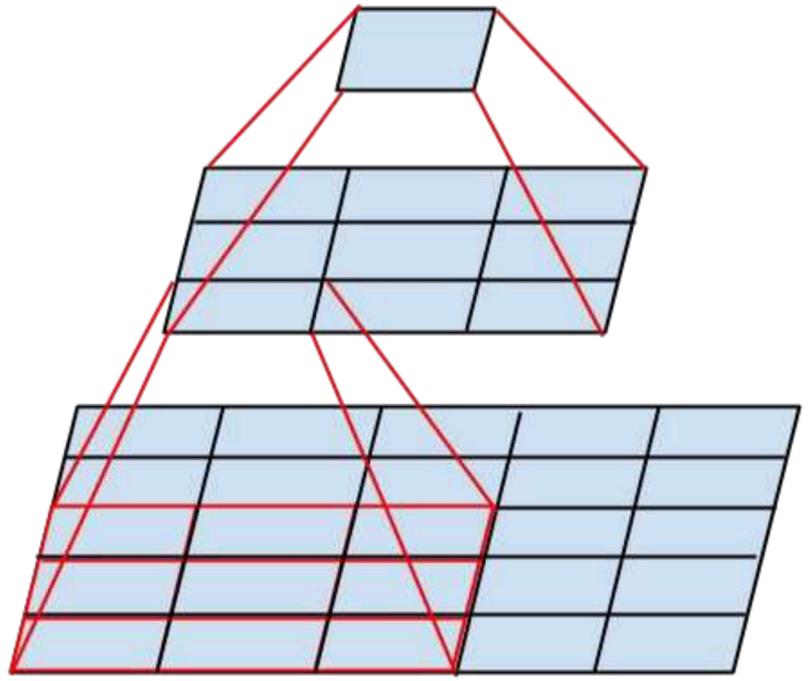
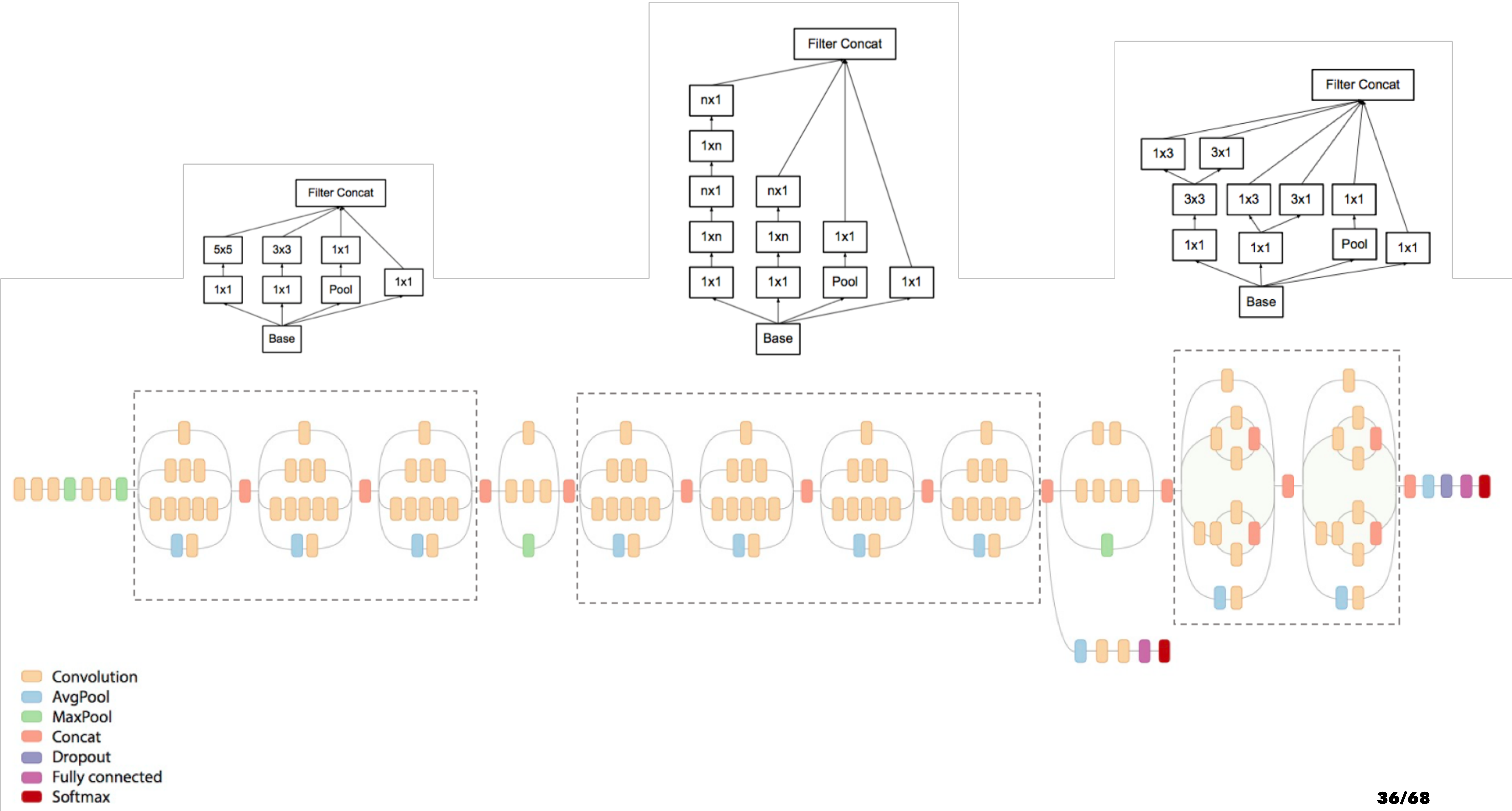
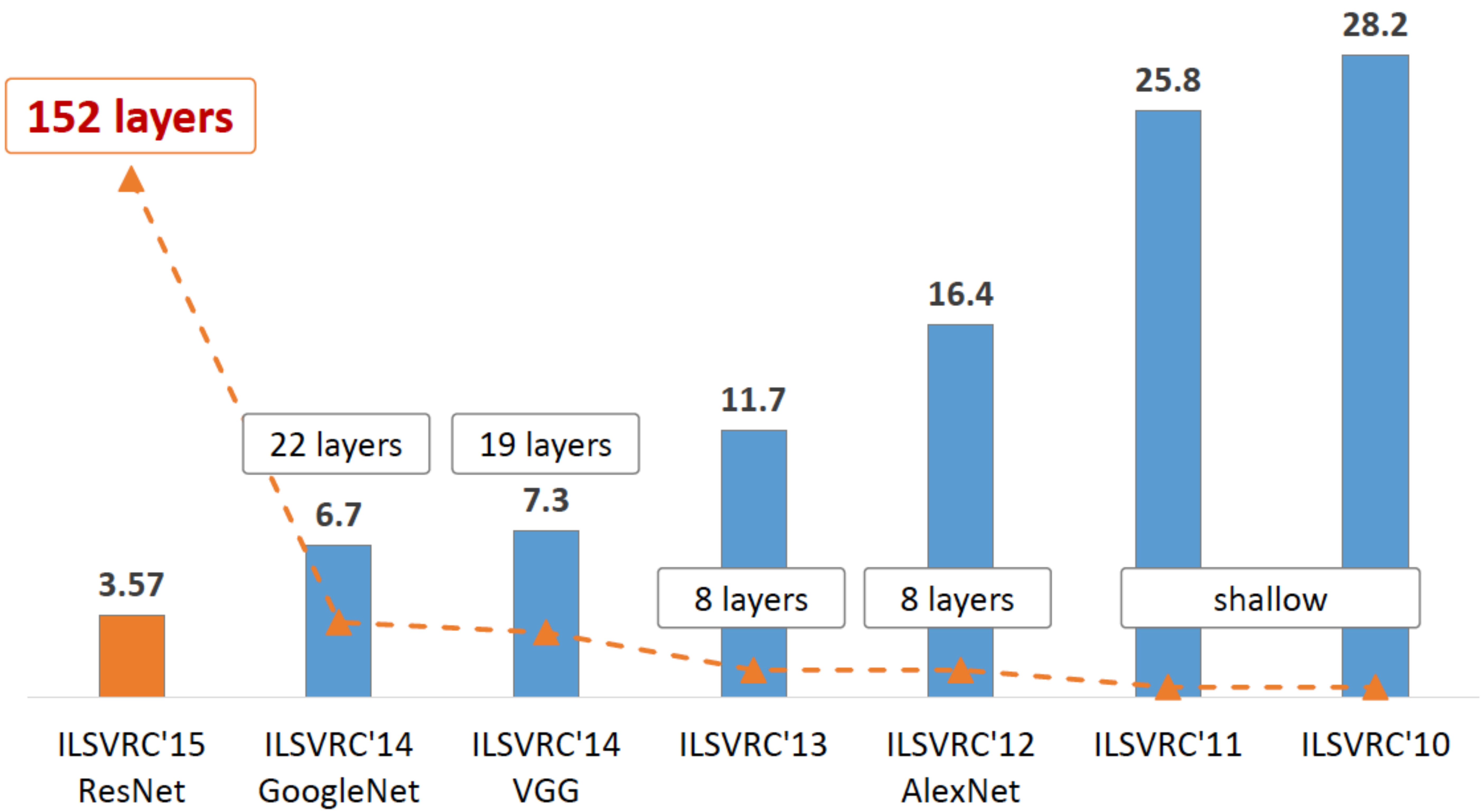


Figure 1. Mini-network replacing the 5×5 convolutions.



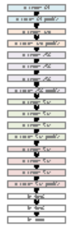
Filter Concat
5x5
3x3
1x1
1x1
1x1
Pool
1x1
Base
Filter Concat
nx1
1xn
nx1
nx1
1xn
1xn
1x1
1x1
1x1
Pool
1x1
Base
Filter Concat
1x3
3x1
3x3
1x3
3x1
1x1
1x1
1x1
Pool
1x1
Base
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax



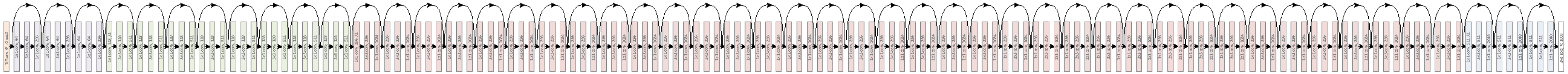
ImageNet Classification top-5 error (%)

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)



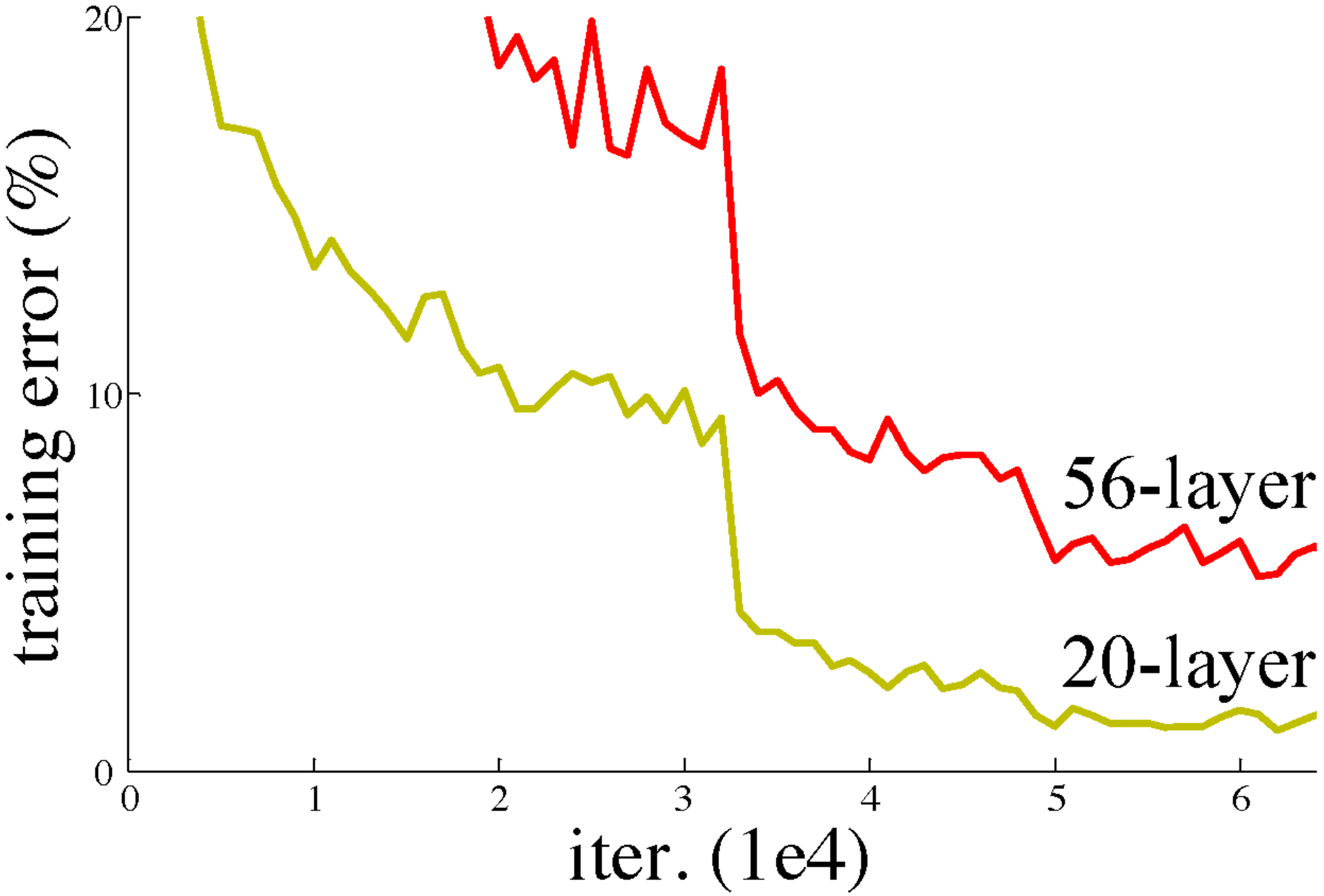
GoogleNet, 22 layers
(ILSVRC 2014)



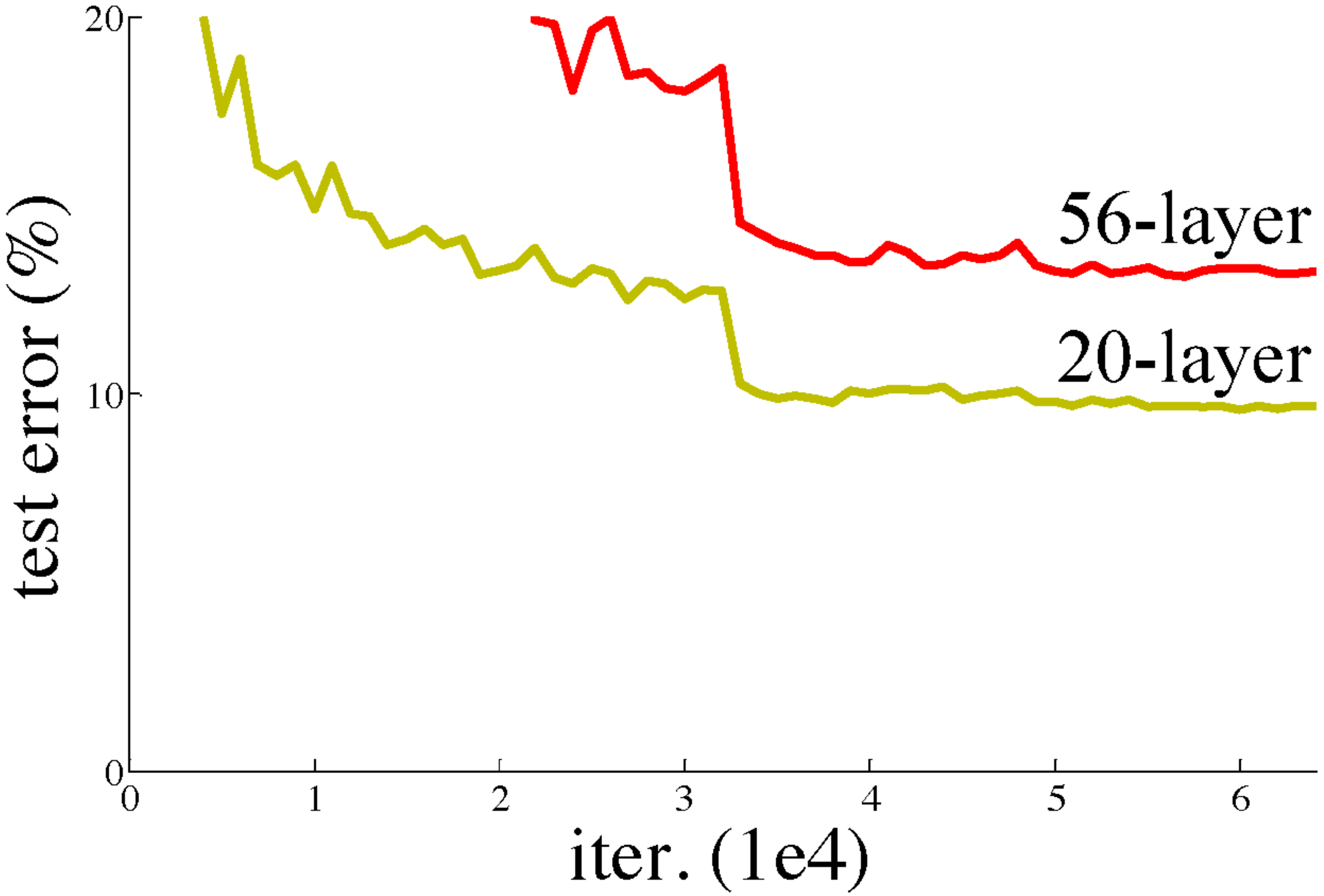
ResNet, 152 layers
(ILSVRC 2015)

Is learning better networks as easy as stacking more layers?

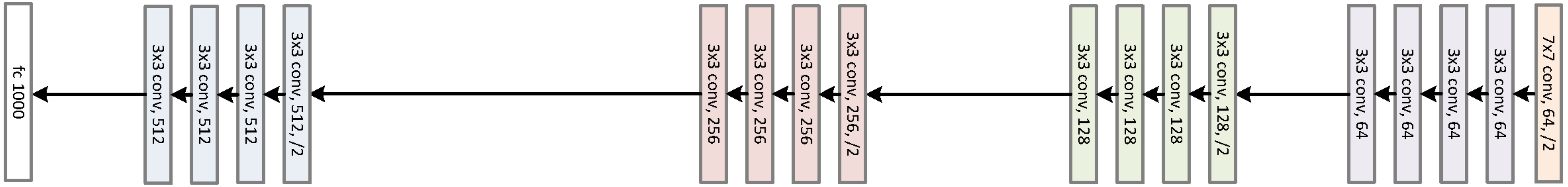
🤔 WTF!



training error (%)
20
10
0
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer

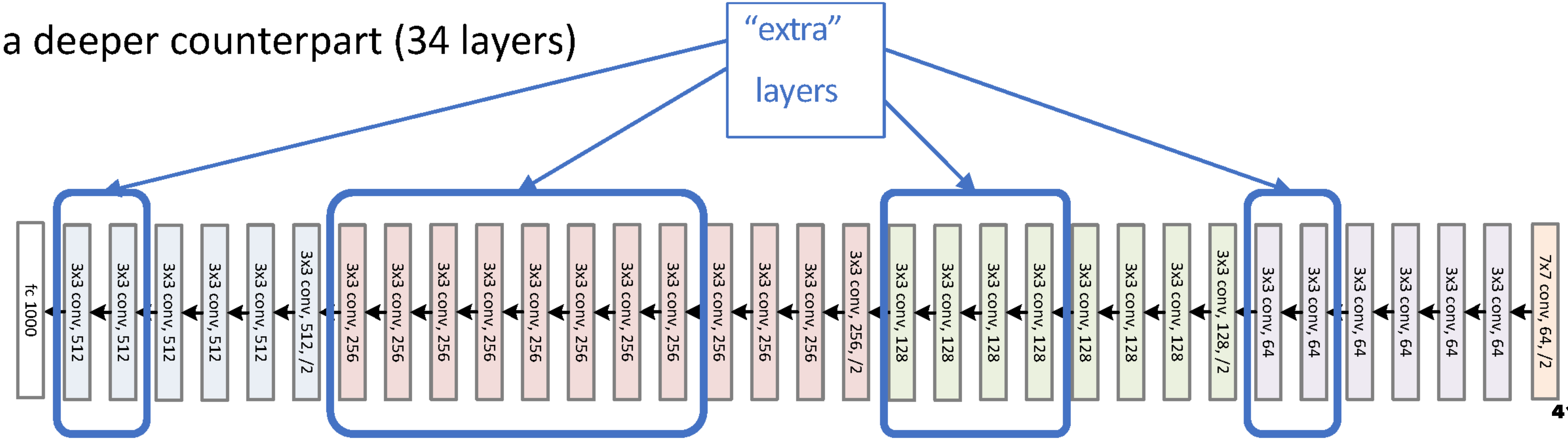


test error (%)
20
10
0
0
1
2
3
4
5
6
iter. (1e4)
56-layer
20-layer



a shallower model (18 layers)

a deeper counterpart (34 layers)



Plain Network

```
      ┌──      any two stacked layers      ──┐

      ┌───────────┐           ┌───────────┐
      │   weight  │           │   weight  │
──x─> │layer + BN │ ──ReLU─>  │layer + BN │──ReLU─>
      └───────────┘           └───────────┘

            desired mapping: H(x)
```

Residual Network (2015)

```
   ┌────────────identity shortcut (x)──────────────┐
   │                                                │
   │                                                ▼
   │  ┌──  ·   ·   · F(x) ·   ·   ·  ──┐
   │
   │  ┌────────────┐        ┌────────────┐       ┌──┐
   │  │   weight   │        │   weight   │       │+ │
─x─┴─>│layer + BN  │ ─ReLU─>│layer + BN  │  ───> │  │─ReLU─>
      └────────────┘        └────────────┘       └──┘
```

H(x) = F(x) + x

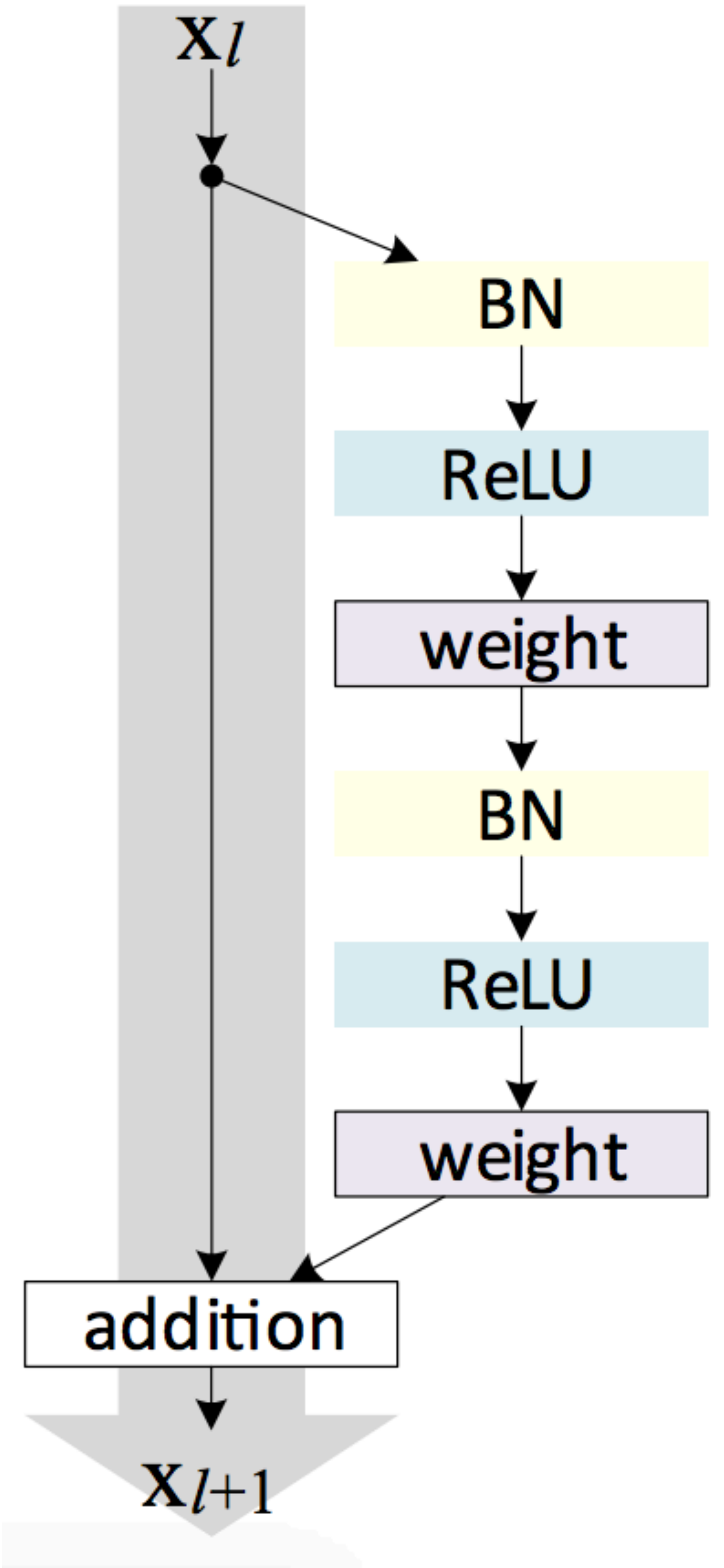
or F(x) = H(x) − x

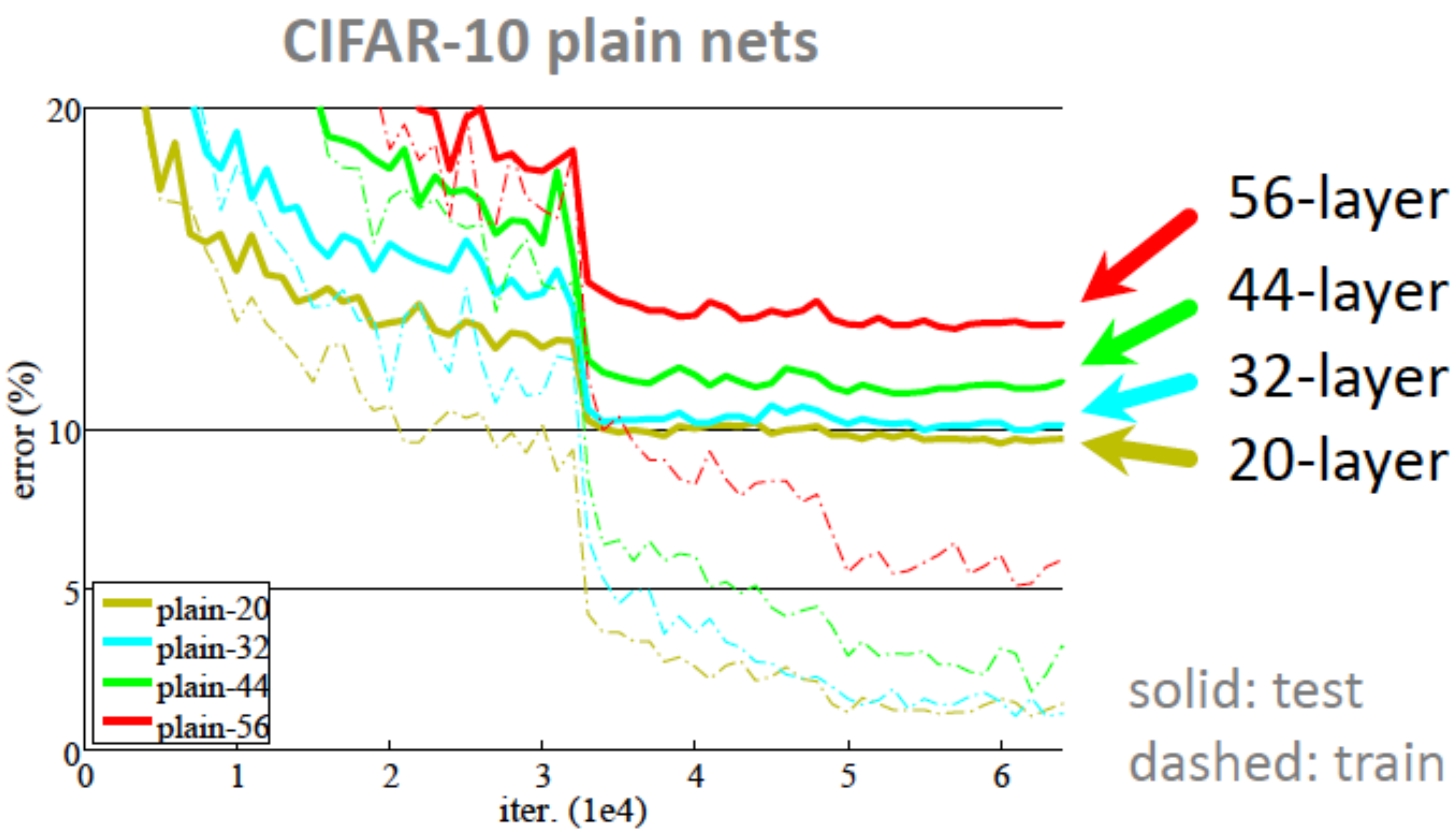
Shortcut Connections

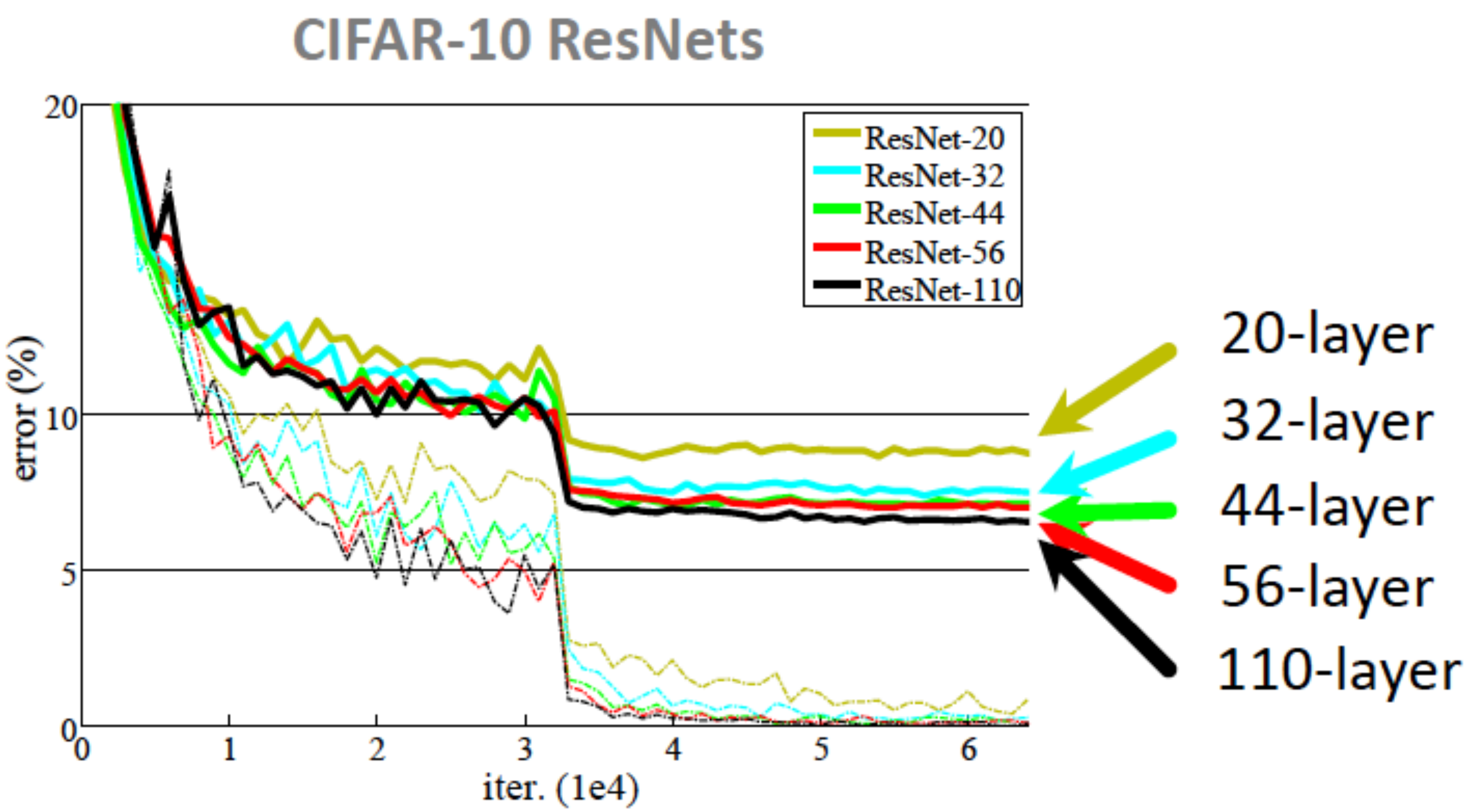
- Add a linear layer connected from the network input to the output by Ripley 1996
- A few intermediate layers are directly connected to auxiliary classifiers by Lee et al. 2014 or Szegedy et al. 2015
- "Inception" layer composed of a shortcut branch and a few deeper branches by Szegedy et al. 2015
- Highway networks: shortcuts with gating functions by Schmidhuber et al. 2015

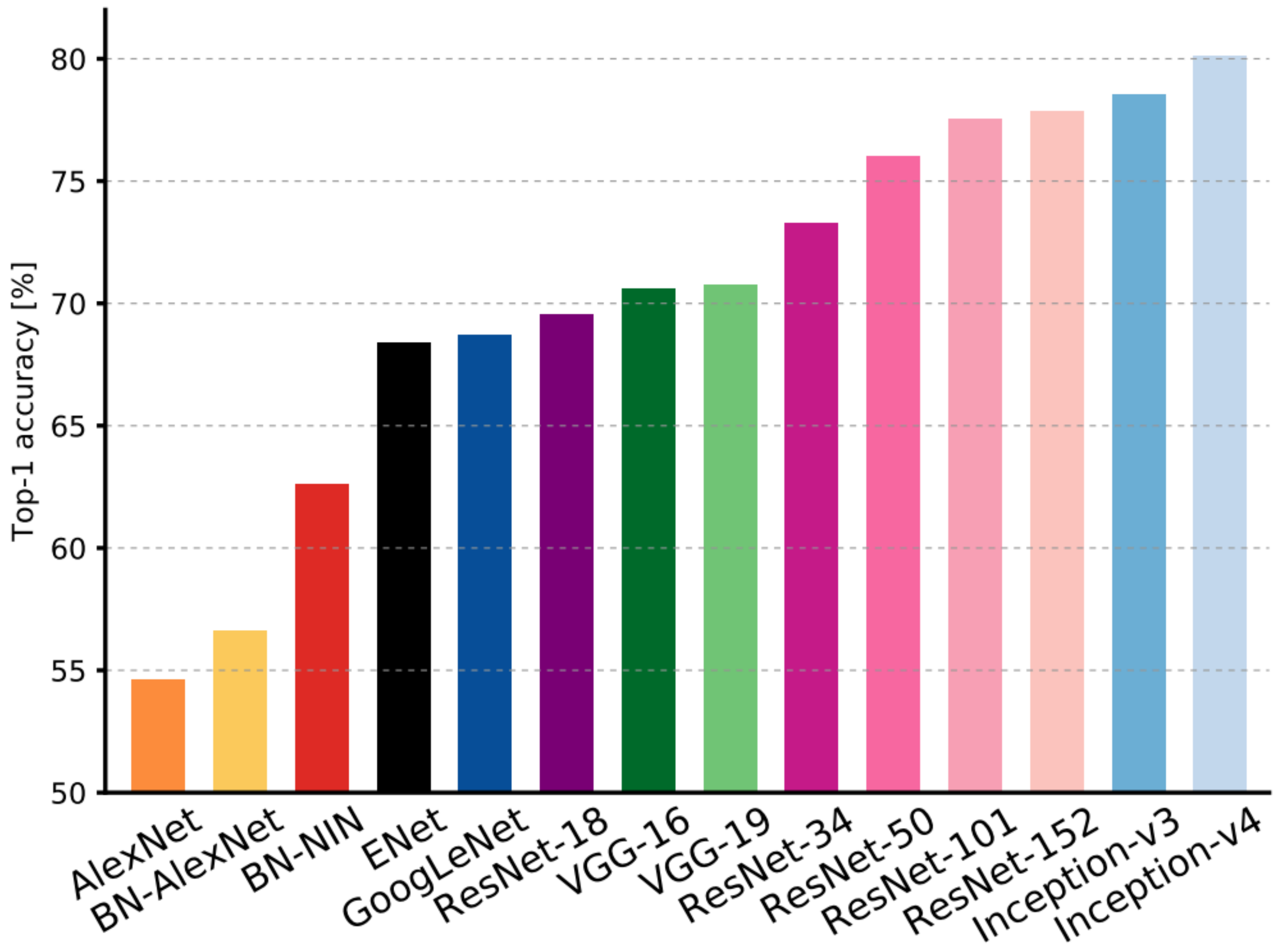
New ResNet (2016)

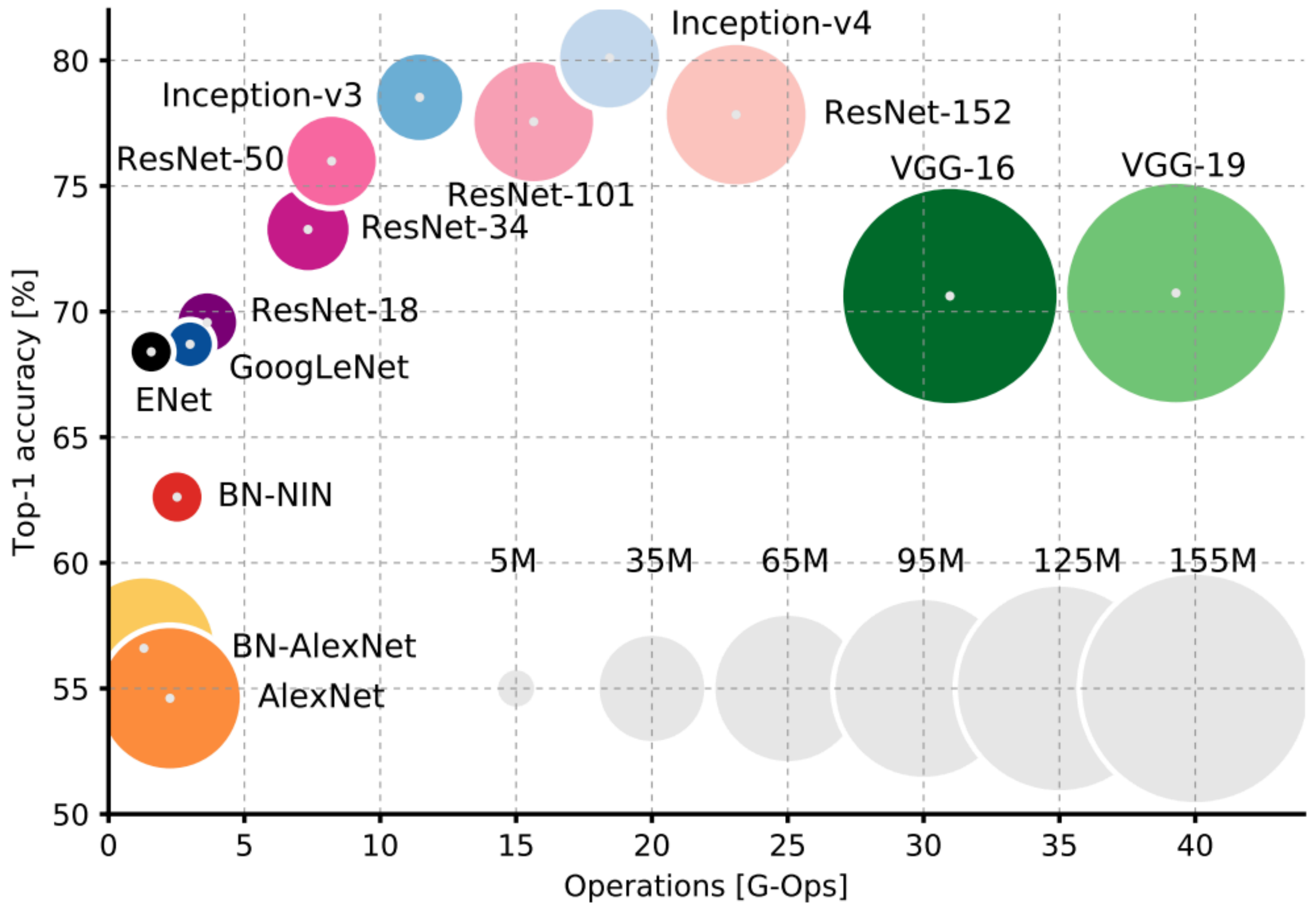
- **Deep** and VGG-Style:
  - All convolutions proceeded by Batch Normalization and ReLU
  - When spatial size /2 then increase number of filters x2
  - Xavier2 initialization
  - SGD + Momentum (0.9)
- No: Max pooling, hidden fully connected layers or Dropout




CIFAR-10 plain nets
20
10
5
0
error (%)
0 1 2 3 4 5 6
iter. (1e4)
plain-20
plain-32
plain-44
plain-56
56-layer
44-layer
32-layer
20-layer
solid: test
dashed: train


CIFAR-10 ResNets
20
10
5
0
error (%)
0 1 2 3 4 5 6
iter. (1e4)
ResNet-20
ResNet-32
ResNet-44
ResNet-56
ResNet-110
20-layer
32-layer
44-layer
56-layer
110-layer


Top-1 accuracy [%]
80
75
70
65
60
55
50
AlexNet
BN-AlexNet
BN-NIN
ENet
GoogLeNet
ResNet-18
VGG-16
VGG-19
ResNet-34
ResNet-50
ResNet-101
ResNet-152
Inception-v3
Inception-v4


Top-1 accuracy [%]
Inception-v4
Inception-v3
ResNet-152
ResNet-50
ResNet-101
ResNet-34
VGG-16
VGG-19
ResNet-18
GoogLeNet
ENet
BN-NIN
5M
35M
65M
95M
125M
155M
BN-AlexNet
AlexNet
0
5
10
15
20
25
30
35
40
Operations [G-Ops]

Other architectures

- Wide Residual networks (2016)
- ResNeXt (2016)
- Stochastic depth (2016)
- FractalNet (2017)
- Densely connected convolutional networks (2017)
- SqueezeNet (2017)
- Squeeze-and-excitation networks (2017)

Squeeze and excitation networks

- Calculate per feature (or channel) statistics (squeeze)
- Pass this through a 2 layer model with weights W that predicts weights for each channel
- Scale the input channels by this weighting
- Intuition: a gating mechanism for the network to select the most important channels

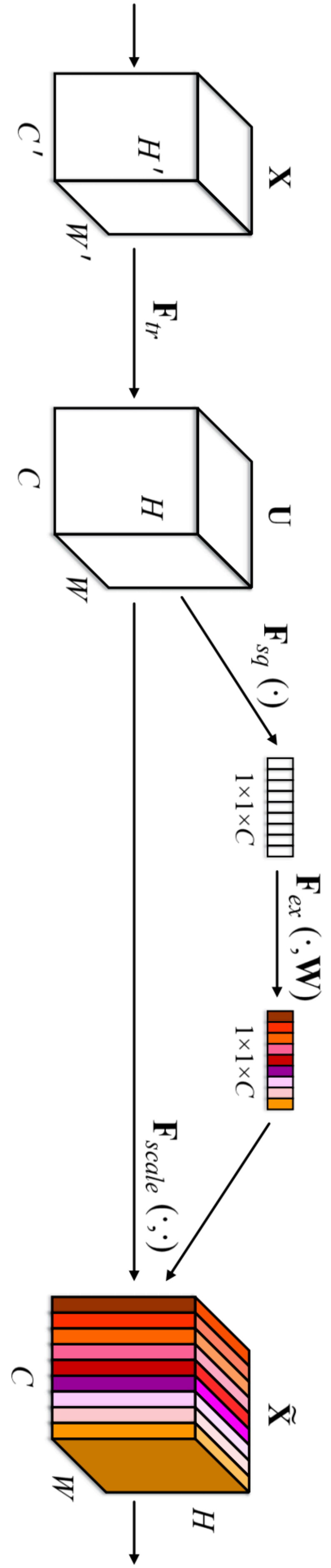
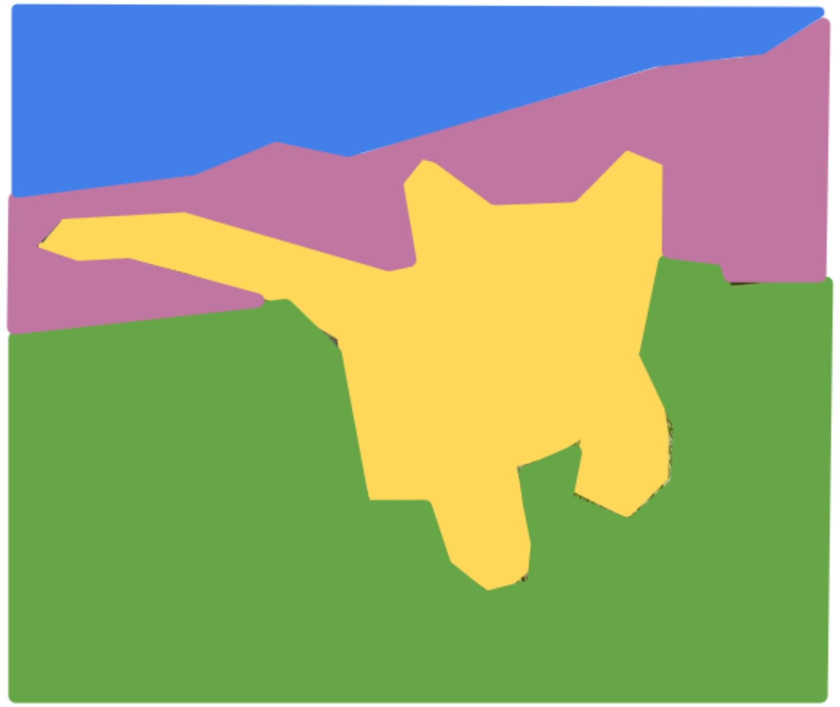


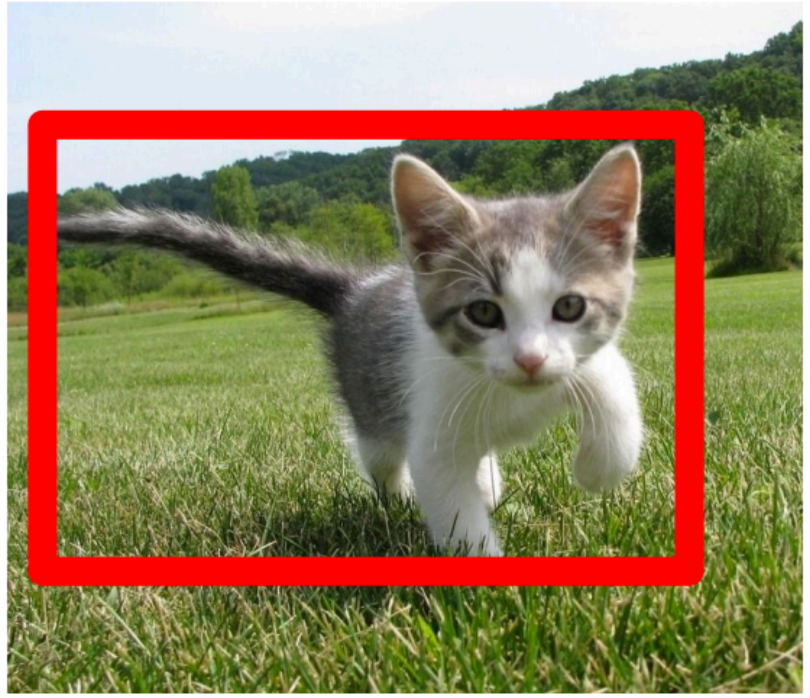
Figure 1. A Squeeze-and-Excitation block.

Semantic Segmentation



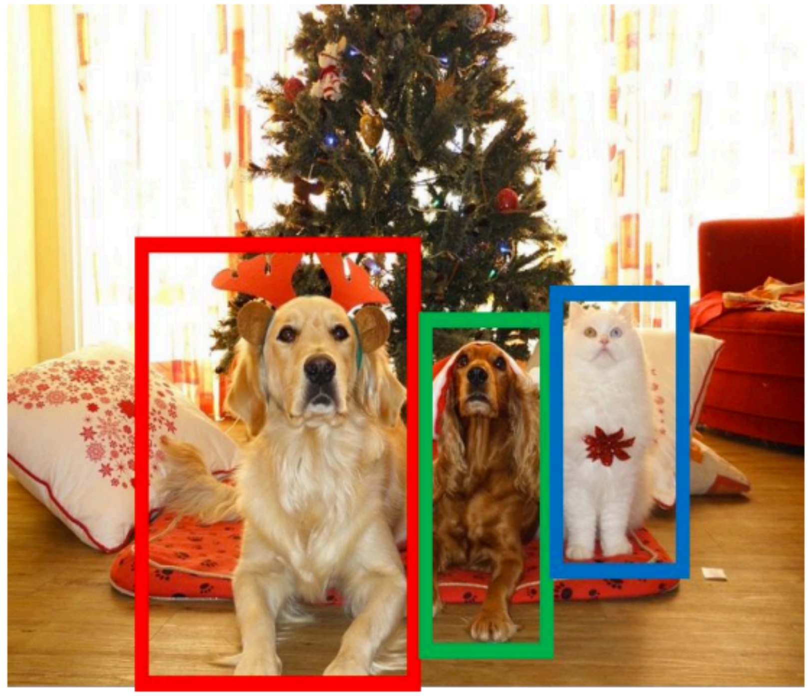
GRASS, CAT, TREE, SKY

Classification + Localization



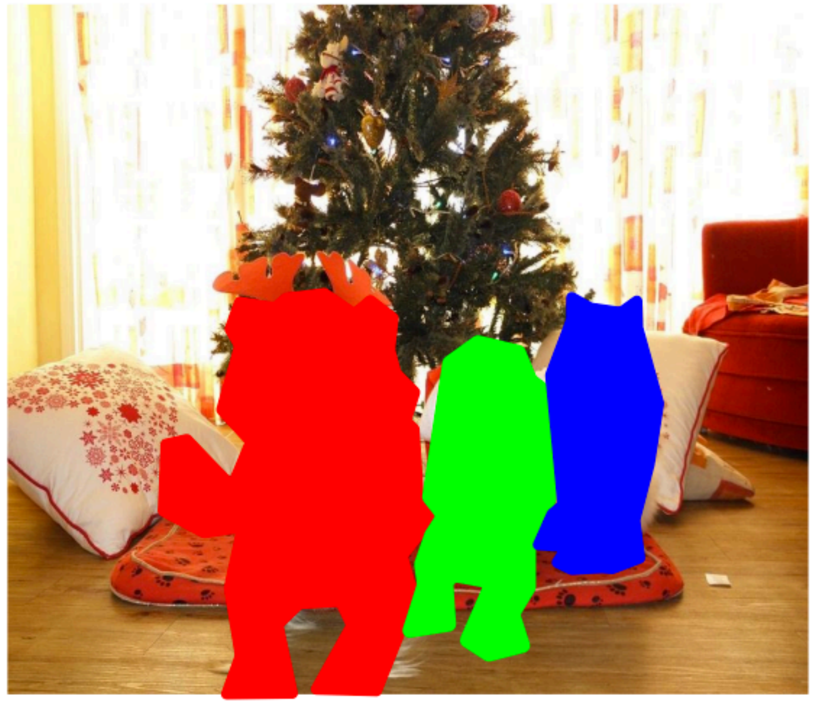
CAT

Object Detection



DOG, DOG, CAT

Instance Segmentation



DOG, DOG, CAT

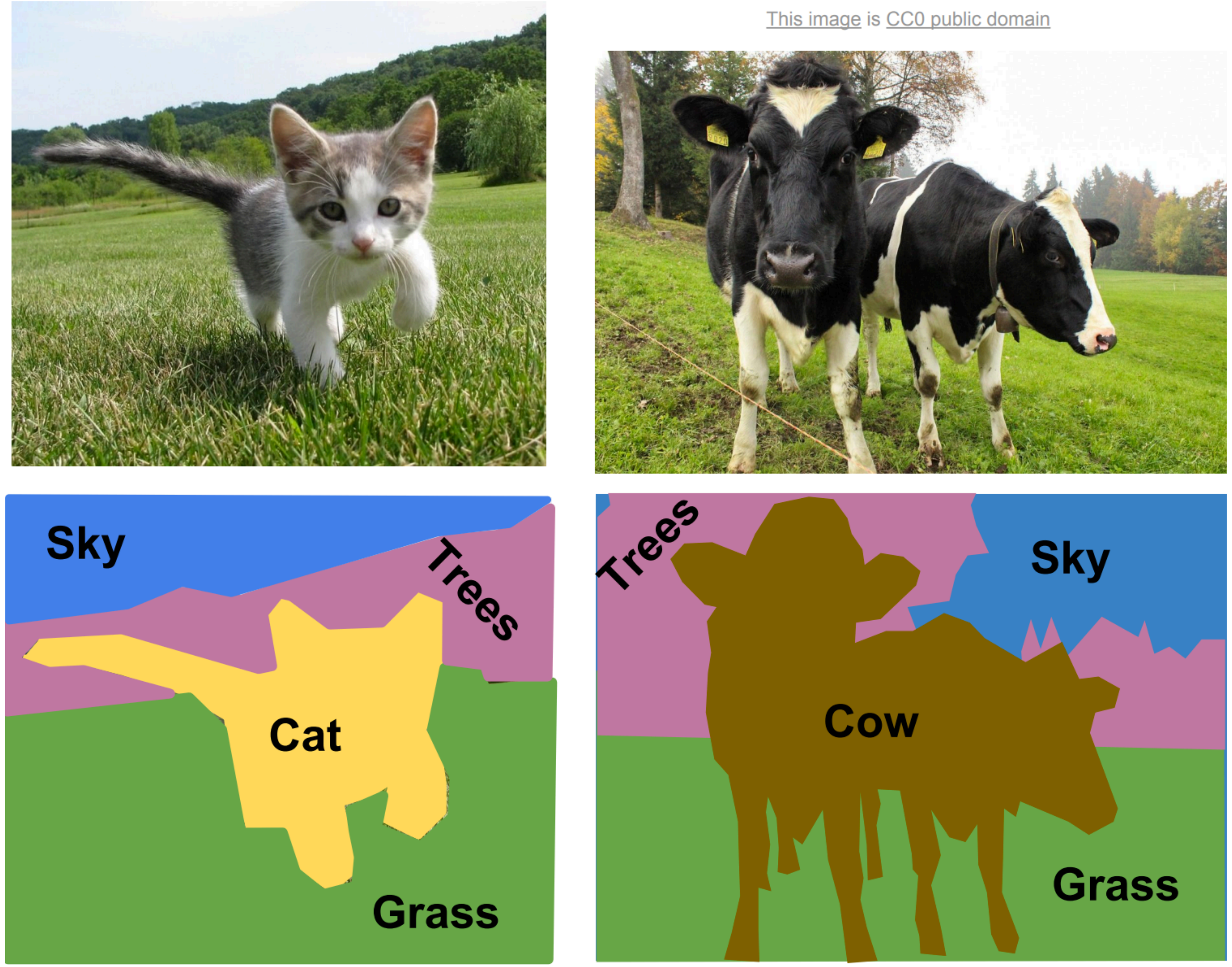
No objects, just pixels

Single Object

Multiple Object

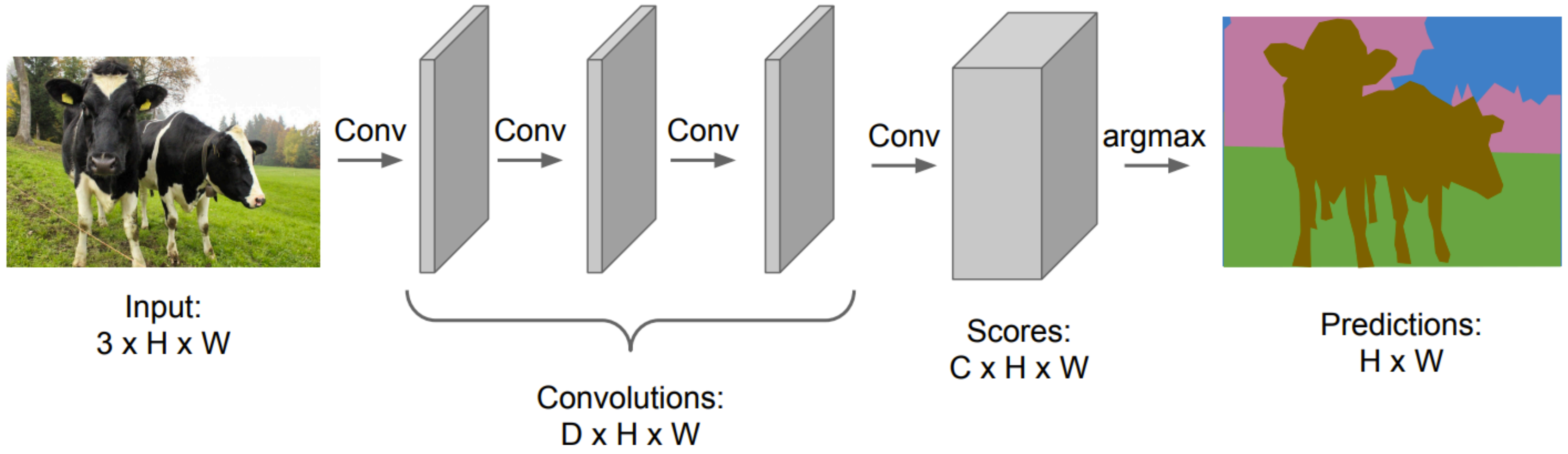
Semantic Segmentation

- Label each pixel with a category label
- Only care about pixels, no objects
    - Both cows are labeled as a cow blob
- Naive idea: sliding window



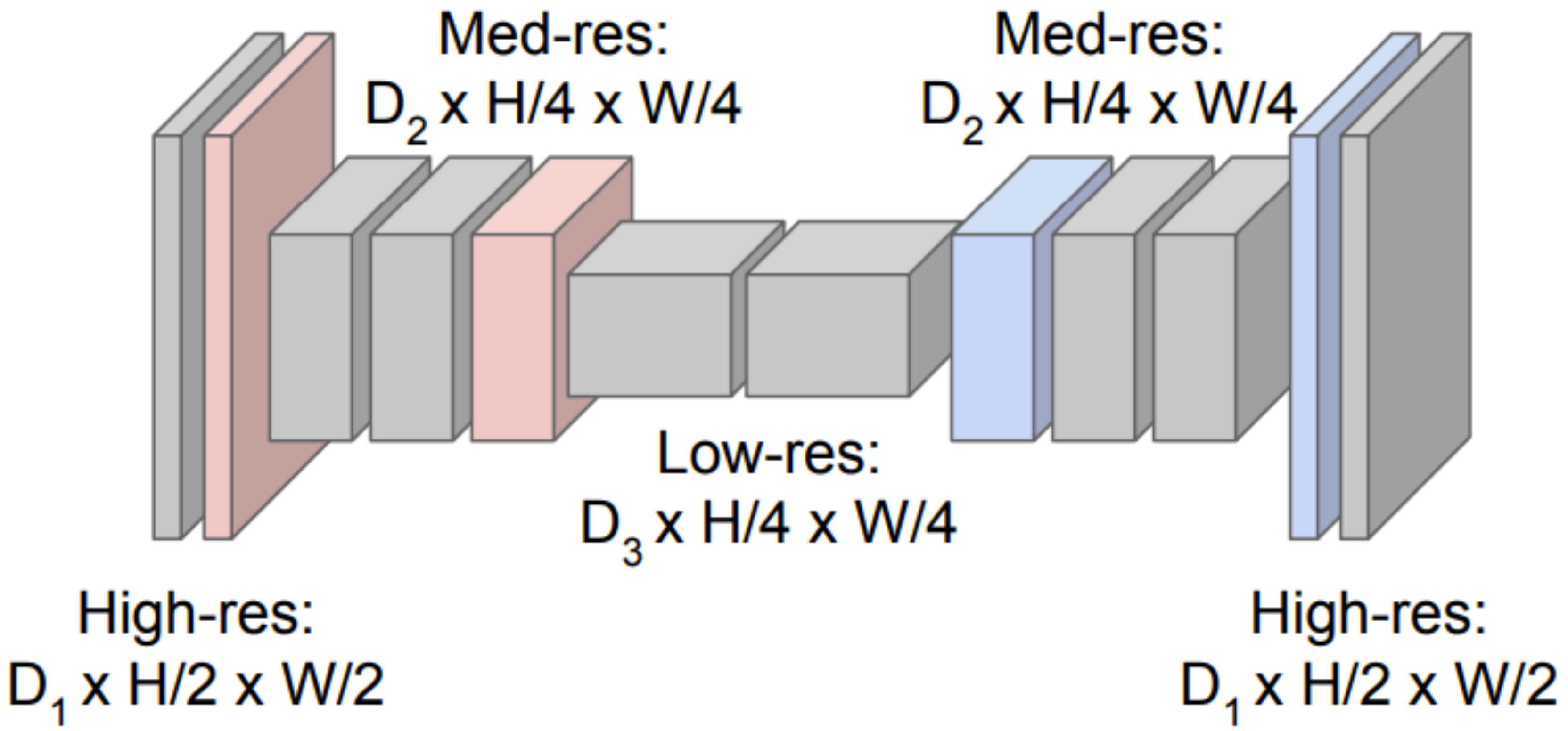
Another Idea?

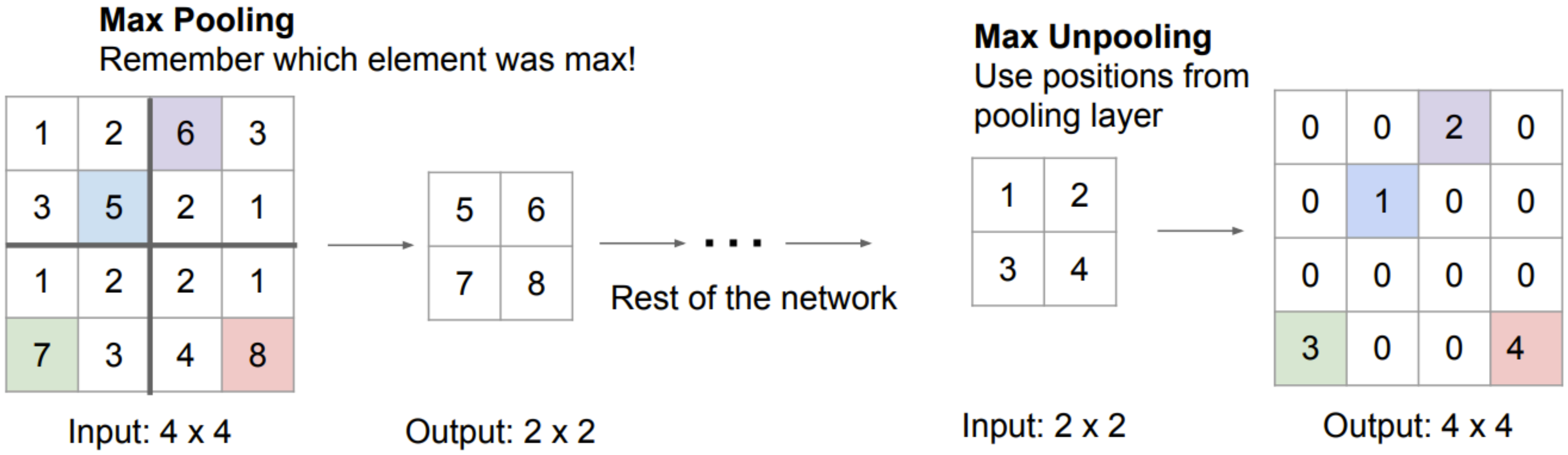
- Design a Conv Net to make predictions for all pixels at once
- Problem: convolutions at original image resolution will be very expensive



Deconvolution Networks (2015)

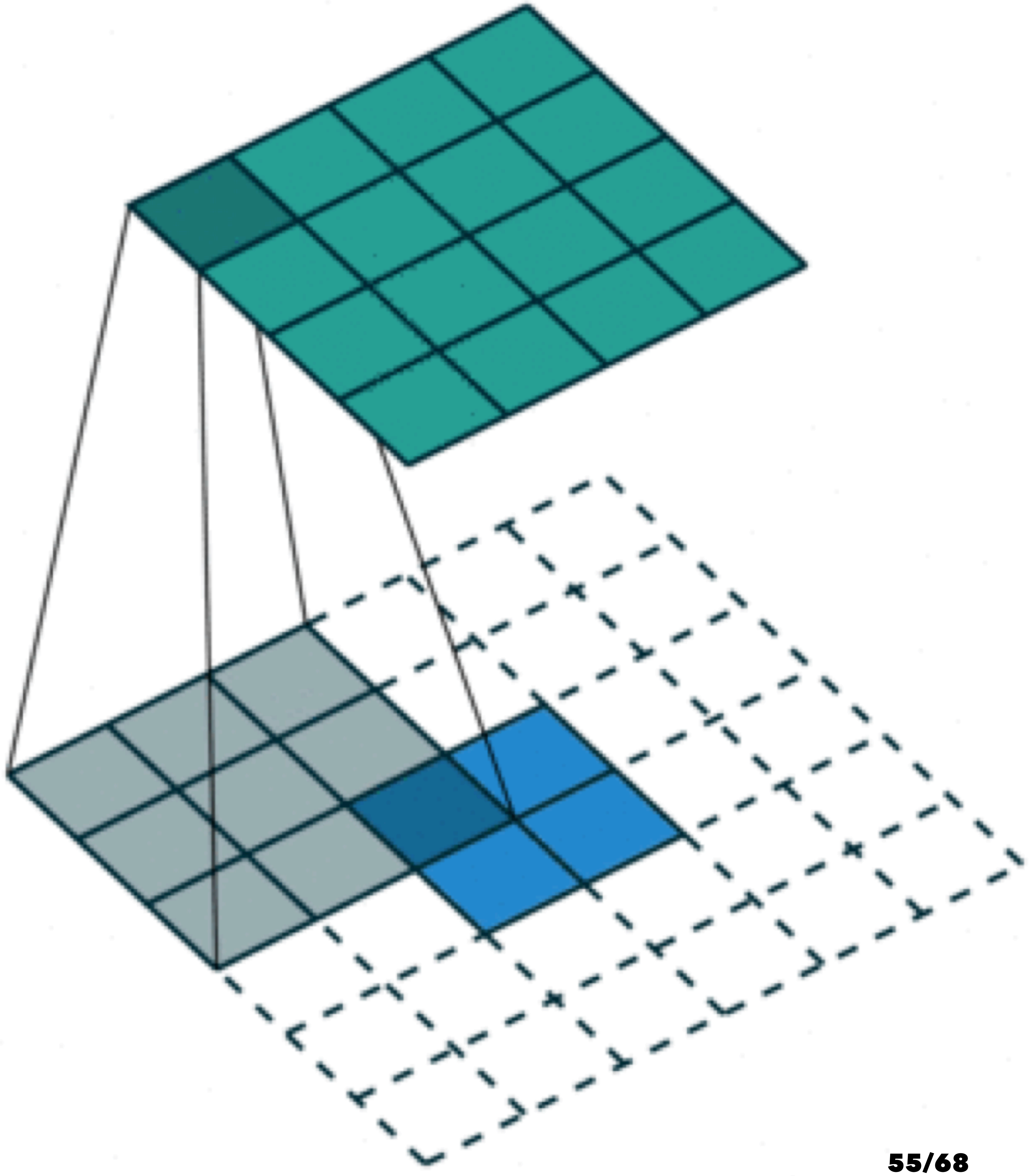
- Main idea: use down-sampling and up-sampling
- Down-sample: pooling, strided convolutions
- Up-sampling: ??



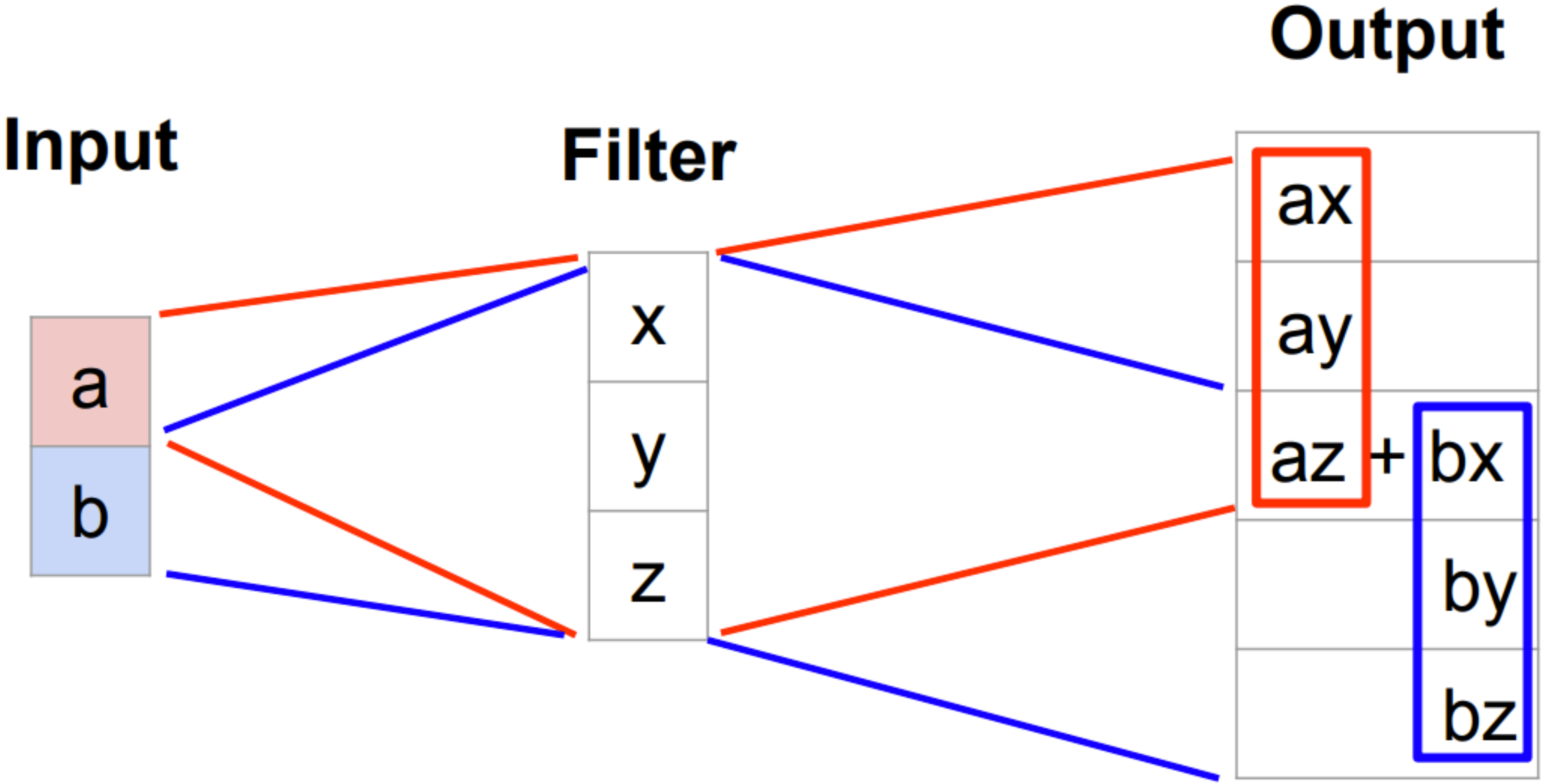

Max Pooling
Remember which element was max!
1 2 6 3
3 5 2 1
1 2 2 1
7 3 4 8
Input: 4 x 4
5 6
7 8
Output: 2 x 2
...
Rest of the network
Max Unpooling
Use positions from pooling layer
1 2
3 4
Input: 2 x 2
0 0 2 0
0 1 0 0
0 0 0 0
3 0 0 4
Output: 4 x 4

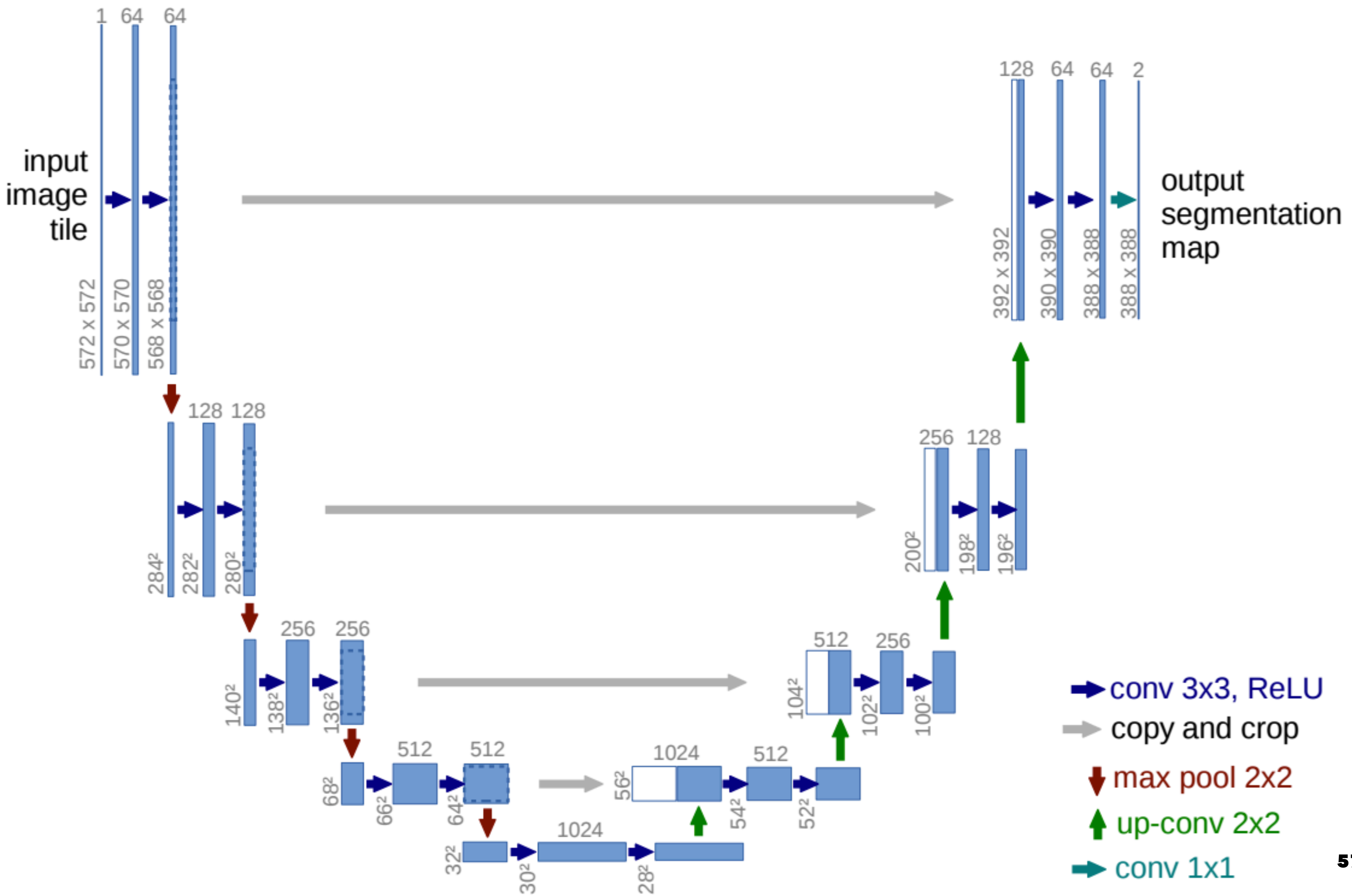
Transpose Convolution*

- Output is the filter weighted by the input
- At each overlap we sum to get output



* Other names: Deconvolution (bad), Upconvolution, Fractionally strided convolution or Backward strided convolution

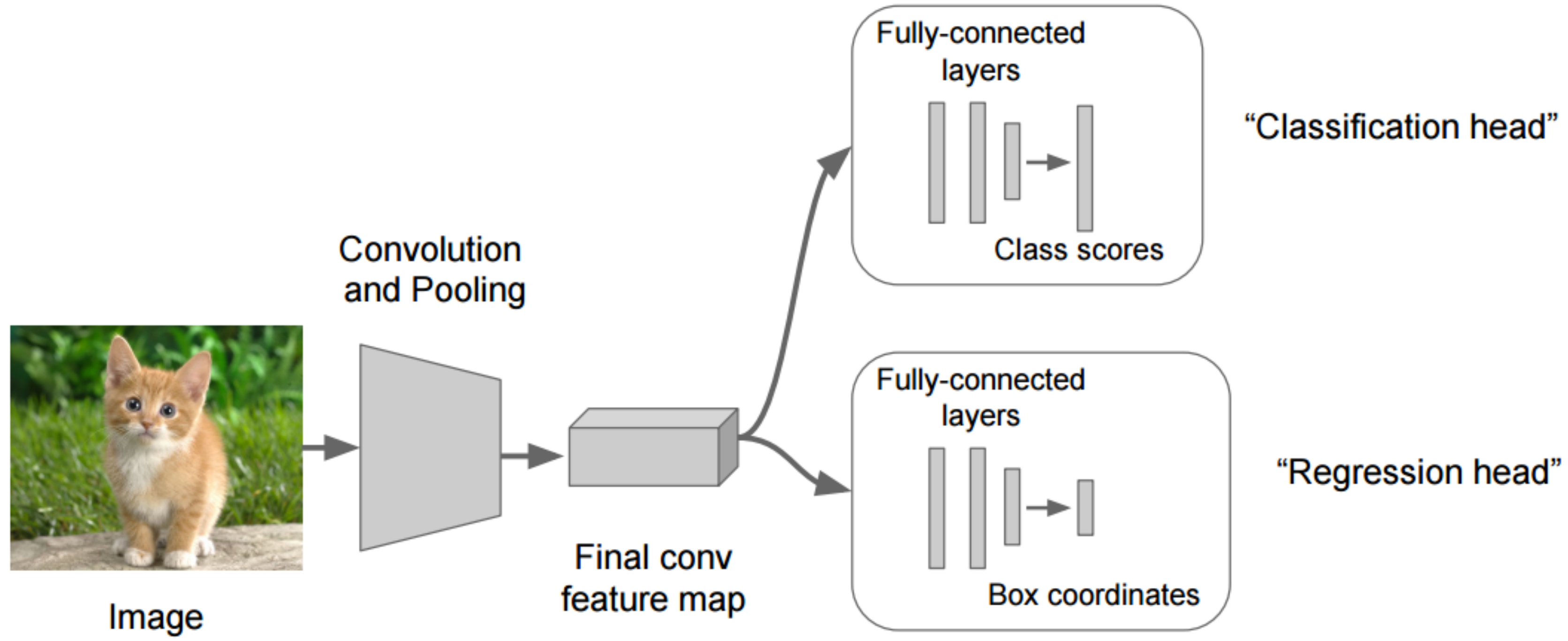

Output
Input
Filter
a
b
x
y
z
ax
ay
az
+
bx
by
bz



input image tile
output segmentation map
1
64
64
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

Localization

- Model must predict:
  - bounding box
  - label
- Idea: treat localization as a regression problem
- Generalizes to localizing exactly K objects

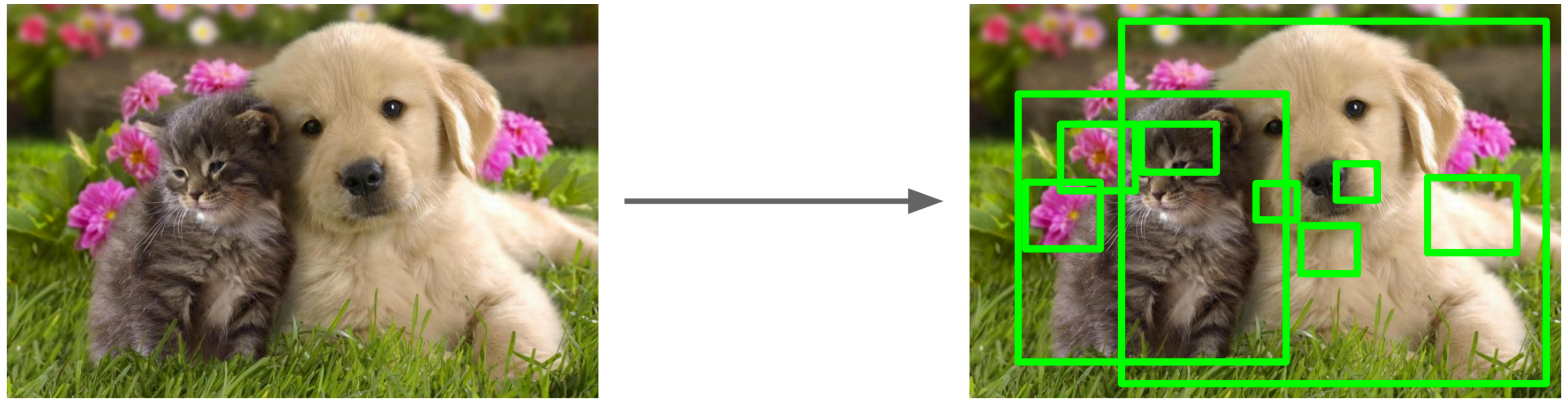


Object Detection

- Detect all instances from a set of classes in input image
- Bounding boxes of these instances
- Cannot use regression: since we have variable sized outputs
- Could apply a CNN to many different crops to predict class or background
    - Problem: again very computationally expensive

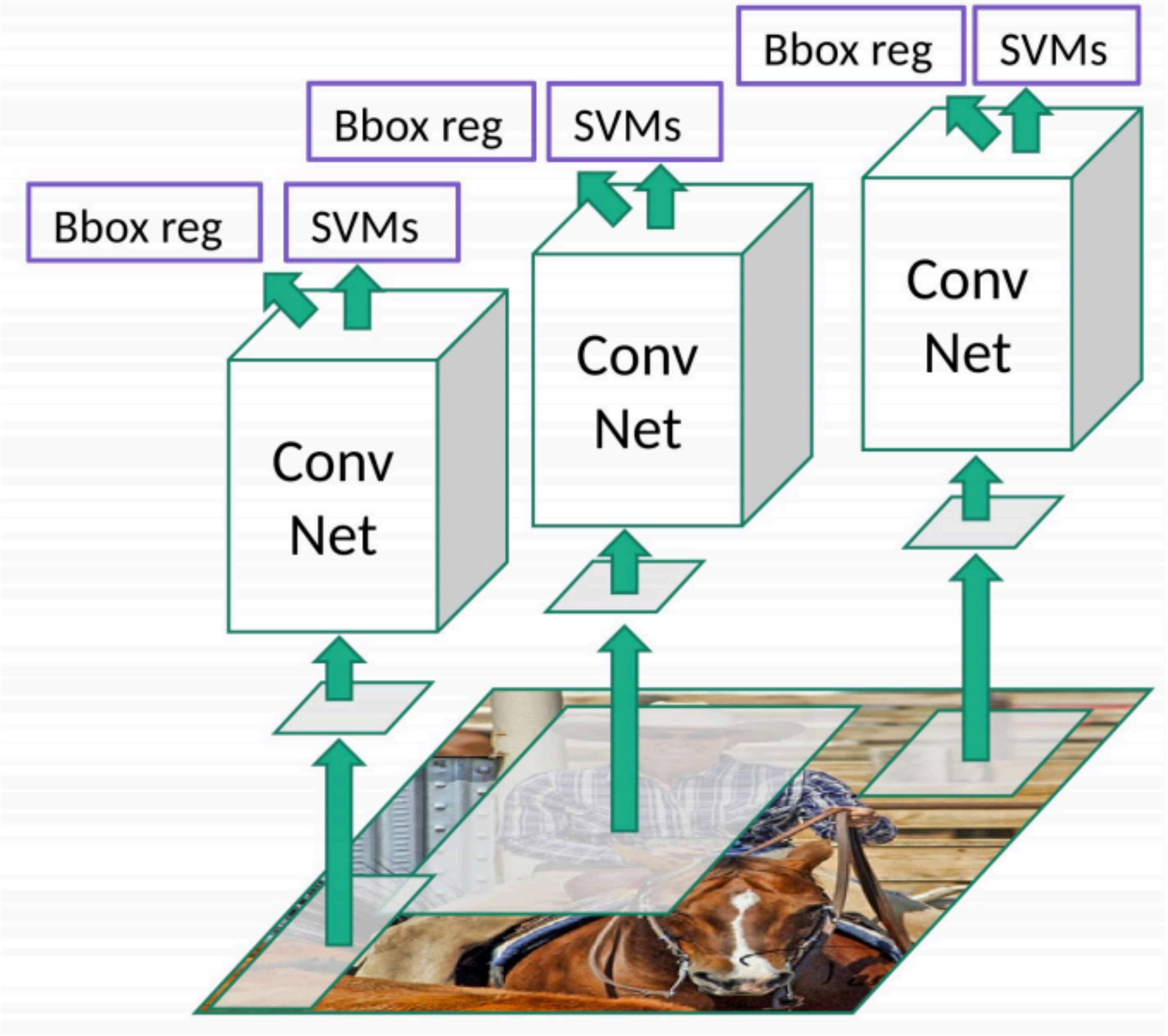
Region Proposals

- Find regions which might contain objects
- Relatively fast: Selective Search (2012) gives 2,000 region proposals in a few seconds
- Uses standard computer vision techniques (no deep learning)



R-CNN (2014)

- Run region proposal to get ROIs
- Wrap ROIs to fixed square size for CNN
- Run each through CNN
- Classify region via SVM and bounding box via linear regression

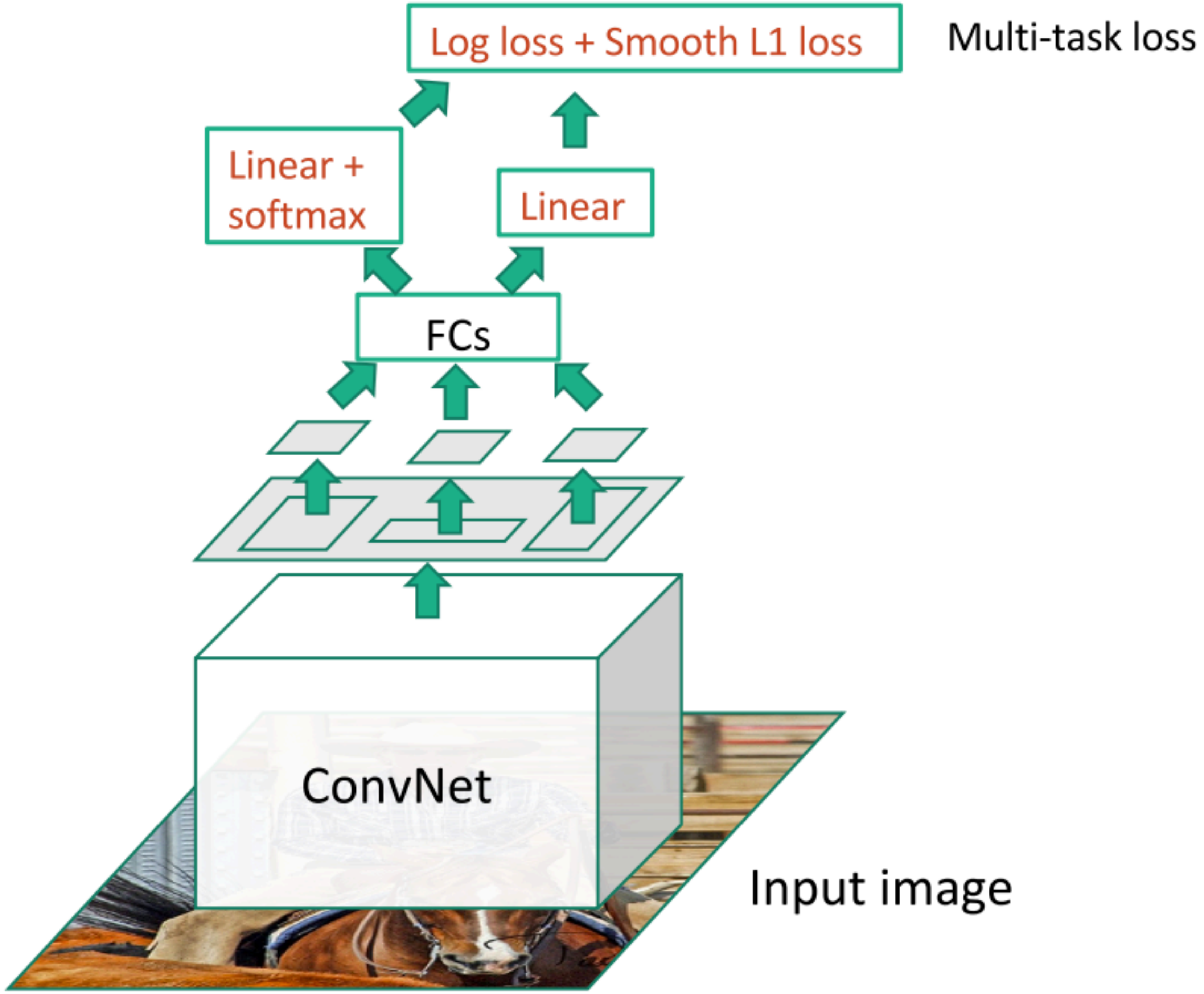


R-CNN Issues

- Ad hoc training objectives:
  - fine tune network with softmax classifier
  - train post-hoc linear SVMs
  - train post-hoc bounding-box regressions
- Training is slow and takes a lot of disk space
- Inference is also slow

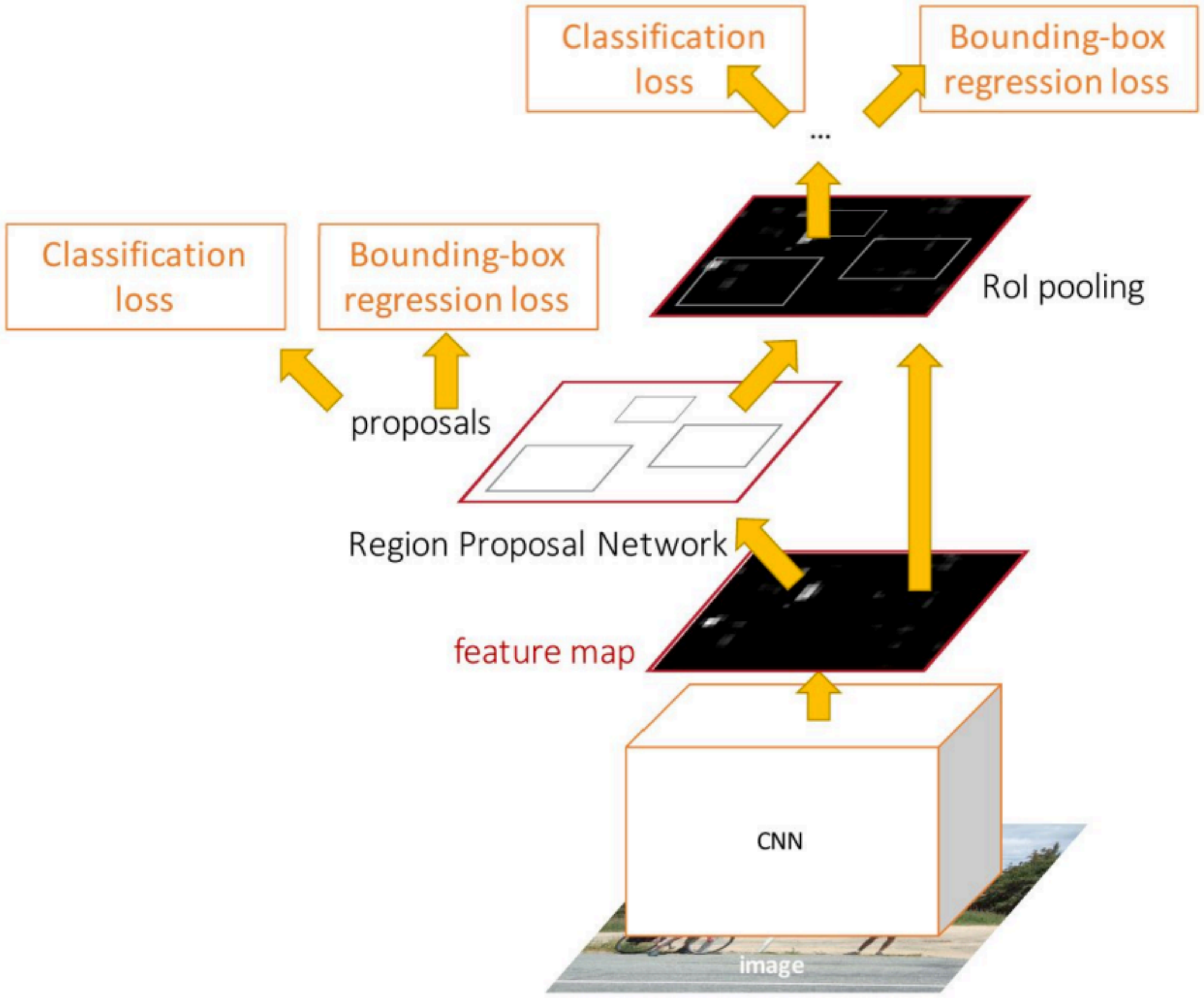
Fast R-CNN (2015)

- Forward whole image through Conv Net
- At some convolutional feature map, project the ROIs
- Do ROI pooling to wrap these regions for fully connected layers
- Use softmax and regressor together with multi-task loss
- Fast R-CNN 10x faster to train and inference less than a second per image



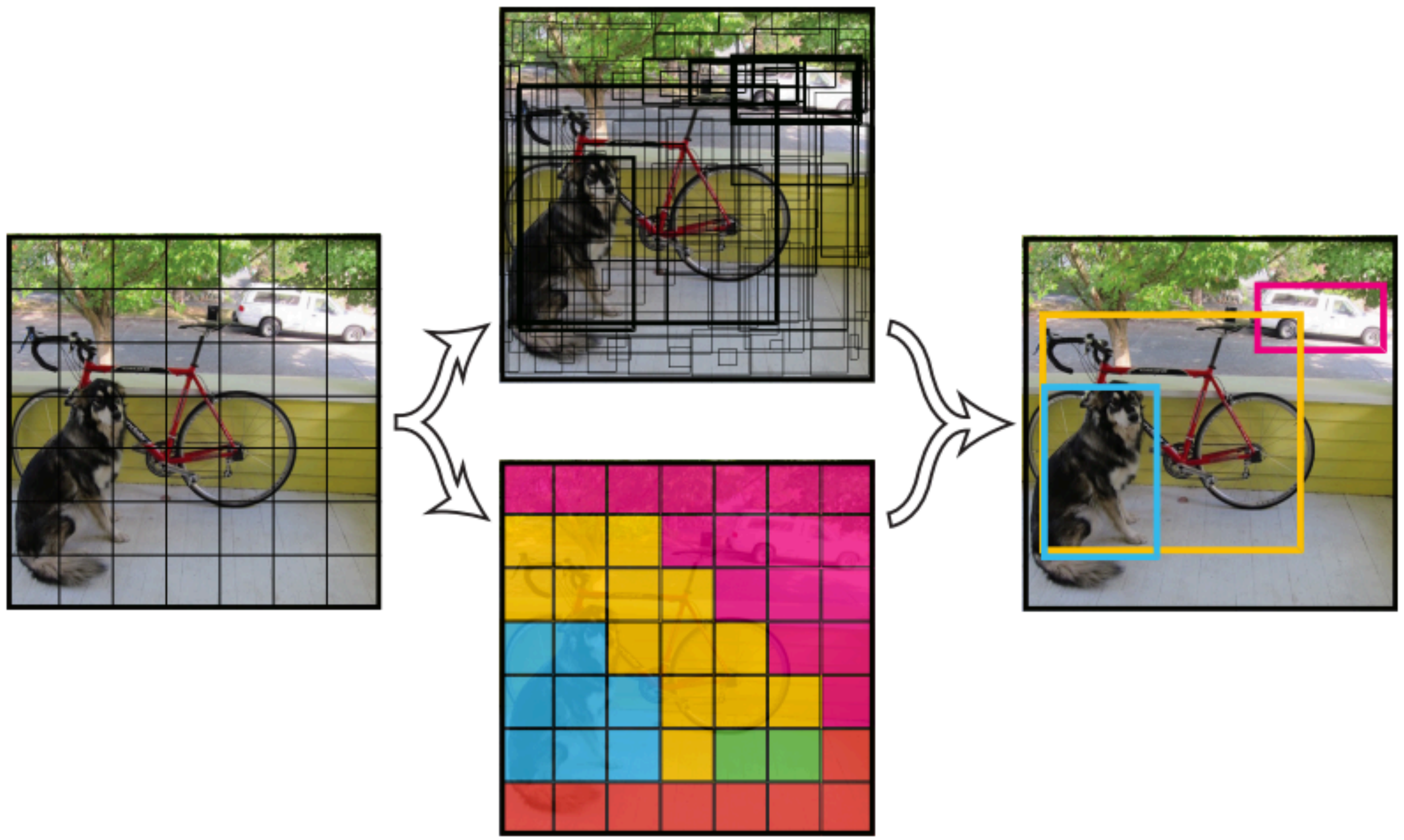
Faster R-CNN (2015)

- Insert Region Proposal Network (RPN) to predict proposals from feature map
- Jointly train:
  1. RPN to classify object / not object
  2. RPN regression box coordinates
  3. Final classification score (object classes)
  4. Final box coordinates correction



YOLO / SSD (2016)

- Divide image into even grid
- Centered in each grid create B base boxes
- Within each grid:
  - regress from each of the B boxes to a final box
  - predict score for each of C classes (including background)



Object Detection Trade-offs (2017)

- Base networks: VGG, ResNet, etc.
- Architecture: Faster R-CNN, SSD, R-FCN, etc.
- Image size
- Number of region proposals
- Takeaways: Faster R-CNN slower but more accurate, SSD faster but not as accurate

Instance Segmentation

- Want to detect all instances
- Predict a pixel mask for each instance detected

Mask R-CNN (2017)

- Looks like Faster R-CNN: full image goes to convolution network to learn ROI proposals
- One branch makes a classification and bounding box predictions of ROI
- Other branch goes to another CNN to predict the pixel masks for each of C classes