Convolutional Neural Networks

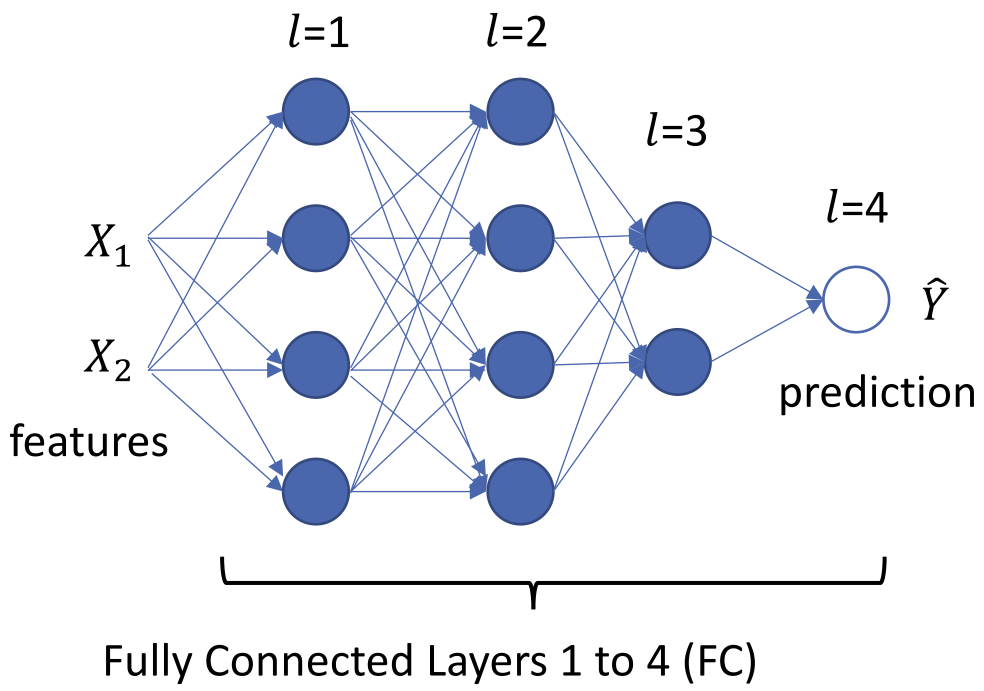
Ulrich Finkler, Wei Zhang

CSCI-GA.3033-022 HPML

Convolutional Neural Networks

CNN - Motivations

- **Fully-Connected Neural Networks** for image recognition/classification:
  1. **Dimension Problem**:
     - Reshaping to 1D: loosing the spatial structure
  2. **Size Problem:**
     - Example features: 1000 × 1000 × 3 [Y × X × RGB]
     - Weights: 3M just for first layer
     - 100 layers: about 300M DP weights => **2.4GB model size**
  3. **Overfitting problem:**
     - One weight per pixel leads to **overfitting**



Convolution Example – Vertical Edge Detection

6 x 6 image

8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0

dot

3 x 3 filter

1	0	-1
1	0	-1
1	0	-1

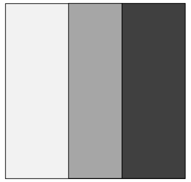
=

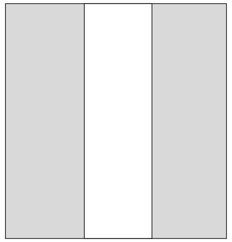
4 x 4 image

0	24	24	0
0	24	24	0
0	24	24	0
0	24	24	0

$(8 \times 1 + 8 \times 0 + 8 \times (-1)) \times 3 = 0$

$(8 \times 1 + 8 \times 0 + 0 \times (-1)) \times 3 = 24$





Padding

- Convolution
  - Image size 6 × 6 (n = 6)
  - Filter size f = 3 × 3 (f = 3)
  - Output: 4 × 4 (n' = 4)
  - Output size formula:
    - size: **n' × n'**
    - where **n' = n – f + 1**

6 x 6 image

8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0
8	8	8	0	0	0

dot

3 x 3 filter

1	0	-1
1	0	-1
1	0	-1

=

4 x 4 image

0	24	24	0
0	24	24	0
0	24	24	0
0	24	24	0

- Problem:
  - Edges are used less by the convolution (less relevant)
  - Output image shrinks
  - Sometimes image is not a multiple of the filter

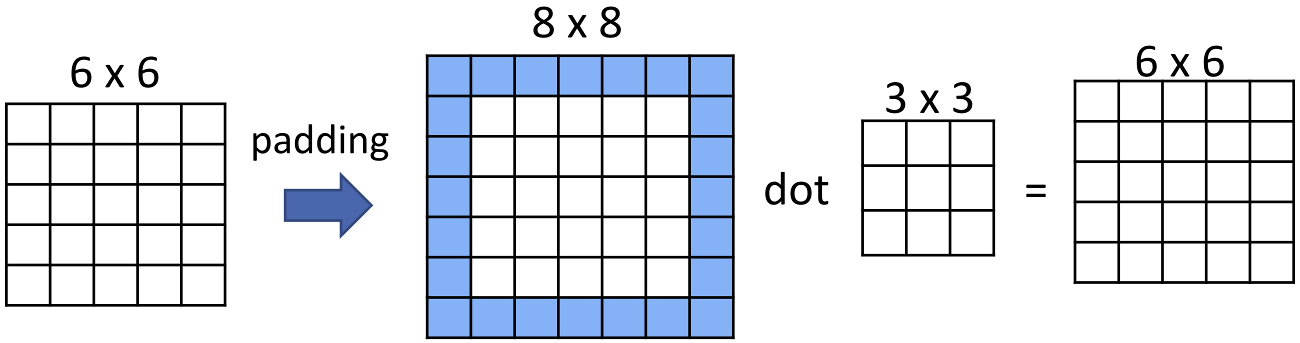
Padding - Definition

- Padding:
  - Extend input image adding a "frame"

- Convolution size:
  - Input image: **n × n**
  - Filter: **f × f**
  - Output image: **n' × n'**
  - **Without padding**:
    - **n' = n – f + 1**
  - **With padding** of **p**:
    - **n' = n + 2p – f + 1**

- More definitions:
  - "Same" convolution: convolution with padding when output image and input image size are the same
  - "Valid" convolution: no padding

- In PyTorch you have to specify the size of the padding (Default is 0)



- Example Convolution with padding of 1:
  1. Do padding and obtain a 8 x 8 image
  2. Do convolution and obtain a 6 x 6 image

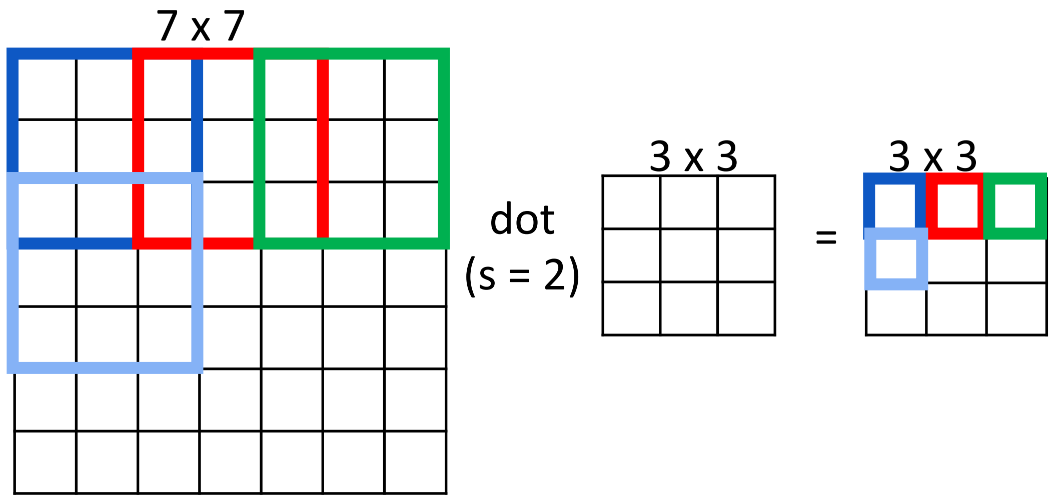
Stride

- Sometimes we want to obtain a **smaller output image** than the standard convolution or we want to have a lighter computation
- We can have the filter **jump a few pixels** at each step (stride)
- There are several implications:
  - The **compute time will diminish**
  - The **output size also will diminish**
  - The **convolution will be less fine-grained** (precise)

Stride - Definition

- Stride:
  - Jump S elements when moving filter
  - If filter falls outside image:
    - ignore computation: no output
- Convolution size:
  - Input image: **n × n**
  - Filter: **f × f**
  - Output image: **n' × n’**
  - Stride size: **s**
  - **With padding** of **p** and stride **s**:

$$n' = \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

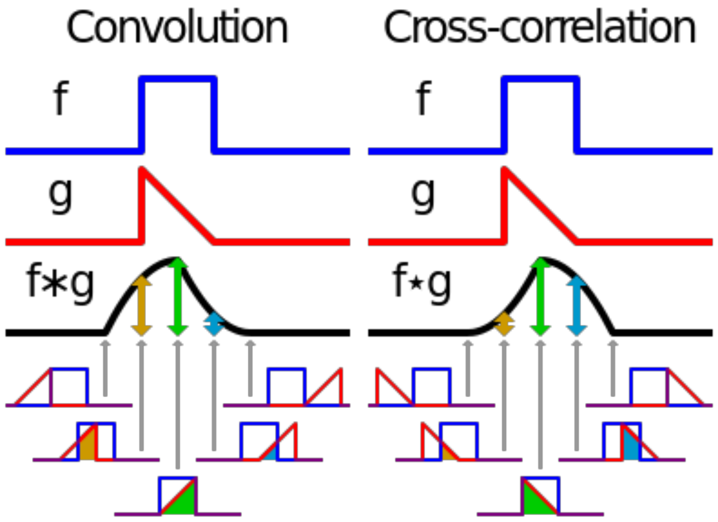


- Example Convolution with stride of 2
  - Input image: 7x7
  - Filter: 3x3
  - Stride: 2
  - Output image **n' × n'** : 3x3

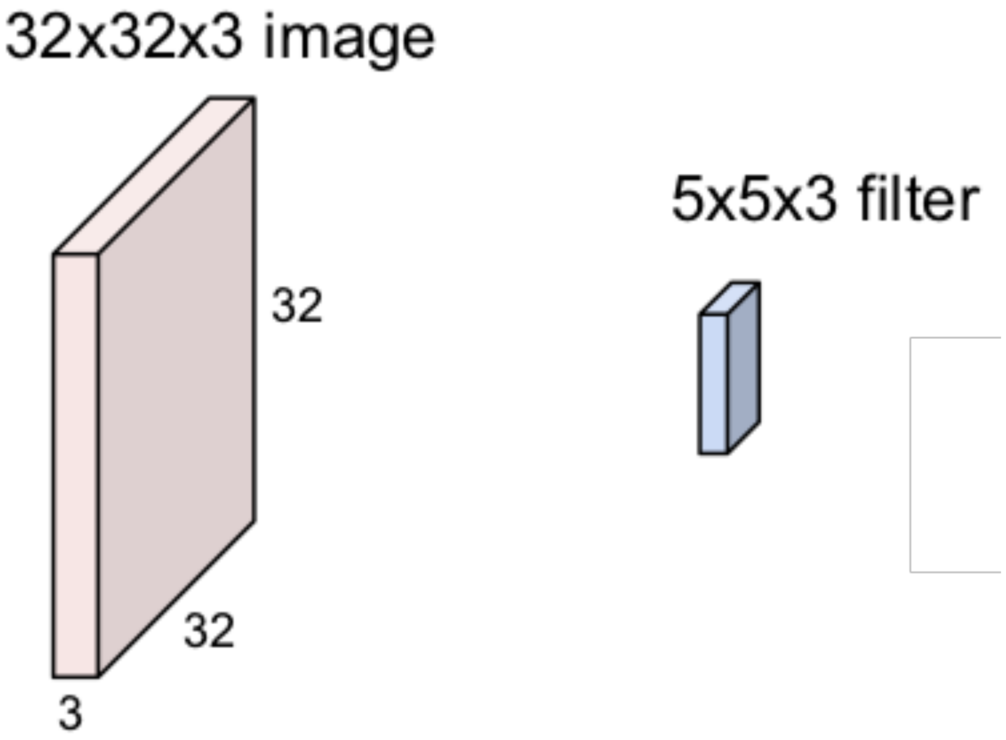
$$n' = \left\lfloor \frac{7+0-3}{2} + 1 \right\rfloor = 3$$

Convolution vs. Cross-correlation

- What we have been using is actually called **cross-correlation**
- **Math Actual Convolution:**
  - take the **transpose** of the filter before doing the dot product
- **Deep Learning:**
  - Taking the transpose doesn't significantly affect the results so is omitted to save compute time



Convolution Layer with 3 dimensions



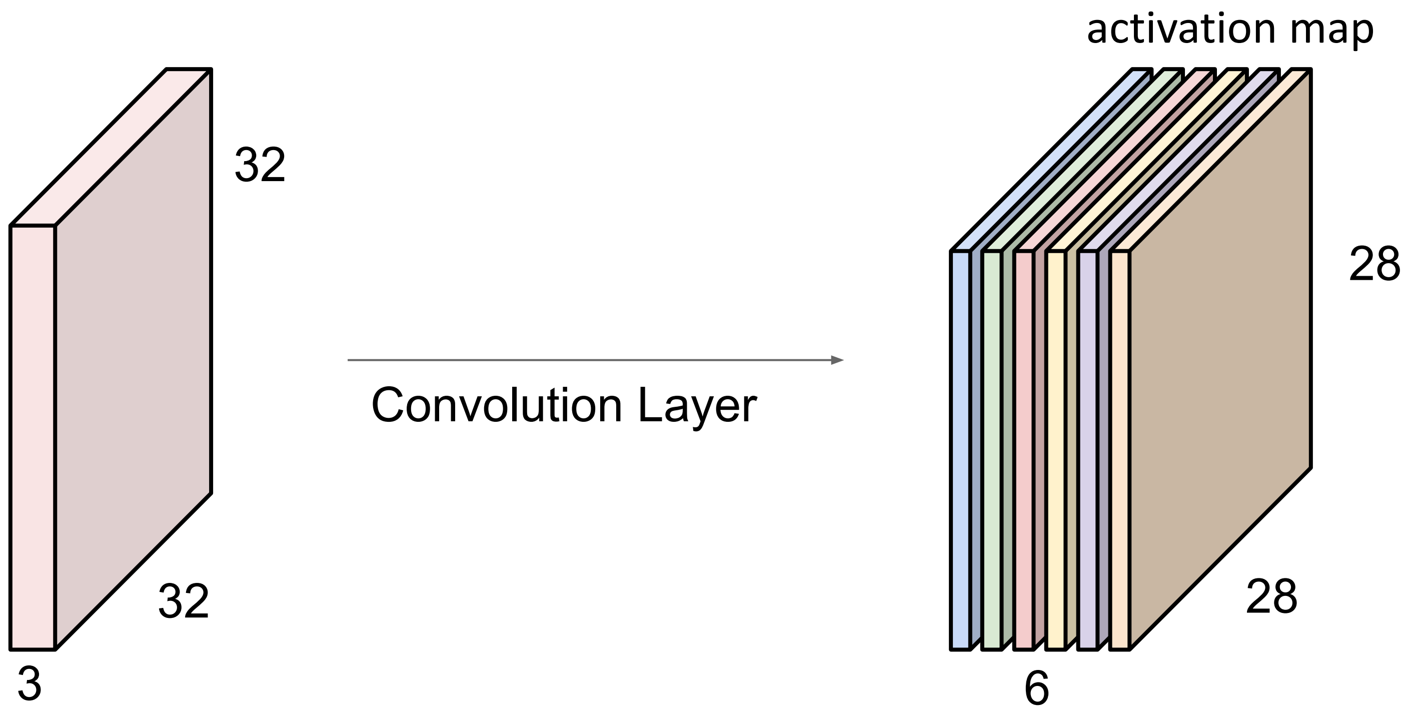
- Preserve the spatial structure of the image
- Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"
- Filters always extend the full depth of the input volume (the 3 channels in this example)

Output image: Activation Map

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)

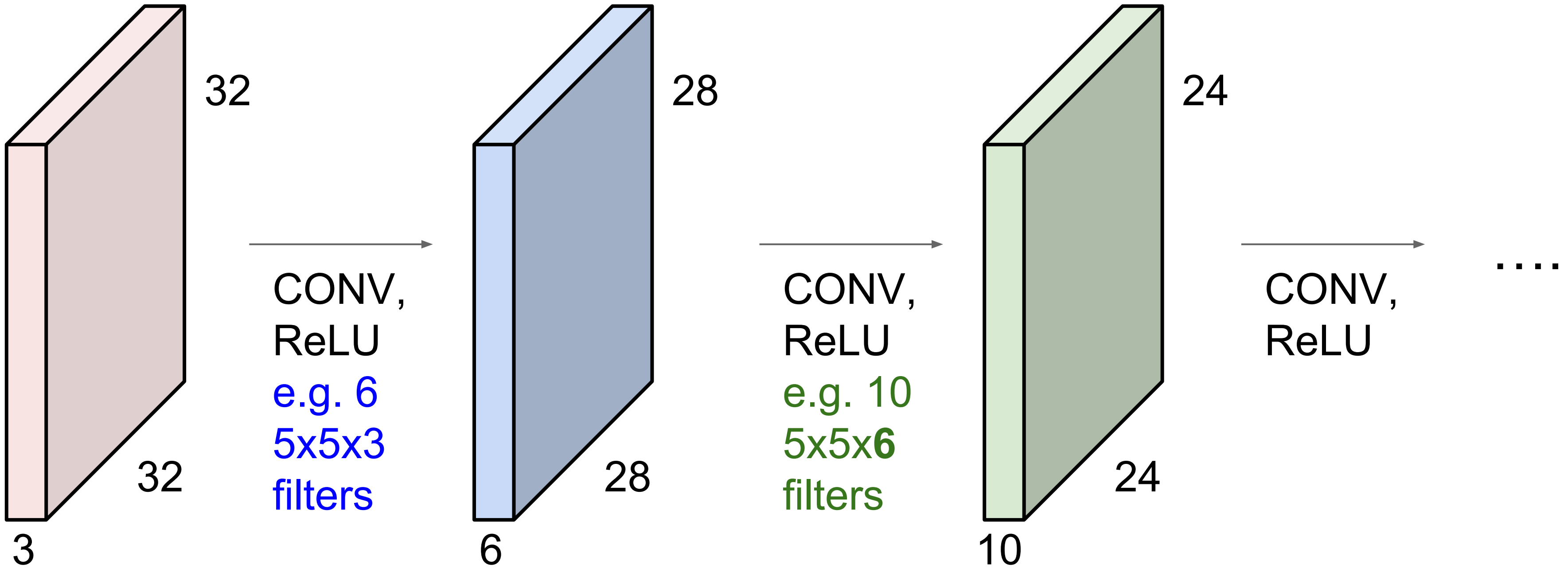
Multiple filters Activation Map

- If we had 6 5x5x3 filters, we'll get **6 separate activation maps**
- We stack these up to get a "new image" of size 28x28x6!



Convolutional Network

- A Convolution Network is a sequence of Convolutional Layers, interspersed with activation functions

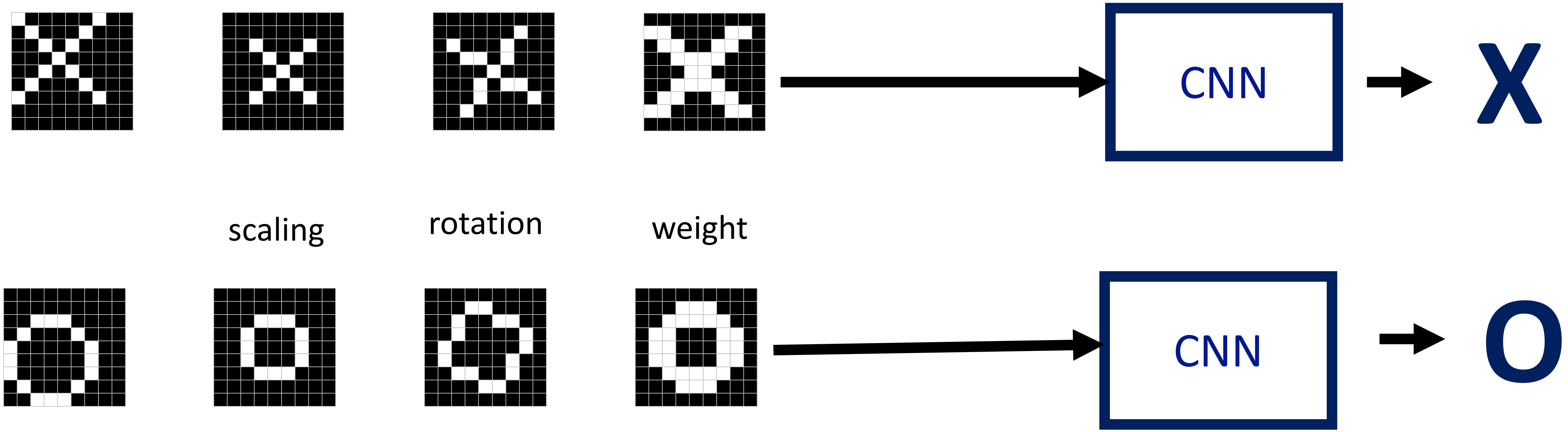


Receptive field

- A convolutional layer operates over a local region of the input to that layer
- The effective **receptive field** of a convolutional layer is the size of the input region to the network that contributes to a layers' activations
- For example:
  - if the first convolutional layer has a receptive field of 3x3 then it's effective receptive field is also 3x3
  - However if the second layer also has a 3x3 filter, then it's (local) receptive field is 3x3, but it's effective receptive field is 5x5

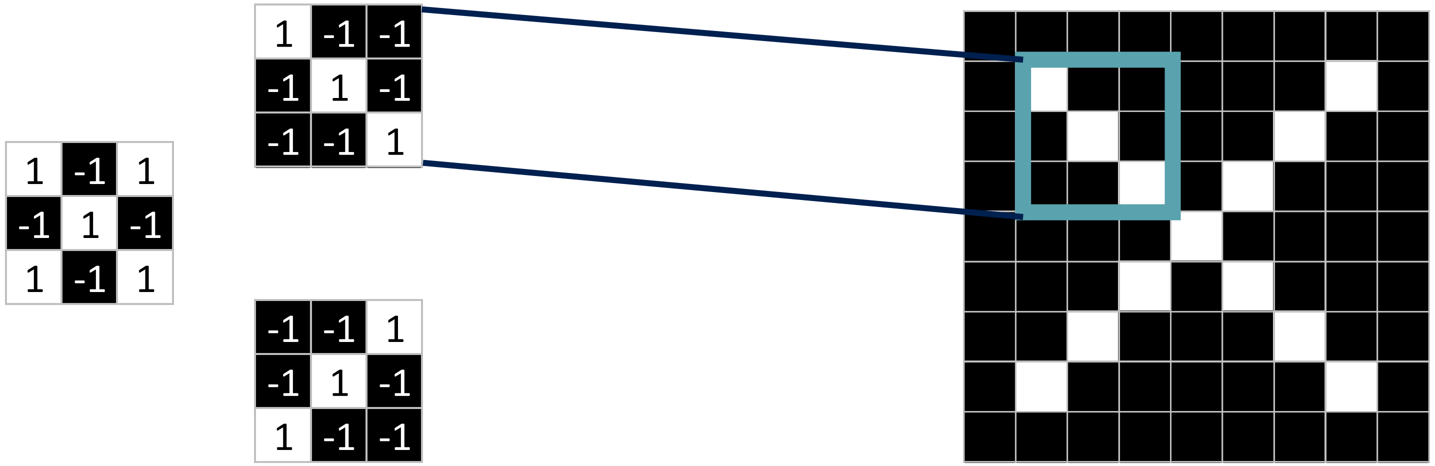
Convolution example – Image Classification

- Says whether a picture is of an X or an O

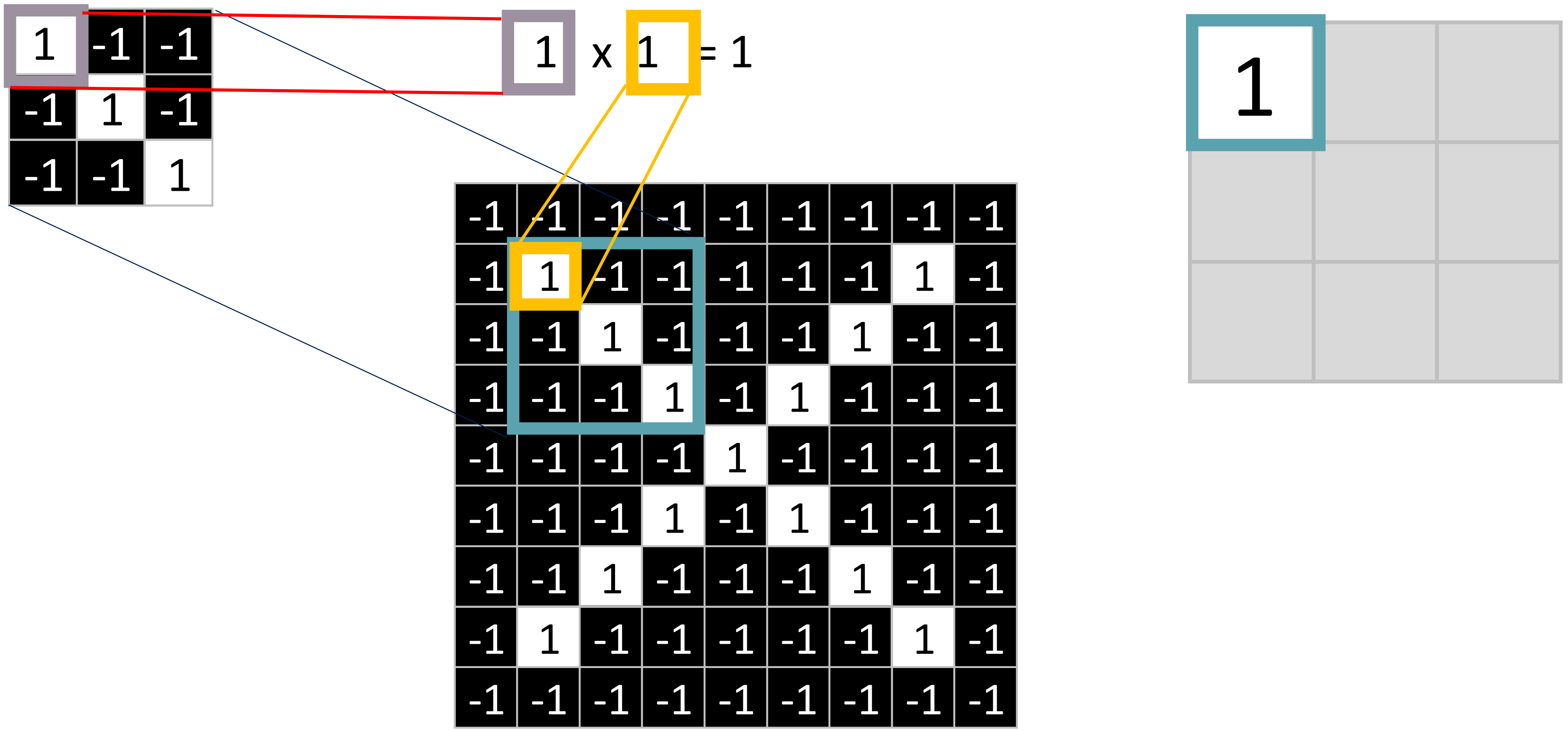


Feature recognition Example

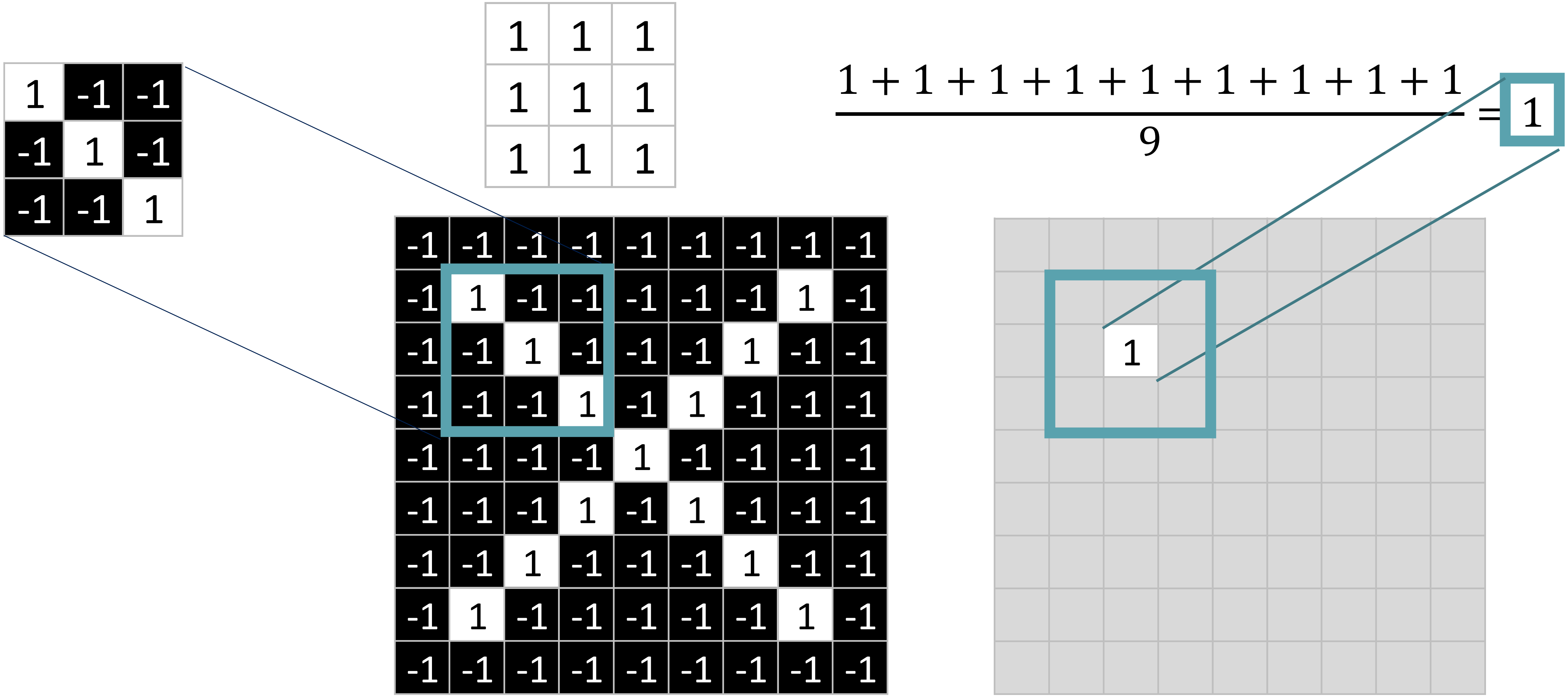
1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature **[optional: to normalize values]**



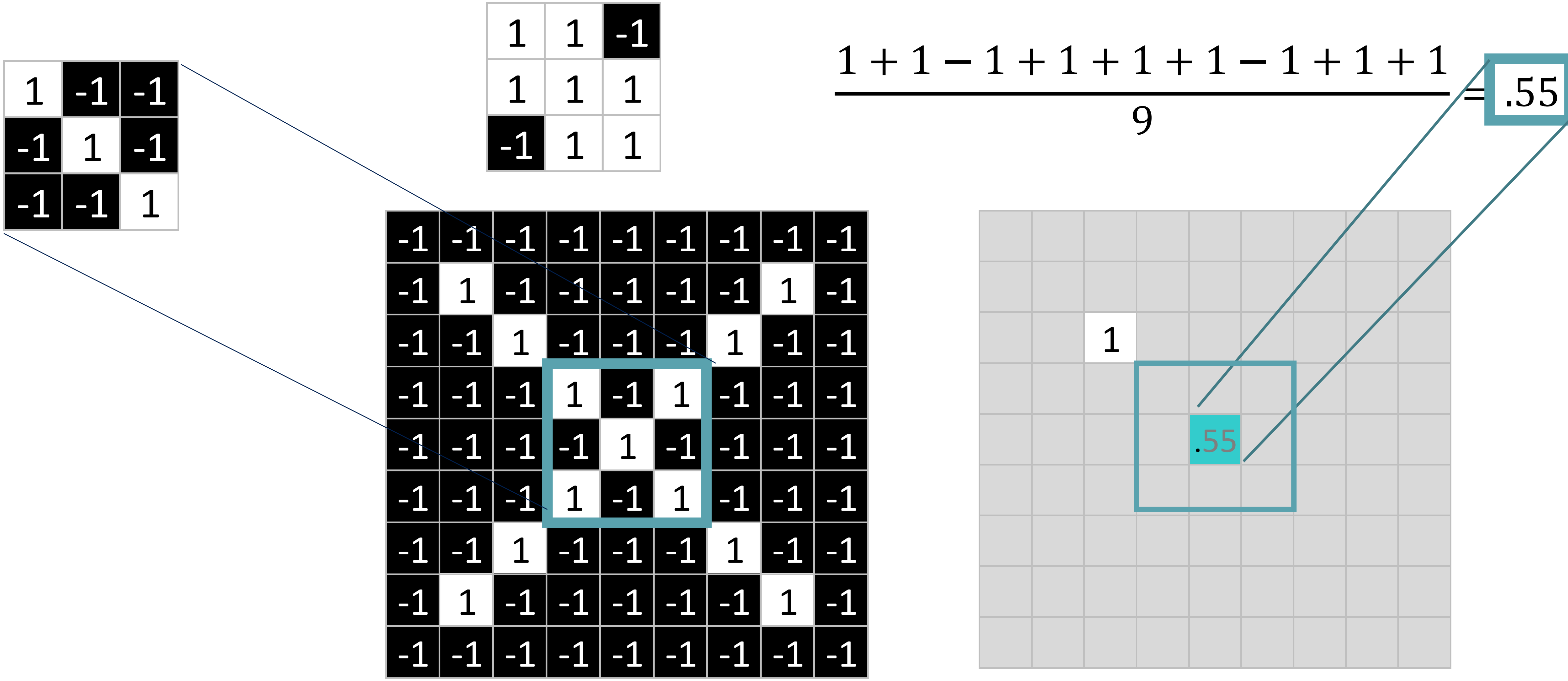
Feature recognition



Feature recognition



Feature recognition



$$\frac{1+1-1+1+1+1-1+1+1}{9} = .55$$

Convolution: Search every possible match

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

$\otimes$

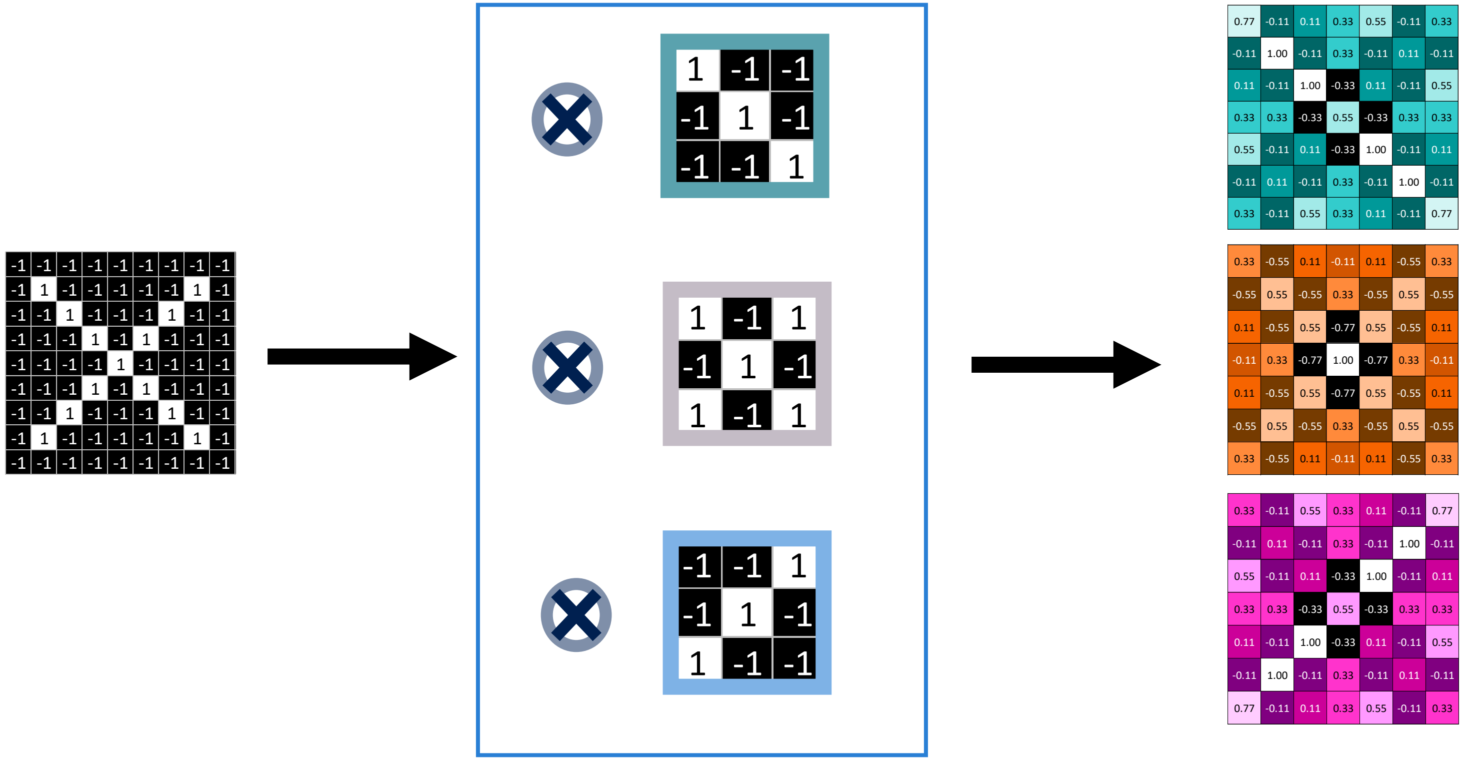
1	-1	-1
-1	1	-1
-1	-1	1

=

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Convolution Layer

- Using Multiple Filters: One image becomes a stack of features (filtered images)



Pooling: Shrinking the image stack

1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually the same window size).
3. Walk your window across your filtered images.
4. From each window, take the maximum (or average) value.

Max Pooling

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

maximum

1.00	0.33	0.55	0.33

- Window size=2
- Stride=2

Max Pooling

A stack of images becomes a stack of smaller images.

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.11	0.33	-0.77	1.00	-0.77	0.33	-0.11
0.11	-0.55	0.55	-0.77	0.55	-0.55	0.11
-0.55	0.55	-0.55	0.33	-0.55	0.55	-0.55
0.33	-0.55	0.11	-0.11	0.11	-0.55	0.33

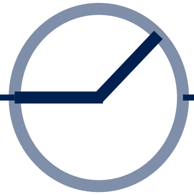
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	-0.11	0.55	0.33	0.11	-0.11	0.77
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

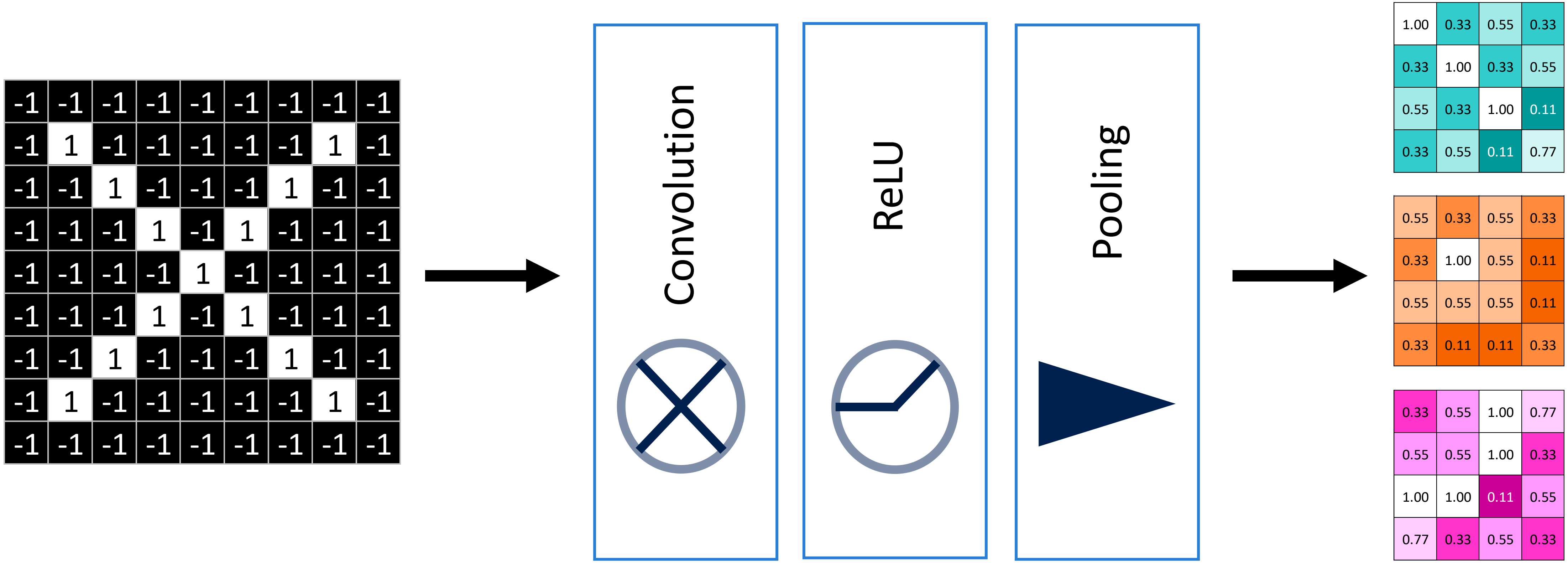
Rectified Linear Units (ReLUs)

0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

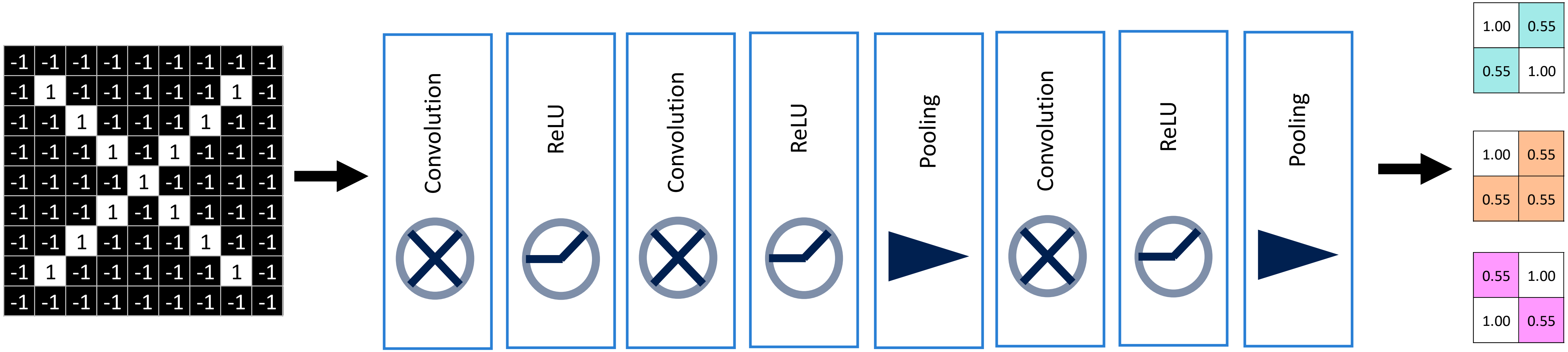


0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

Layers get stacked

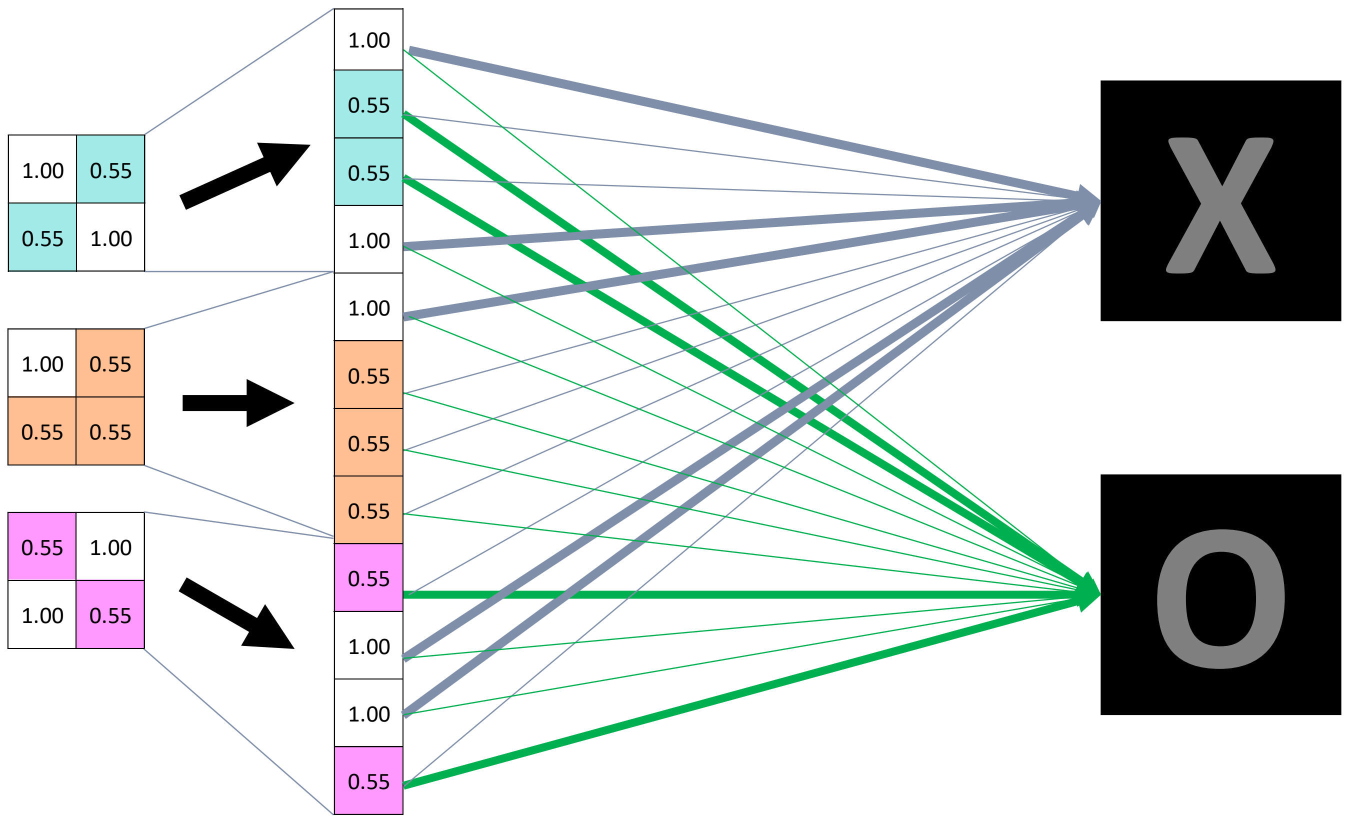


Layers get stacked

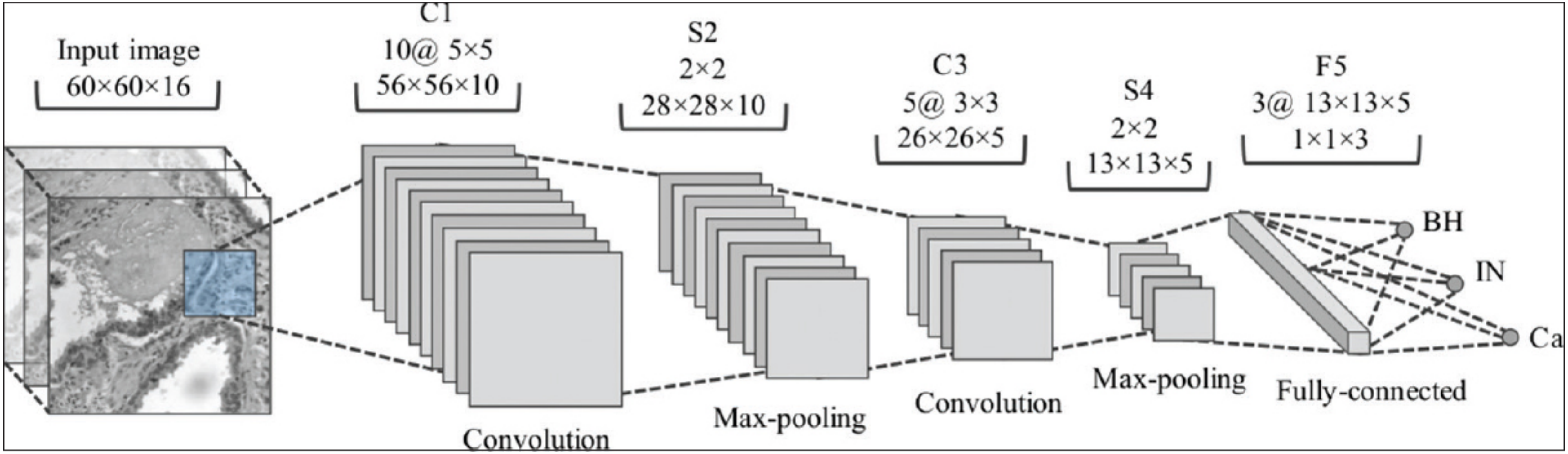


Classification with a Fully Connected Layer

- Stack of images obtained with multiple filters can be the input of FC layer



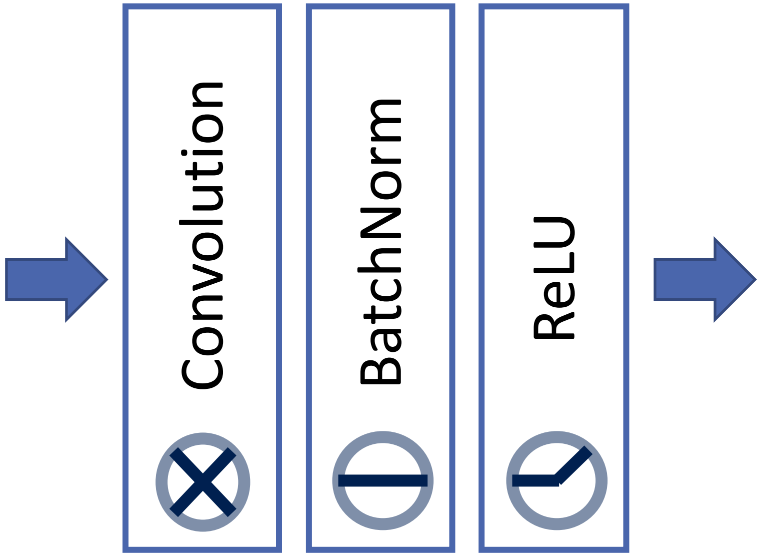
CNN Application Example



- Classifications of multispectral colorectal cancer tissues using convolution neural network, **J Pathol Inform** 2017, **8**:1

Batch Normalization Layer

- Distribution of each layer input changes during training
  - Because weights change after each update (step)
- Batch Normalization is used to normalize the distribution of the inputs
  - Use of higher learning rate
  - Faster convergence
  - In Pytorch:
    - BatchNorm1d(), BatchNorm2D(), BatchNorm3D()
    - Based on the formula from paper: https://arxiv.org/abs/1502.03167



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

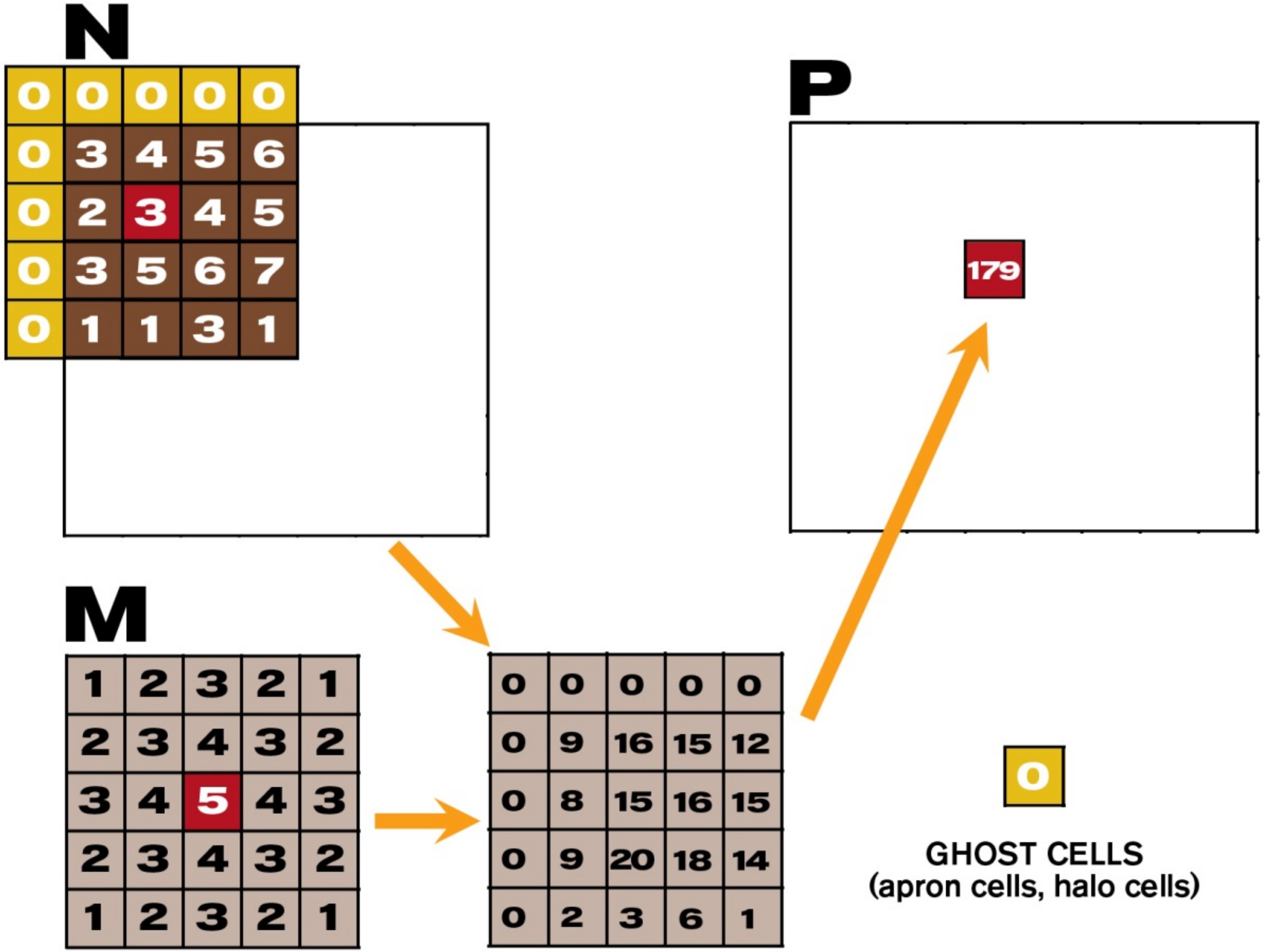
CNN in PyTorch Example - MNIST

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.fc = nn.Linear(7*7*32, 10)
```

```
    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

2D Convolution in CUDA

- N is the image
- M is the filter (mask)
- Padding elements (or ghost cells) are set to 0
- Padding can also used to have alignment for better CUDA performance
  - alignment to 4,8,16 required for best performance



Simple 2D Convolution in CUDA

- *in* is the input image
  - Assume already allocated
- *out* is the output image
  - Assume already allocated
- Set all pixels to 0 (including padding)
- If our thread is outside the boundaries *h* and *w* we skip the computation

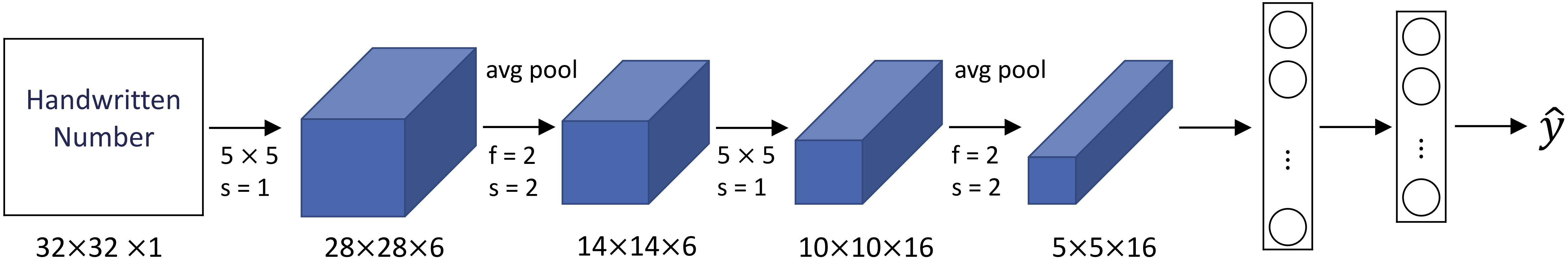
```
__global__ void convolution_2D_basic_kernel(unsigned char * in, unsigned char * filter, unsigned char * out,
            int filterwidth, int w, int h) {
      int Col  = blockIdx.x * blockDim.x + threadIdx.x;
      int Row  = blockIdx.y * blockDim.y + threadIdx.y;

      if (Col < w && Row < h) {
          int pixVal = 0;
          N_start_col  = Col -   (filterwidth/2);
          N_start_row = Row - (filterwidth/2);

         for(int j = 0; j < filterwidth; ++j) {
              for(int k = 0; k < filterwidth; ++k) {

                  int curRow = N_Start_row + j;
                  int curCol =   N_start_col  + k;
                  // Verify we are inside the boundaries h and w
                  if(curRow > -1 && curRow < h && curCol > -1 && curCol < w) {
                      pixVal += in[curRow * w + curCol] * filter[j*filterwidth+k];
                  }
              }
          }
          // Write our new pixel value out
          out[Row * w + Col] = (unsigned char)(pixVal);
      }
  }
```

Standard CNN Networks

LeNet –1998

- First CNN paper
- No Padding
- Few parameters: 60K
- Sigmoid/Tanh activation
- Different filters would look at different channels to reduce computation cost
- Paper: "Gradient Based Learning Applied to Document Recognition" Yann LeCun Leon Bottou Yoshua Bengio and Patrick Haner

LeNet

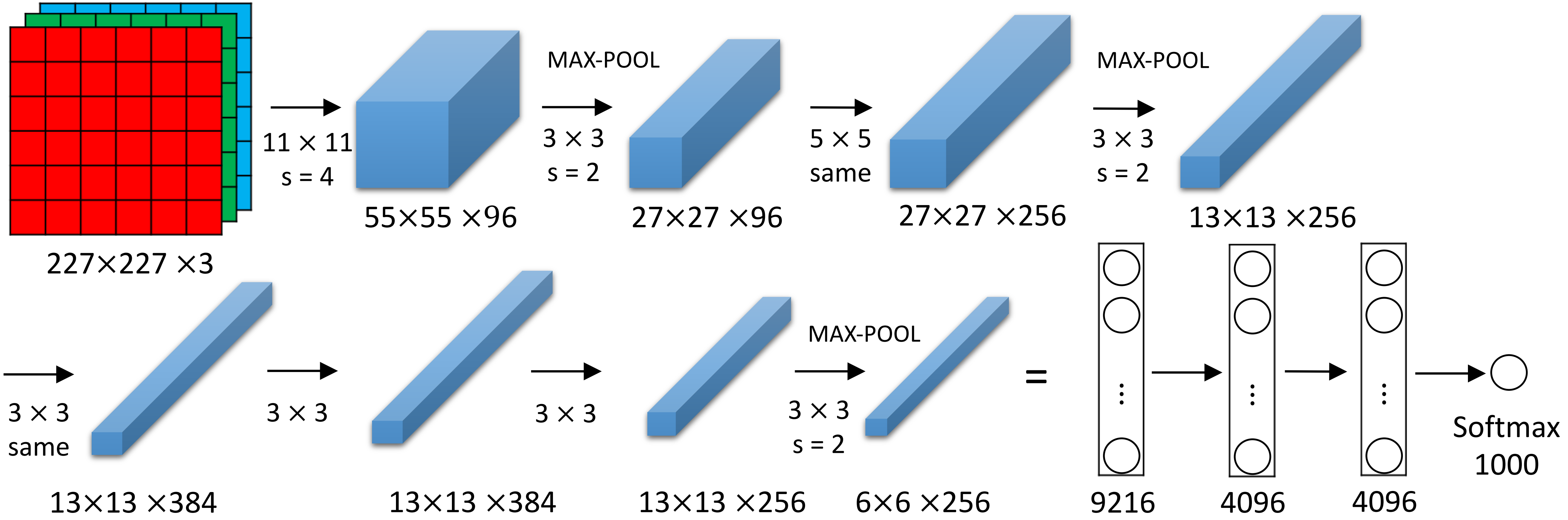


- First CNN network
- Note how tensors become deeper
- Relatively small network applied to classification of handwritten numbers

AlexNet - 2012

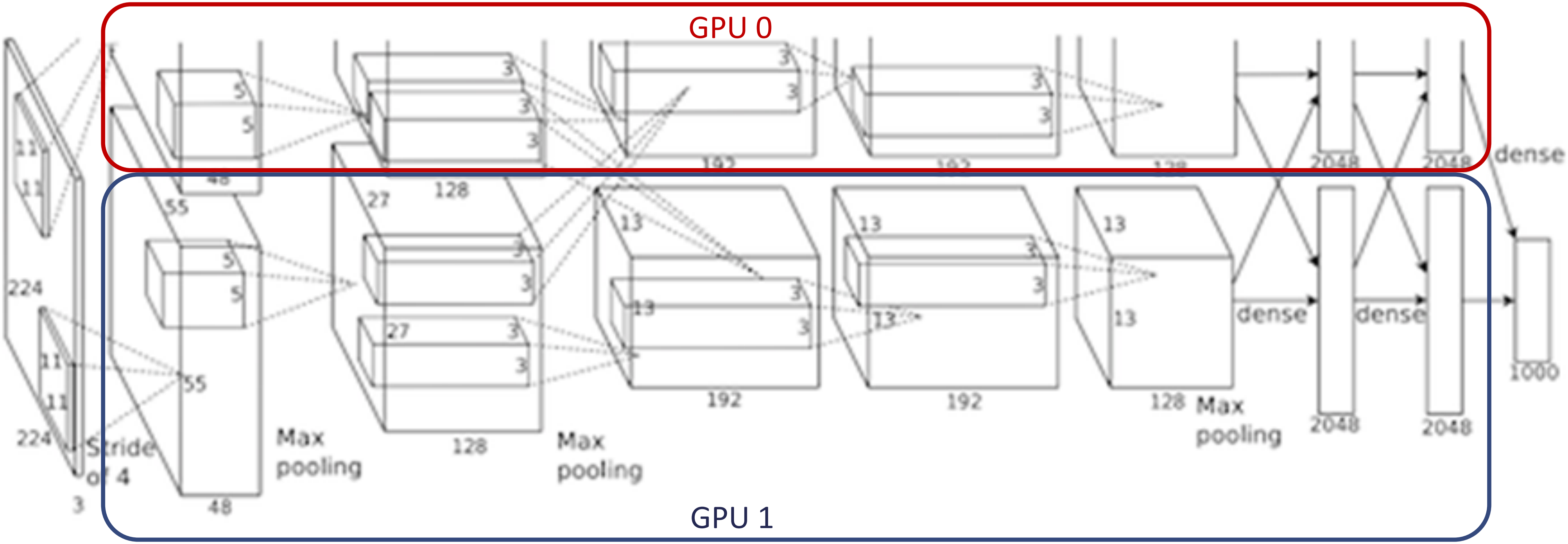
- First time CNN perform so well on ImageNet dataset (15M images)
- Won 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge) Classification with 15.4% top-5 error rate (next best entry: 26.2%)
- Parameters:
  - About 60M
- Training algorithm and hardware:
  - **Model parallelism** with **2 GTX 580 Fermi GPUs for 6 days**
- ReLU, Dropout, Normalization, regularization etc.
- **Relevance**: show power of CNN => started DL trend
- Paper: “ImageNet Classification with Deep Convolutional Neural Networks” Krizhevsky, Sutskever, Hinton.

AlexNet



- Larger model than before: 60M
- 3x3 convolutions and max-pool layers

AlexNet – Model Parallelism with 2 GPUs



AlexNet architecture (May look weird because there are two different "streams". This is because the training process was so computationally expensive that they had to split the training onto 2 GPUs)

https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html

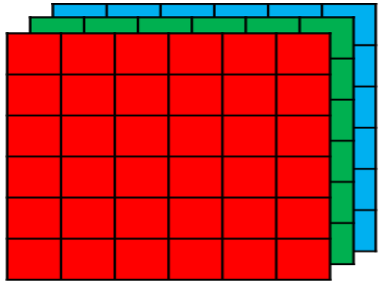
VGG - 2014

- Simple but deep models
- Very regular structure
- Trained on 4 Nvidia Titan Black for 2/3 weeks (**data parallelism**)
- 7.3% error rate on ILSVRC 2014 Classification
- About 138M parameters
- Paper: “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION” Karen Simonyan & Andrew Zisserman

VGG

CONV = 3×3 filter, s = 1, same

MAX-POOL = 2×2 , s = 2



224×224 ×3 → [CONV 64] ×2 → 224×224×64 → POOL → 112×112 ×64 → [CONV 128] ×2 → 112×112 ×128 → POOL → 56×56 ×128

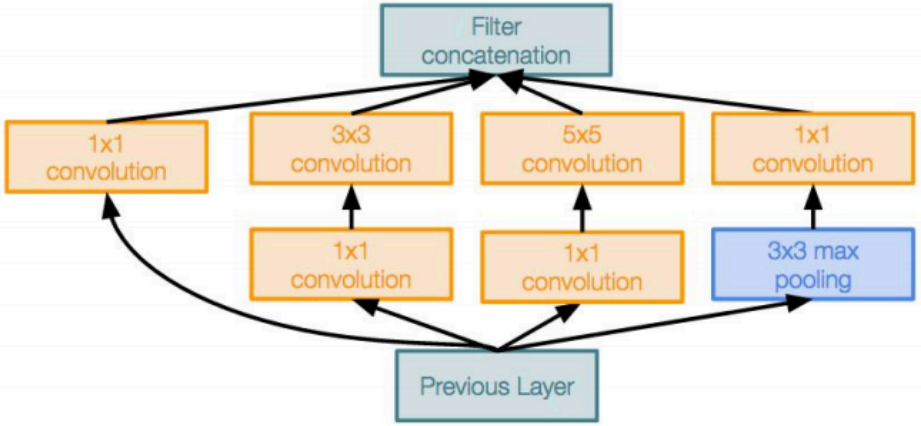
→ [CONV 256] ×3 → 56×56 ×256 → POOL → 28×28 ×256 → [CONV 512] ×3 → 28×28 ×512 → POOL → 14×14×512

→ [CONV 512] ×3 → 14×14 ×512 → POOL → 7×7×512 → FC 4096 → FC 4096 → Softmax 1000

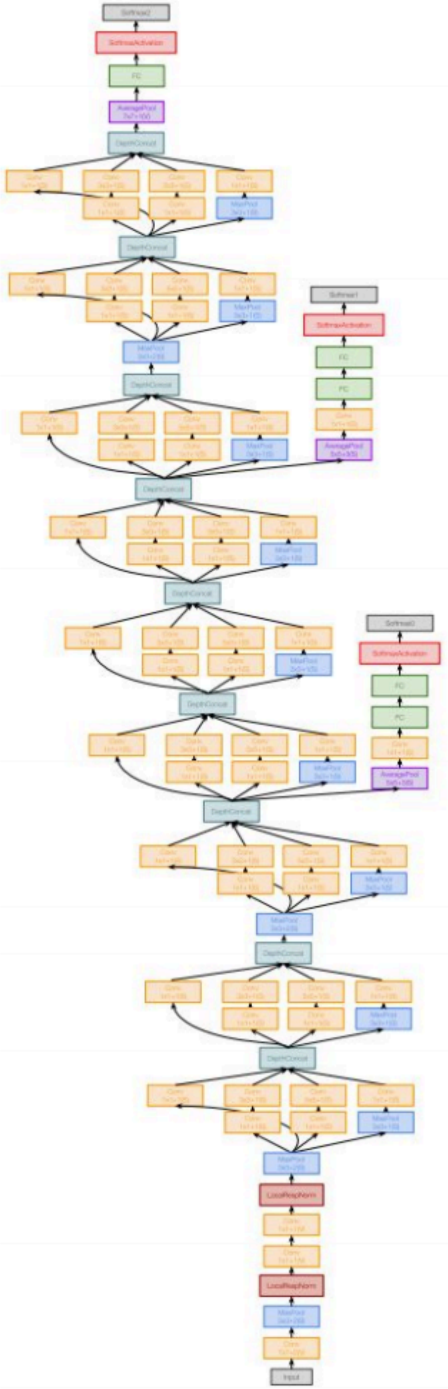
- 2 layers of convolutions interleaved with a pool layer
- Convolutions filters get deeper at each layer

GoogleNet (Inception v1)

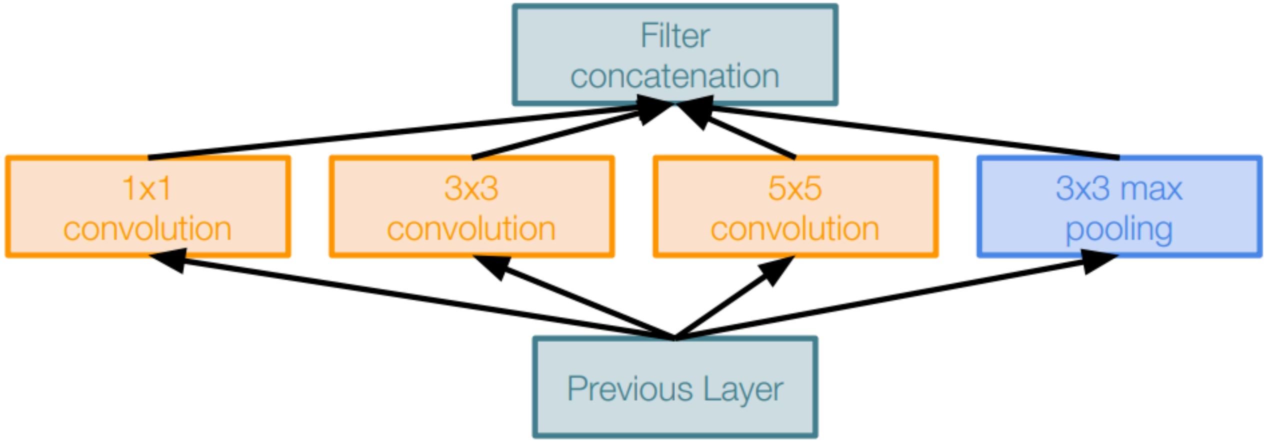
- Deeper networks, with computational efficiency
  - 22 layers
  - Efficient “Inception” module
  - No FC layers
  - Only 5 million parameters!
  - 12x less than AlexNet
  - ILSVRC’14 classification winner
  - (6.7% top 5 error)
- **Inception** module: design a good local network topology (network within a network) and then stack these modules on top of each other
- Paper: *Going Deeper with Convolutions*. Szegedy et al., 2014



Inception module



Inception module

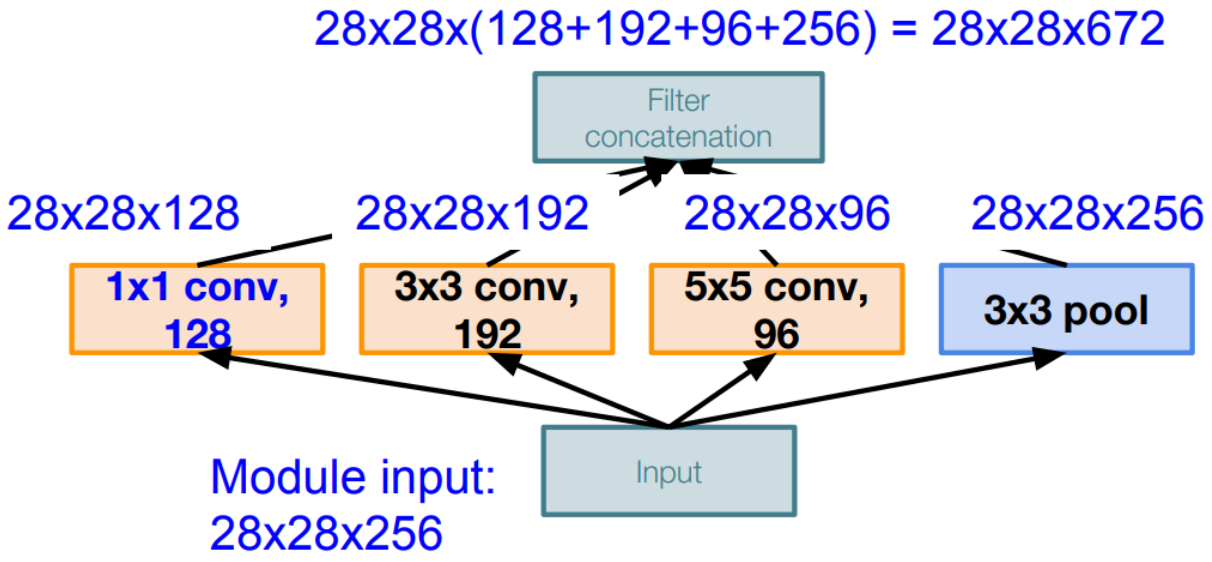


Naive Inception module

- Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise
- What is the problem with this?

→ Computational complexity

Inception module

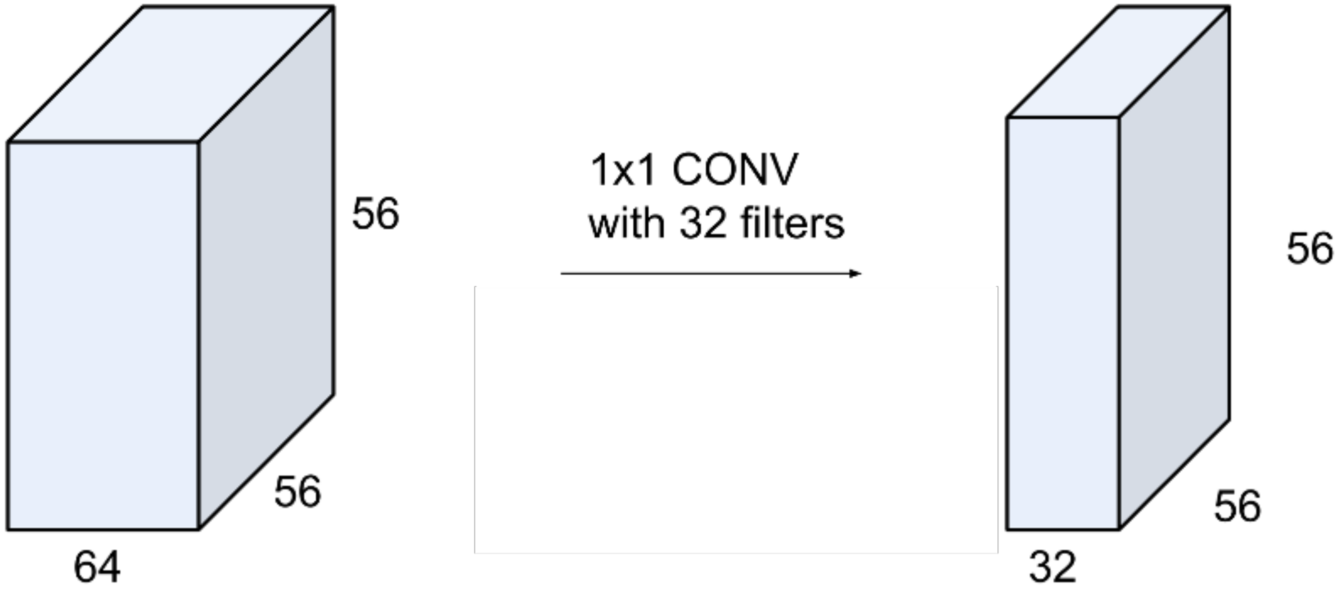


Naive Inception module

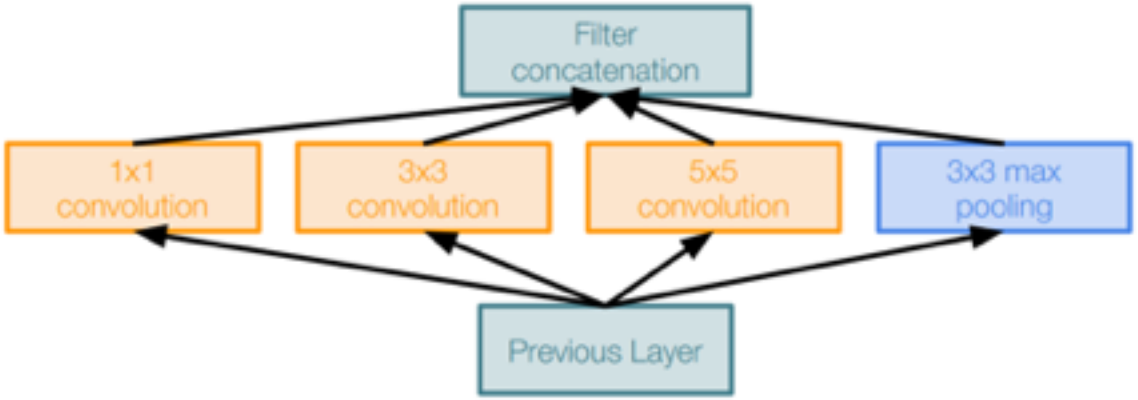
- Conv Ops:
  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x256
  - [5x5 conv, 96] 28x28x96x5x5x256
- Total: 854M ops
- Very expensive computation
- Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!
- Solution: **bottleneck** layers that use 1x1 convolutions to reduce feature depth

1x1 Bottleneck convolution

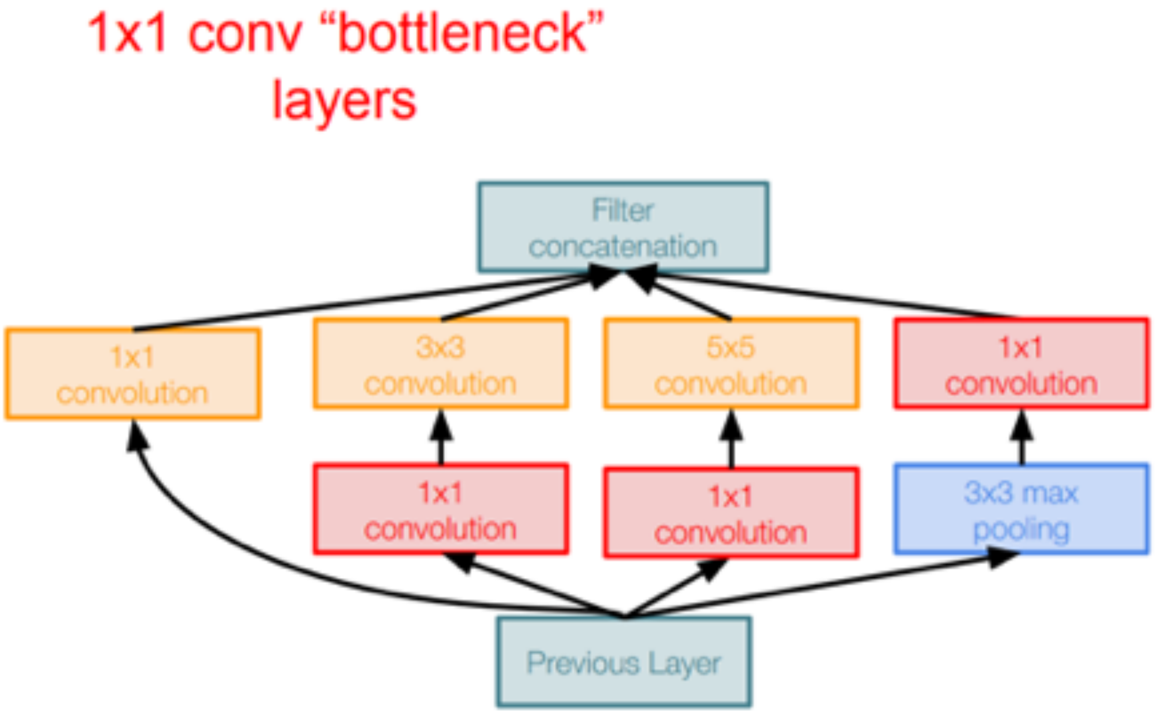
- Each filter has size $1x1xC_{in}$ and performs a C_{in}-dimensional dot product
- Preserves spatial dimensions, reduces depth!
- Projects depth to lower dimension (= combination of feature maps)



Inception module - Bottleneck layers

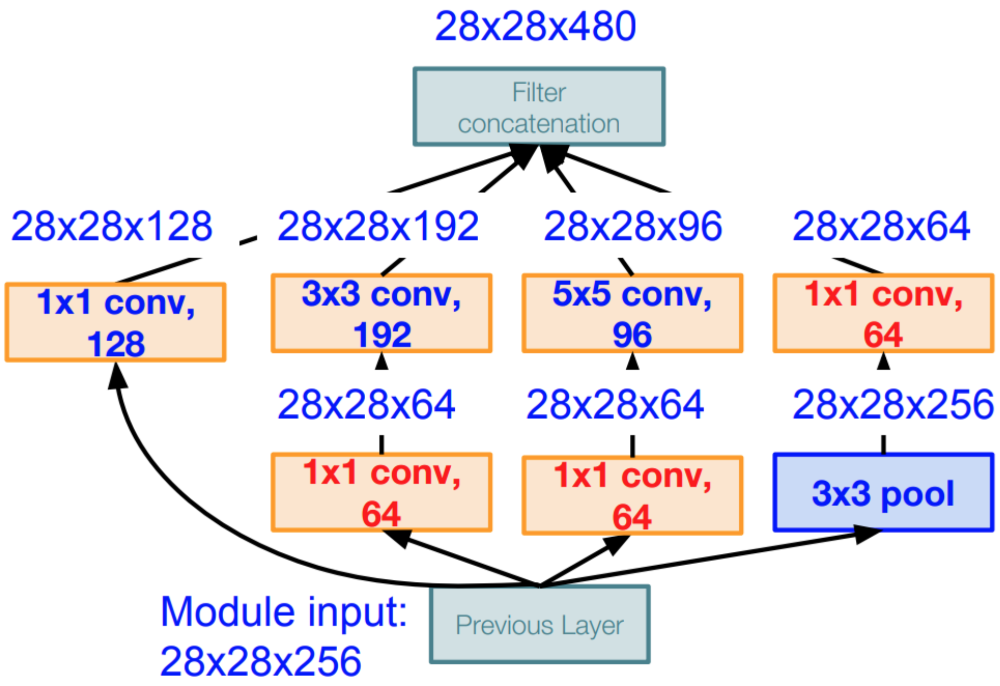


Naive Inception module



Inception module with dimension reduction

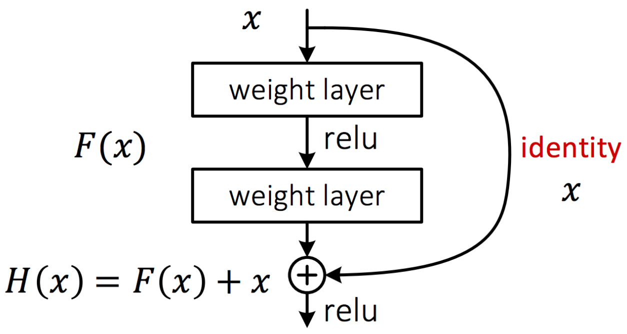
Inception module

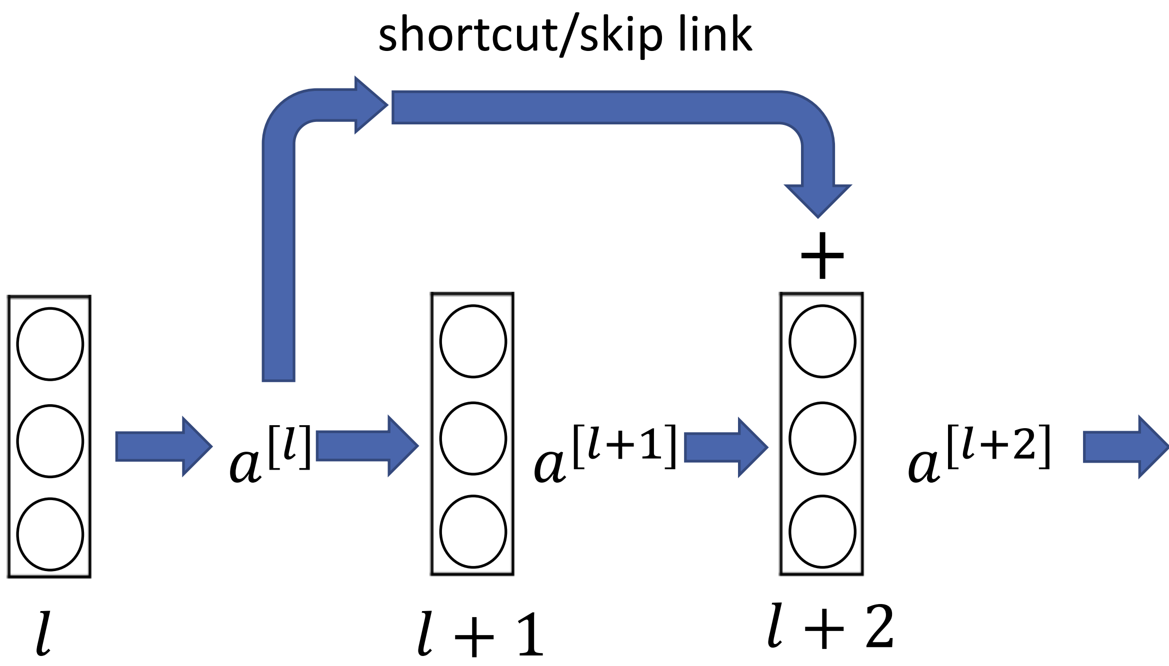


- Using same parallel layers as naive example, and adding 1x1 conv with 64 filter bottlenecks:
- Conv Ops:
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 64] 28x28x64x1x1x256
  - [1x1 conv, 128] 28x28x128x1x1x256
  - [3x3 conv, 192] 28x28x192x3x3x64
  - [5x5 conv, 96] 28x28x96x5x5x64
  - [1x1 conv, 64] 28x28x64x1x1x256
- Total: 358M ops
  - Compared to 854M ops for naive version
- Bottleneck can also reduce depth after pooling layer (480 compared to 672)

ResNet - Residual block

- **The (degradation) problem:**
  - With network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error.
- **The core insight:**
  - Let us consider a shallower architecture and its deeper counterpart that adds more layers onto it. There exists a solution to the deeper model by construction: the layers are copied from the learned shallower model, and the added layers are identity mapping. The existence of this constructed solution indicates that a deeper model should produce no higher training error than its shallower counterpart.





$$z^{[l+1]} = W^{[l+1]}\, a^{[l]} + b^{[l+1]}$$

$$a^{[l+1]} = g(z^{[l+1]})$$

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$

$$a^{[l+2]} = g(z^{[l+2]} + \boldsymbol{a}^{[l]})$$

Residual Networks Benefits

- Easier to optimize and to train “residual” than default layers
- Think about it as computing a “delta” or a slight change of the original input

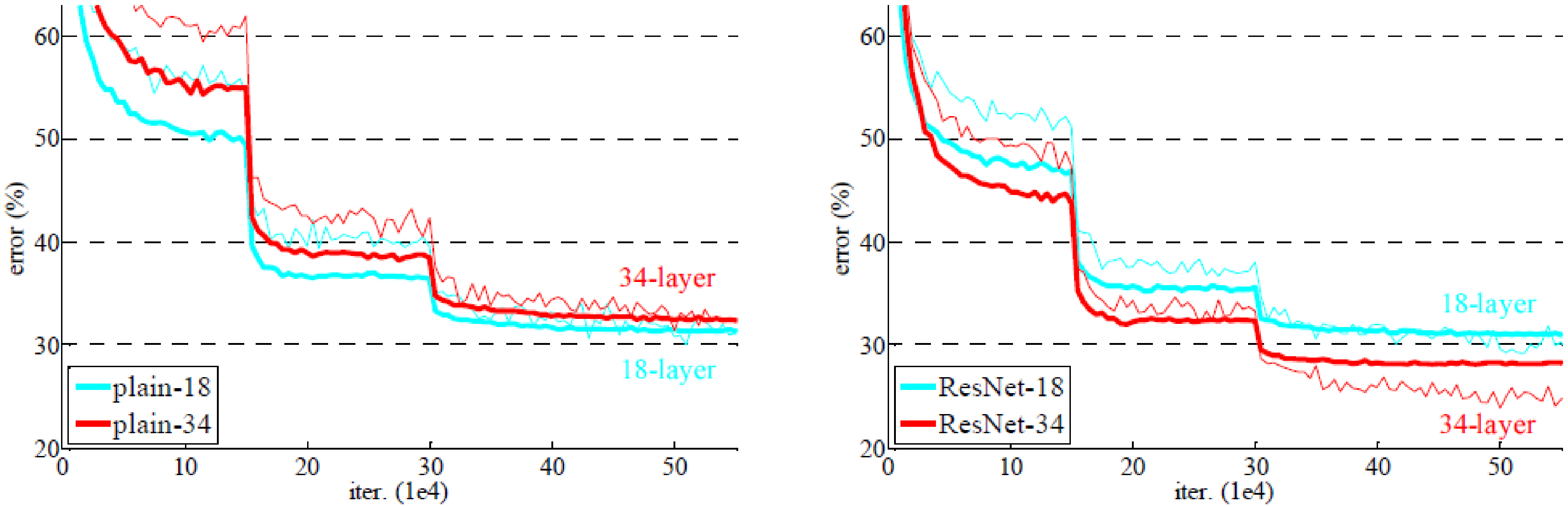
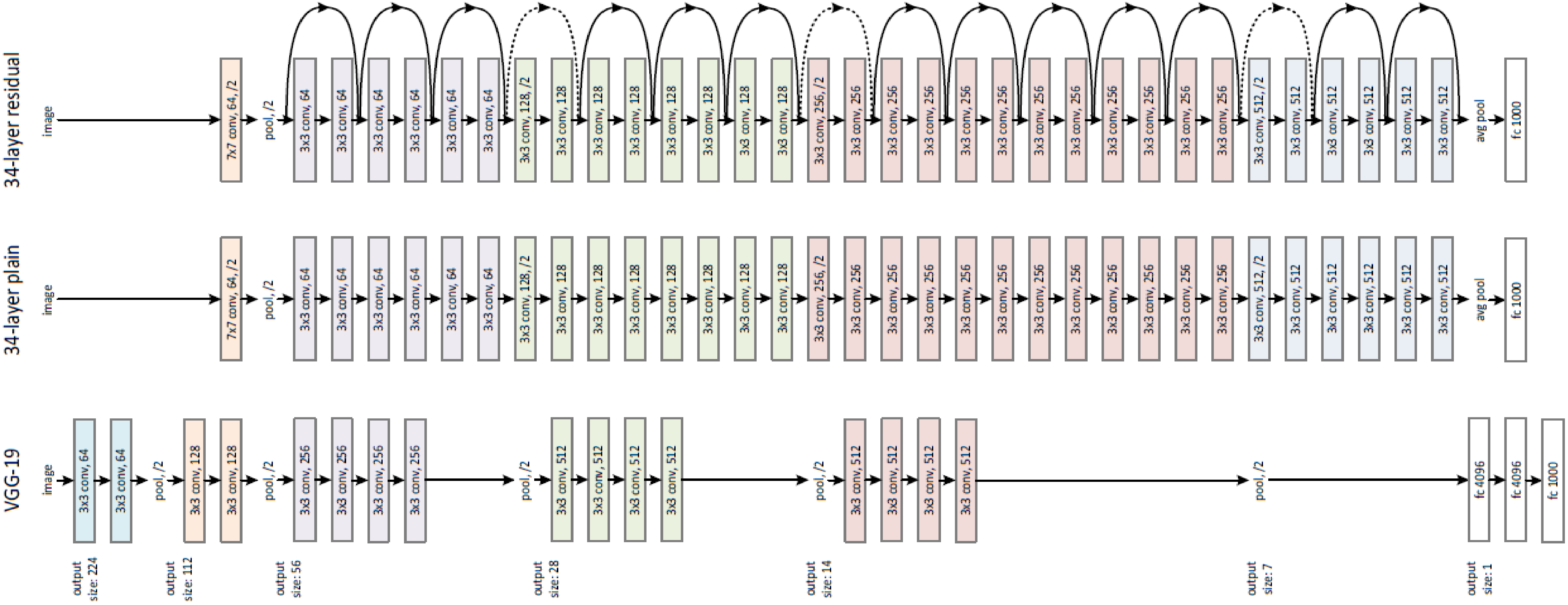


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

[from: “Deep Residual Learning for Image Recognition” Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun]()

ResNet vs Plain vs VGG

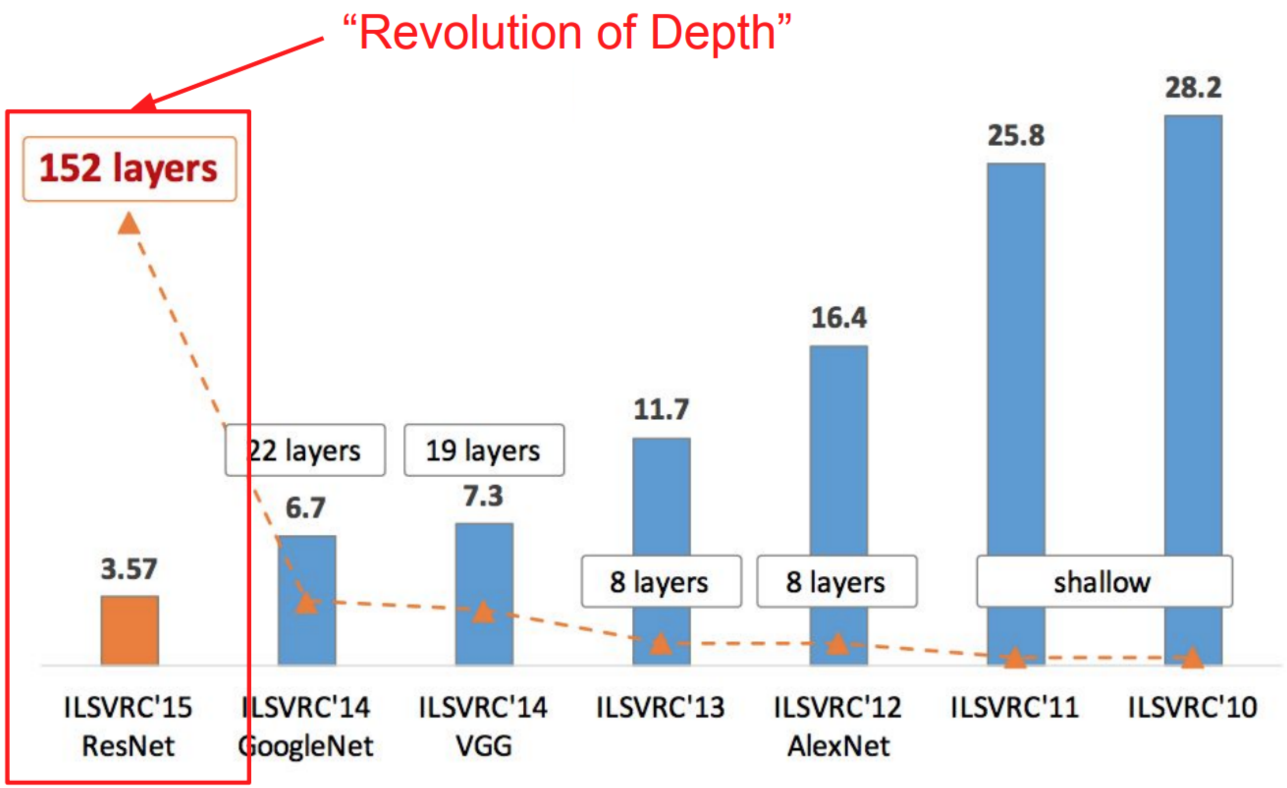


- 34 layer plain becomes 34 layers residual adding shortcut (skip) links
- Dotted shortcut: dimensions change and a transformation is needed to reduce dimensions
- /2 means stride 2 or pool 2x2 (reduce size by 4)

ResNet - 2015

- Uses Residual blocks
- Ultra Deep: 152 layers
- 3.57% error rate on ILSVRC 2014 Classification
- Paper: "Deep Residual Learning for Image Recognition" Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



And then? Bigger, deeper, hybrid

- From then 4 versions of GoogleNet/Inception were proposed
  - A nice guide on the differences between them: https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202
- Deeper Resnet (up to 500 layers)
- Hybrid Inception-ResNet
- Other bigger and deeper network architectures:
  - PolyNet
  - NASNet and PNASNet
  - DenseNet
  - SENet154
  - Xception
  - …

A comparison

- A comparison of several CNN architectures on ImageNet dataset
- From: https://github.com/Cadene/pretrained-models.pytorch

Model	Acc@1	Acc@5
PNASNet-5-Large	82.736	95.992
NASNet-A-Large	82.566	96.086
SENet154	81.304	95.498
PolyNet	81.002	95.624
SE-ResNeXt101_32x4d	80.236	95.028
InceptionResNetV2	80.17	95.234
InceptionV4	80.062	94.926
DualPathNet107_5k	79.746	94.684
DualPathNet131	79.432	94.574
DualPathNet92_5k	79.4	94.62
DualPathNet98	79.224	94.488
SE-ResNeXt50_32x4d	79.076	94.434
ResNeXt101_64x4d	78.956	94.252
Xception	78.888	94.292
SE-ResNet152	78.658	94.374

Model	Acc@1	Acc@5
SE-ResNet101	78.396	94.258
ResNeXt101_32x4d	78.188	93.886
FBResNet152	77.84	93.84
SE-ResNet50	77.636	93.752
DenseNet161	77.56	93.798
ResNet101	77.438	93.672
FBResNet152	77.386	93.594
InceptionV3	77.294	93.454
DenseNet201	77.152	93.548
DualPathNet68b_5k	77.034	93.59
CaffeResnet101	76.4	92.9
CaffeResnet101	76.2	92.766
DenseNet169	76.026	92.992
ResNet50	76.002	92.98
DualPathNet68	75.868	92.774

Model	Acc@1	Acc@5
DenseNet121	74.646	92.136
VGG19_BN	74.266	92.066
NASNet-A-Mobile	74.08	91.74
ResNet34	73.554	91.456
BNInception	73.522	91.56
VGG16_BN	73.518	91.608
VGG19	72.08	90.822
VGG16	71.636	90.354
VGG13_BN	71.508	90.494
VGG11_BN	70.452	89.818
ResNet18	70.142	89.274
VGG13	69.662	89.264
VGG11	68.97	88.746
SqueezeNet1_1	58.25	80.8
Alexnet	56.432	79.194

Lesson key points

- Convolutional Neural Networks
  - Convolution
    - Padding and Stride
    - Channels and Activation Map
  - Max Pooling
  - Batch Normalization
- Standard CNN Networks

- Part of this material has been adapted from the video "How do Convolutional Neural Networks work?" under the Creative Commons License
- Part of this material has been inspired by John Kenny's course "Designing, Visualizing and Understanding Deep Neural Networks" @ UC Berkeley (https://bcourses.berkeley.edu/courses/1453965/pages/cs294-129-designing-visualizing-and-understanding-deep-neural-networks)