Spring 2024

CMPE 258-01 Deep Learning

Dr. Kaikai Liu, Ph.D. Associate Professor

Department of Computer Engineering

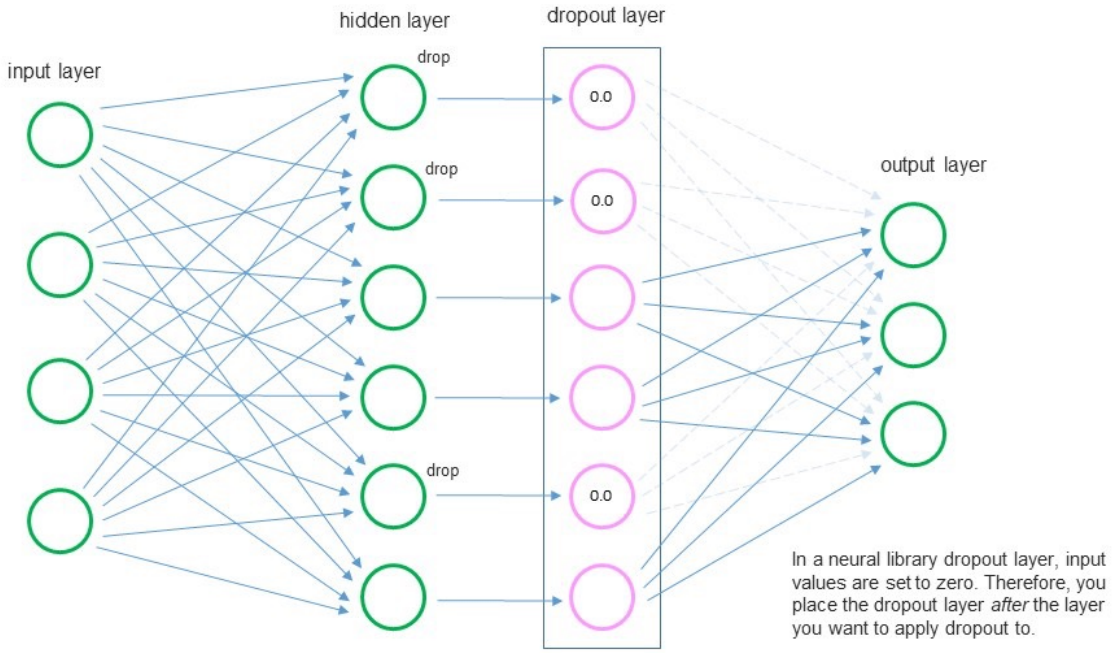
San Jose State University

Email: kaikai.liu@sjsu.edu

Website: https://www.sjsu.edu/cmpe/faculty/tenure-line/kaikai-liu.php

Dropout

- Dropout
  - Dropout is a solution proposed to this problem by Nitish Srivastava, Geoffrey Hinton and few other students at the University of Toronto in 2012. Hinton is now an employee at Google, leading to the giant picking up the patent for the technology.
    - https://patents.google.com/patent/WO2014105866A1/en



OPINIONS

Google Activates 'Dropout' Patent, Neural Network Engineers To Be On Alert?

ANIRUDH VK - 3 WEEKS AGO

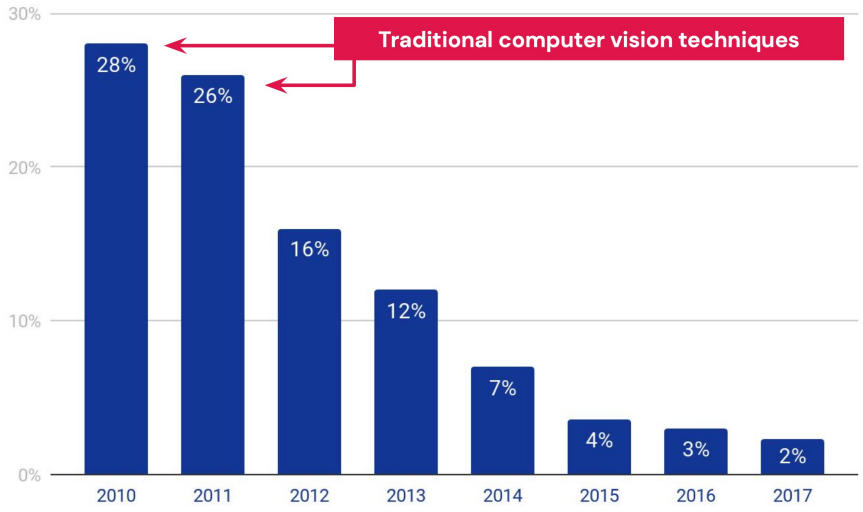
Other Dropout variations

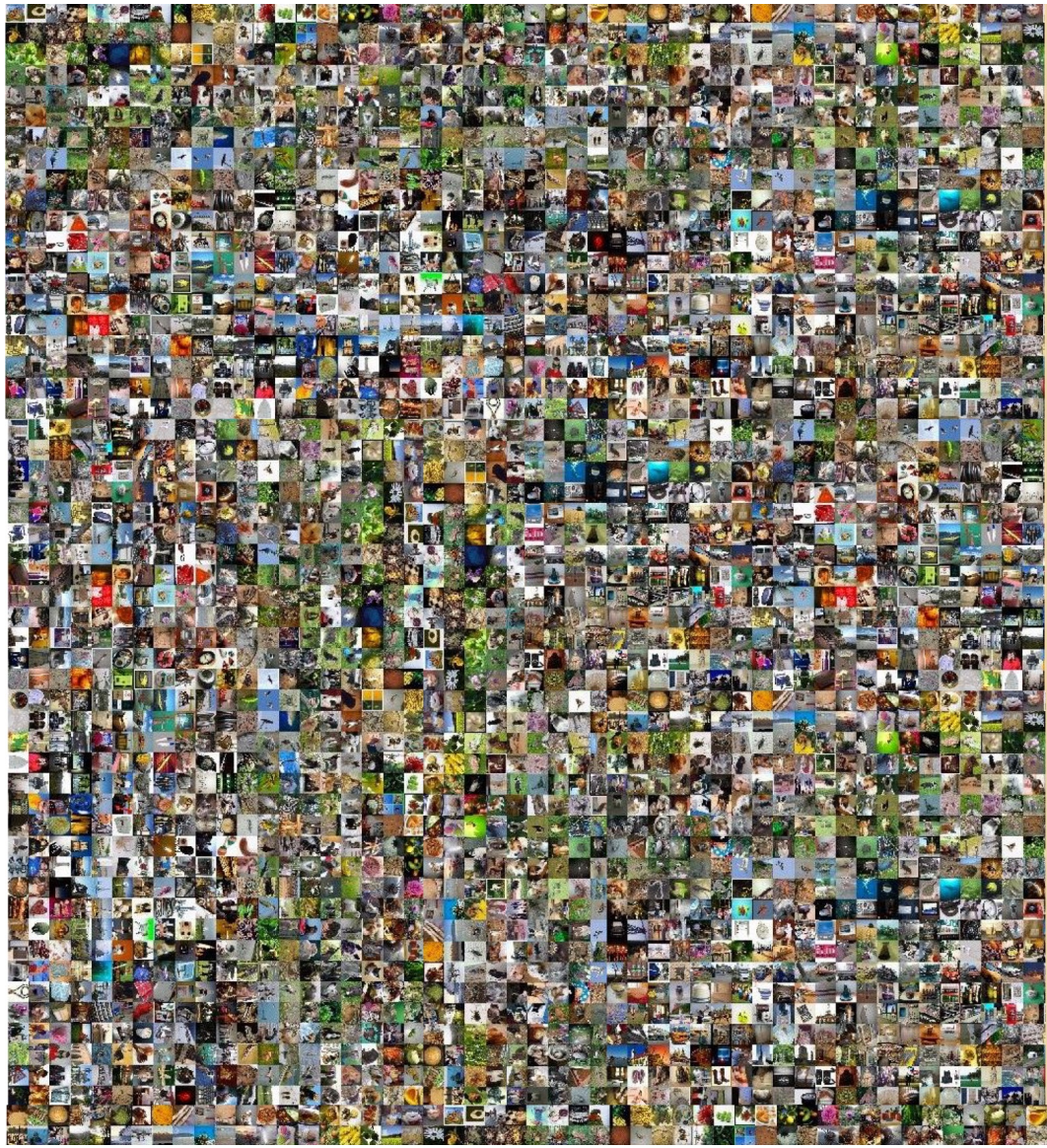
- **Inverted dropout** also randomly drops some units with a probability p. During training, it also scales the activations by the inverse of the keep probability 1−p: to prevent the activations from becoming too large thus the need to modify the network during the testing phase.
- **Gaussian dropout**: is injecting noise to the weights of each unit.
- **DropConnect**: Instead of zeroing out random activations (units), it zeros random weights during each forward pass. The weights are dropped with a probability of 1−p. This essentially transforms a fully connected layer to a sparsely connected layer. DropConnect can be seen as a generalization of Dropout to the full-connection structure of a layer.
- **Variational Dropout**: we use the same dropout mask on each timestep. This means that we will drop the same network units each time. This was initially introduced for Recurrent Neural Networks
- **Attention Dropout**: randomly dropped certain attention units with a probability p
- **Embedding Dropout**: a strategy that performs dropout on the embedding matrix and is used for a full forward and backward pass.
- **DropBlock**: is used in Convolutional Neural networks and it discards all units in a continuous region of the feature map.

Convolutional Neural Networks

Convolutional Neural Networks

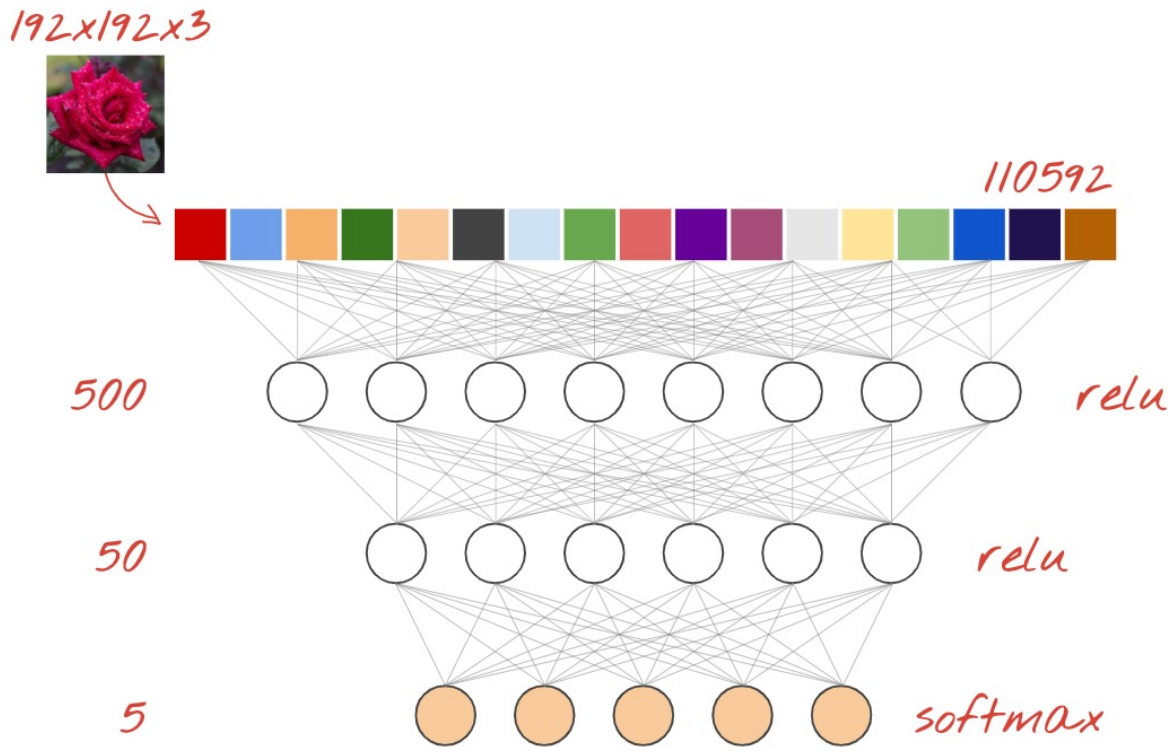
- Neural networks for images
  - A digital image is a 2D grid of pixels.
  - A neural network expects a vector of numbers as input.
- The ImageNet challenge
  - Major computer vision benchmark from 2010 to 2017
  - 1.4M images, 1000 classes, image classification





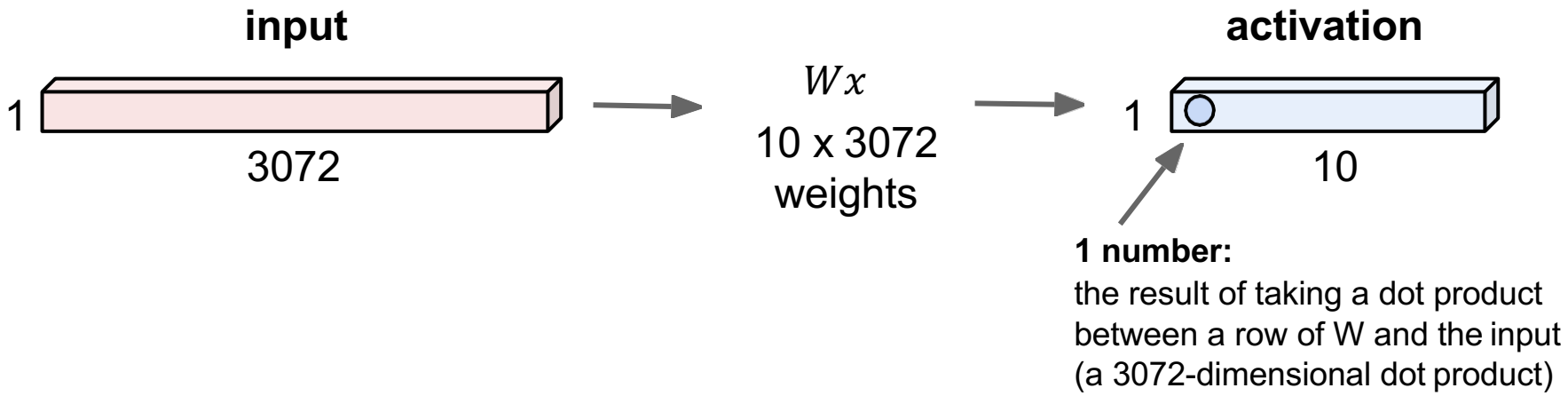
Convolutional Neural Networks

- Fully-connected structure does not scale to larger images
  - An image of more respectable size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights
  - Full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.



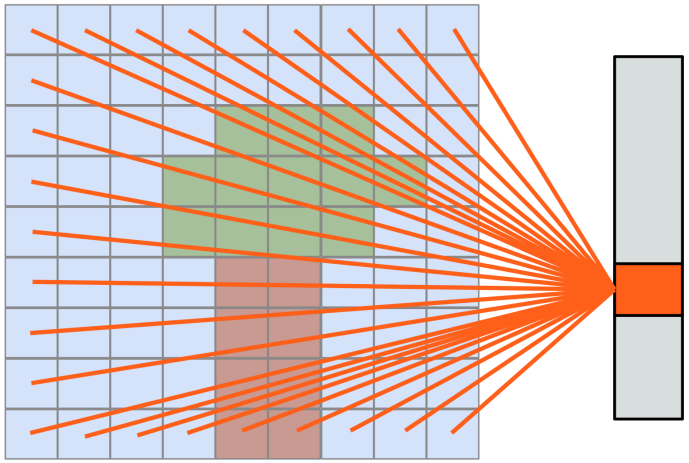
Fully Connected Layer

- Fully Connected Layer
  - 32x32x3 image -> stretch to 3072 x 1
  - the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)



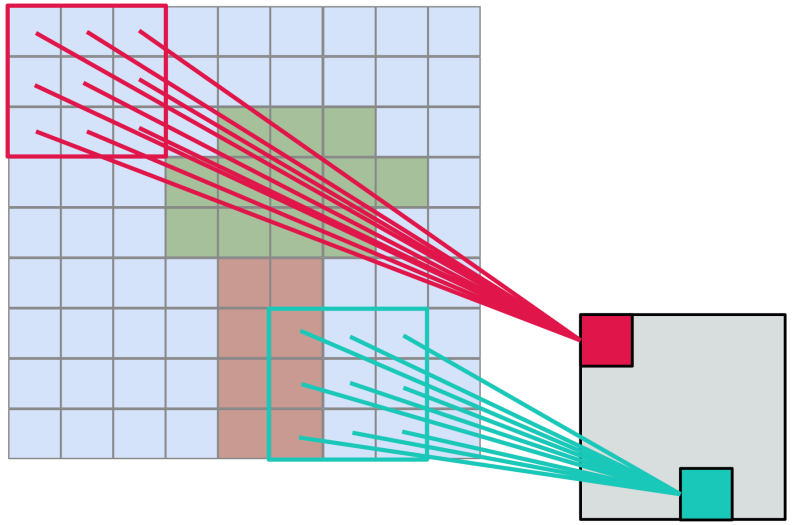
Convolutional Neural Networks

- From fully connected to locally connected
  - Locality: nearby pixels are more strongly correlated
  - Translation invariance: meaningful patterns can occur anywhere in the image



fully-connected unit

$$y = \sum_{i \in \text{image}} \mathbf{w}_i \mathbf{x}_i + b$$

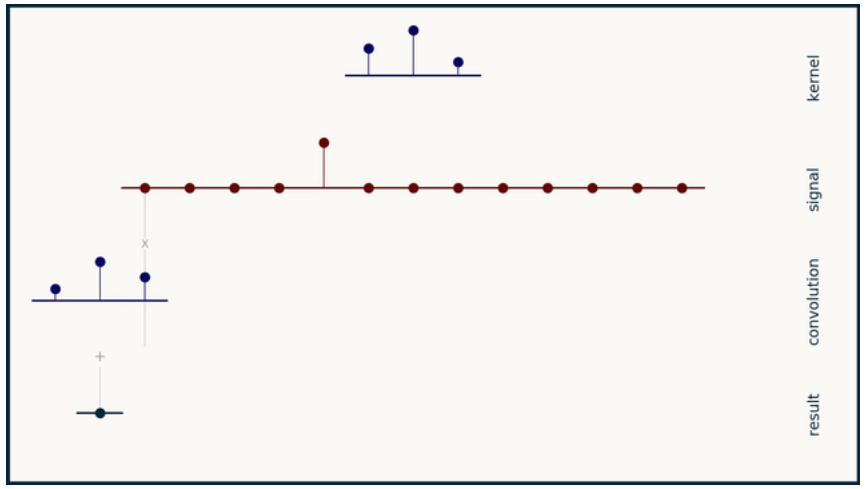


$$y = \sum_{i \in 3 \times 3} \mathbf{w}_i \mathbf{x}_i + b$$

locally-connected units
3×3 receptive field

Convolutional Neural Networks

- 1D Convolution
  - It endows the neural network with the ability to recognize particular patterns within the pixels
  - Convolution is a good way to identify patterns in data that is directly tied to space or time.
  - In **cross-correlation** the kernel is not flipped left-to-right before calculating the sliding dot product
  - You can cross-correlate a signal with itself, and the result is called an autocorrelation



$$y_j = \sum_{k=-p}^{p} x_{j-k} w_k$$

$$y_0 = \sum_{k=-p}^{p} x_{-k} w_k$$

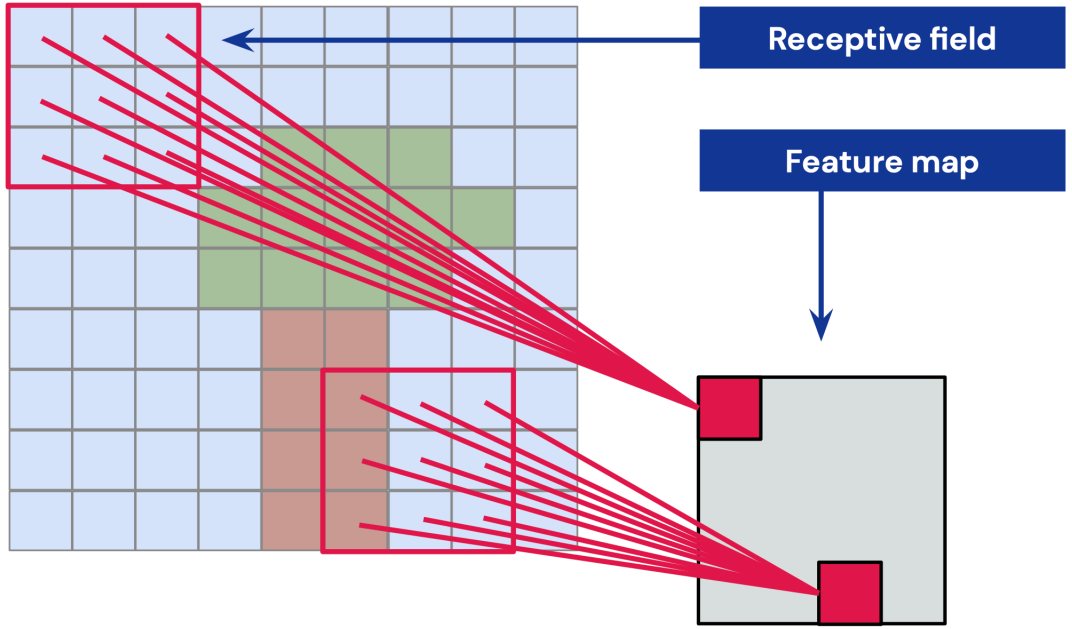
$$y_1 = \sum_{k=-p}^{p} x_{1-k} w_k$$

$$\vdots$$

$$y_m = \sum_{k=-p}^{p} x_{m-k} w_k$$

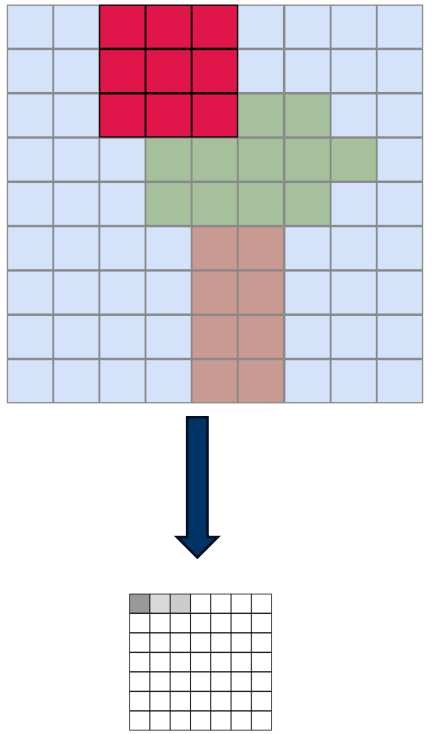
Convolutional Neural Networks

- From locally connected to convolutional
  - The kernel slides across the image and produces an output value at each position



$$y = \mathbf{w} * \mathbf{x} + b$$

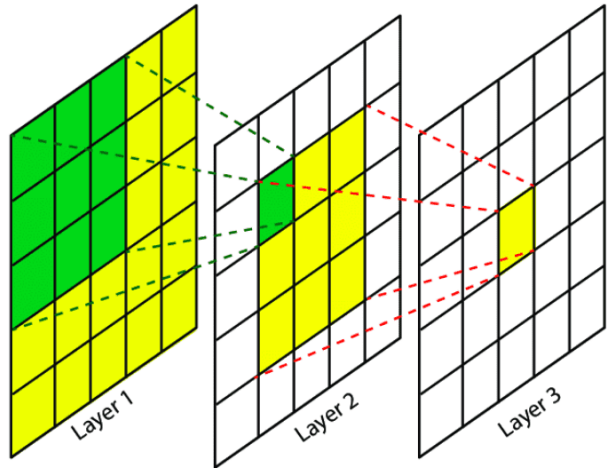
convolutional units
3×3 receptive field

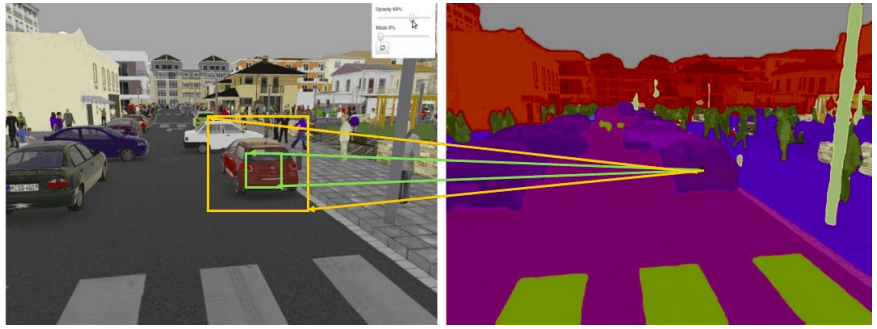


The kernel slides across the image and produces an output value at each position

Receptive Field

- The human visual system consists of millions of neurons, where each one captures different information. We define the neuron's receptive field as the patch of the total field of view. In other words, what information a **single neuron** has access to. This is in simple terms the biological cell's **receptive field**.
- In a deep learning context, the **Receptive Field (RF)** is defined as the size of the region in the input that produces the feature
  - it is a measure of association of an output feature (of any layer) to the input region (patch)
  - A convolutional unit only depends on a local region (patch) of the input. That's why we never refer to the RF on fully connected layers since each unit has access to all the input region.

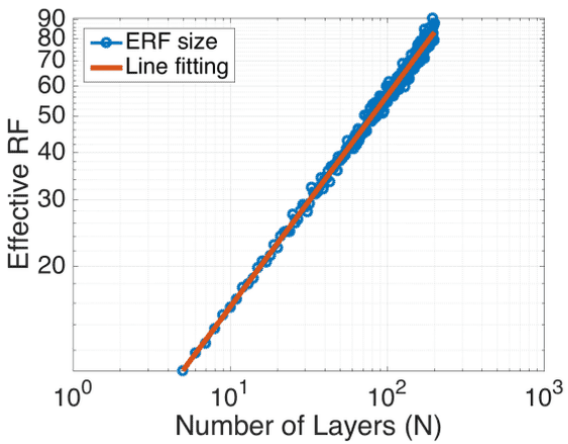


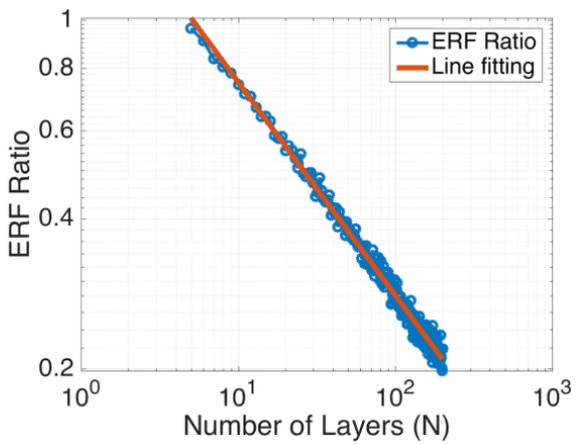


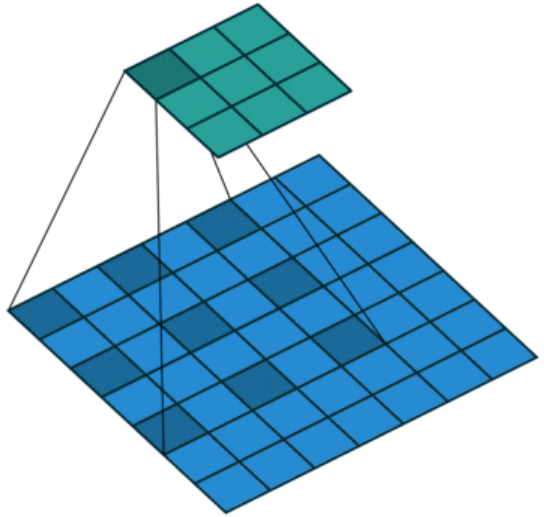
Receptive Field

- How can we increase the receptive field in a convolutional network?
  - Add more convolutional layers (make the network deeper)
  - Add pooling layers or higher stride convolutions (sub-sampling)
  - Use dilated convolutions
    - a 3x3 kernel with a dilation rate of 2 will have the same receptive field as a 5x5 kernel, while only using 9 parameters.
    - a 3x3 kernel with a dilation rate of 4 will have the same receptive field as a 9x9 kernel without dilation

$$r(k-1)+1=k_{prev}$$

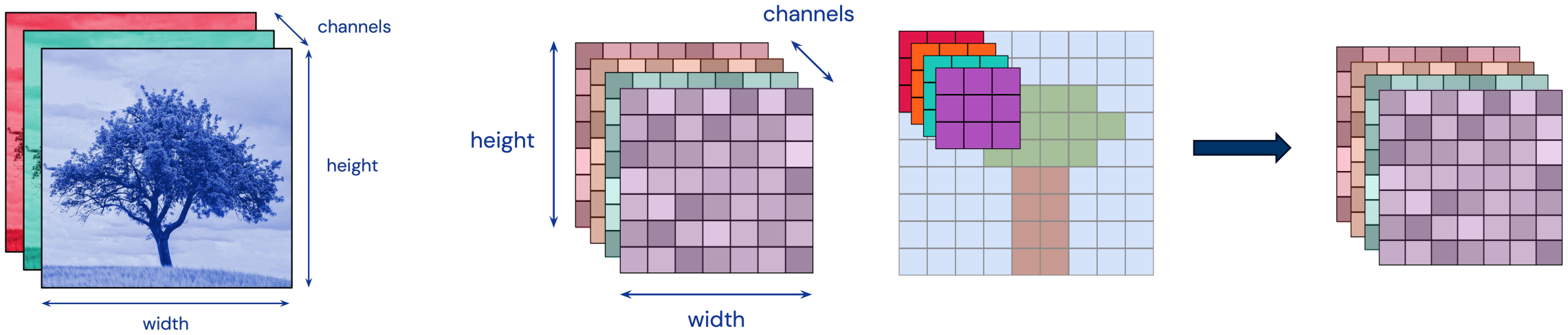




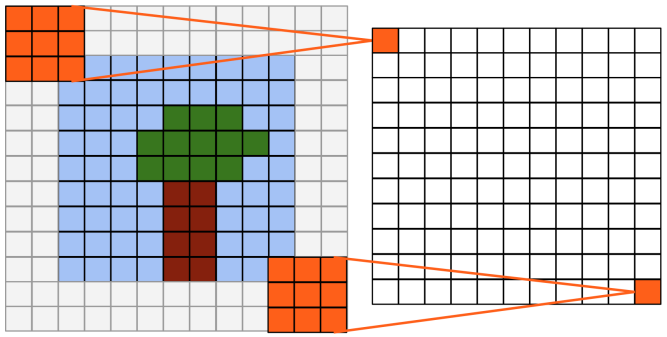


Convolutional Neural Networks

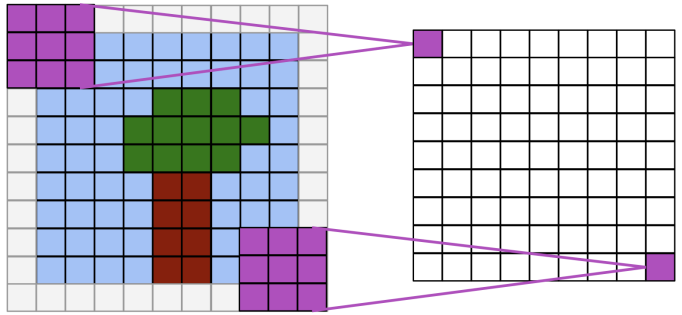
- Valid convolution: output size = input size - kernel size + 1



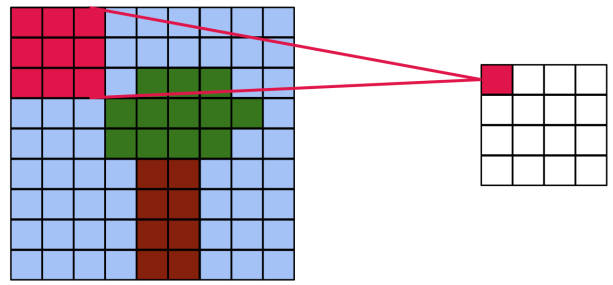
- Variants of the convolution operation



Full convolution: output size = input size + kernel size - 1



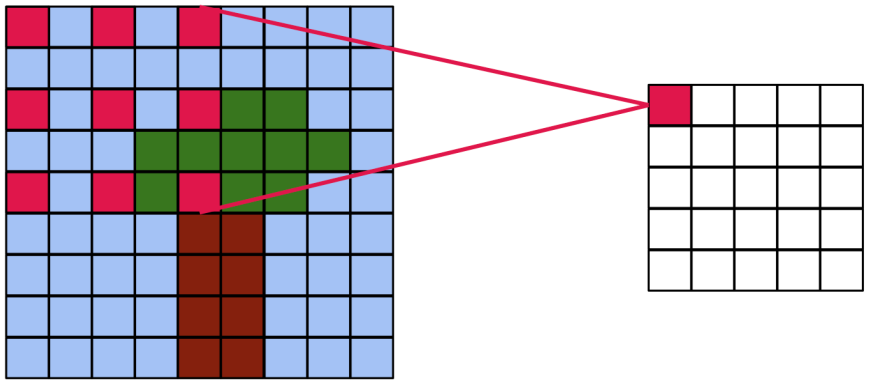
Same convolution: output size = input size



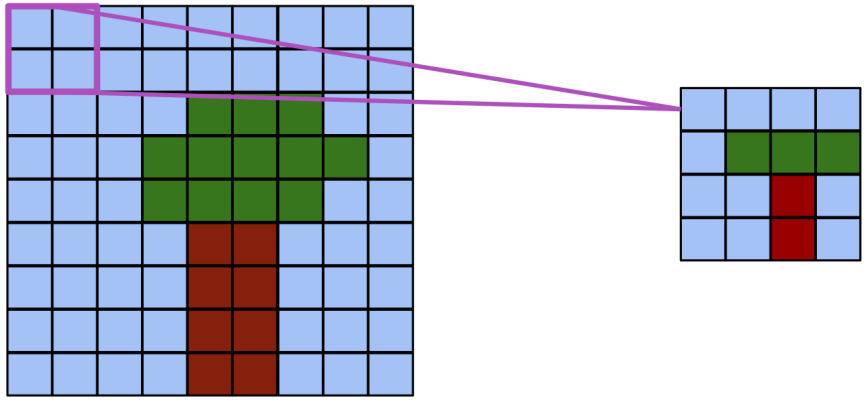
Strided convolution: kernel slides along the image with a step > 1

Convolutional Neural Networks

- Dilated convolution: kernel is spread out, step > 1 between kernel elements

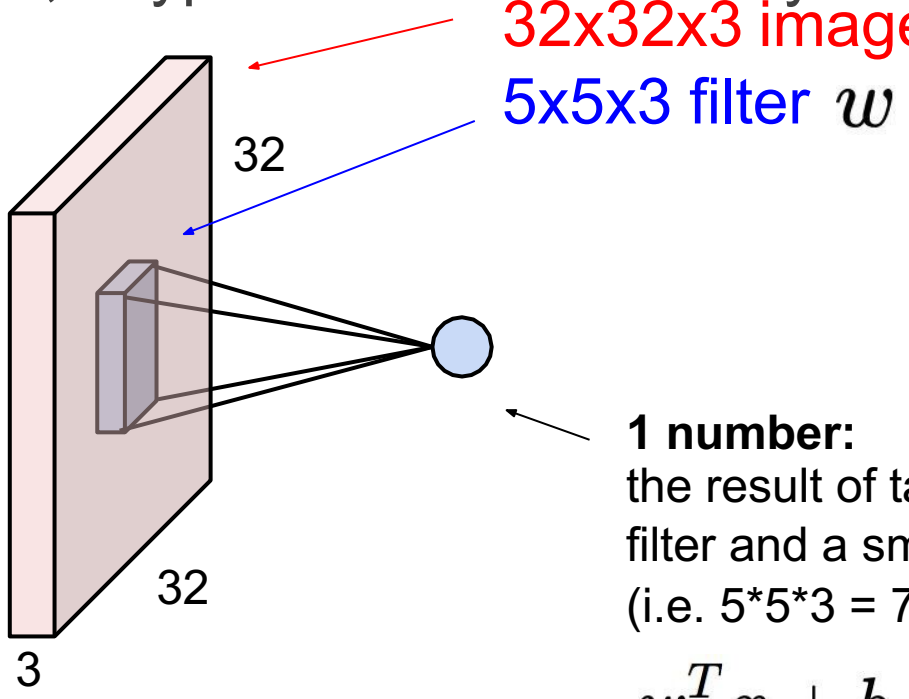


- Pooling: compute mean or max over small windows to reduce resolution



Convolutional Neural Networks

- Convolutional Layer
  - The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
  - For example, a typical filter on a first layer of a ConvNet might have size 5x5x3

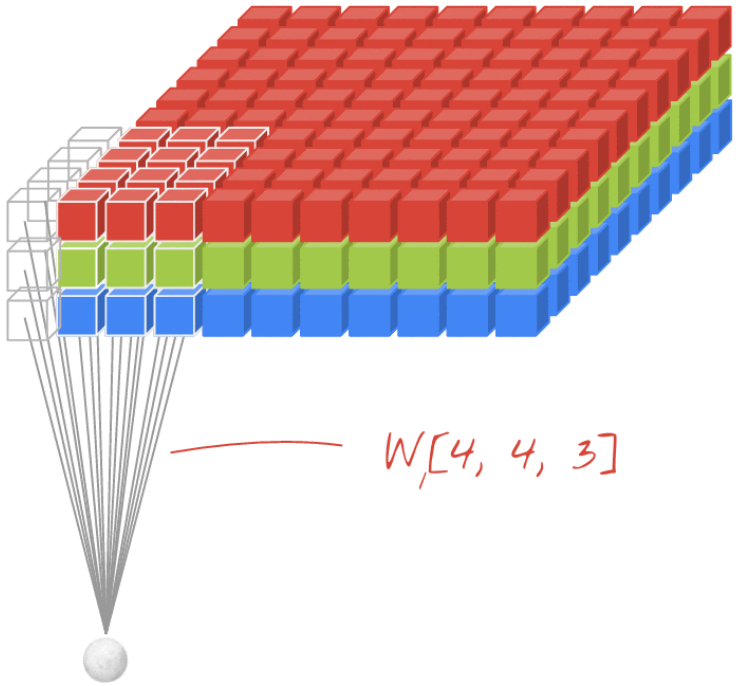


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolutional Neural Networks

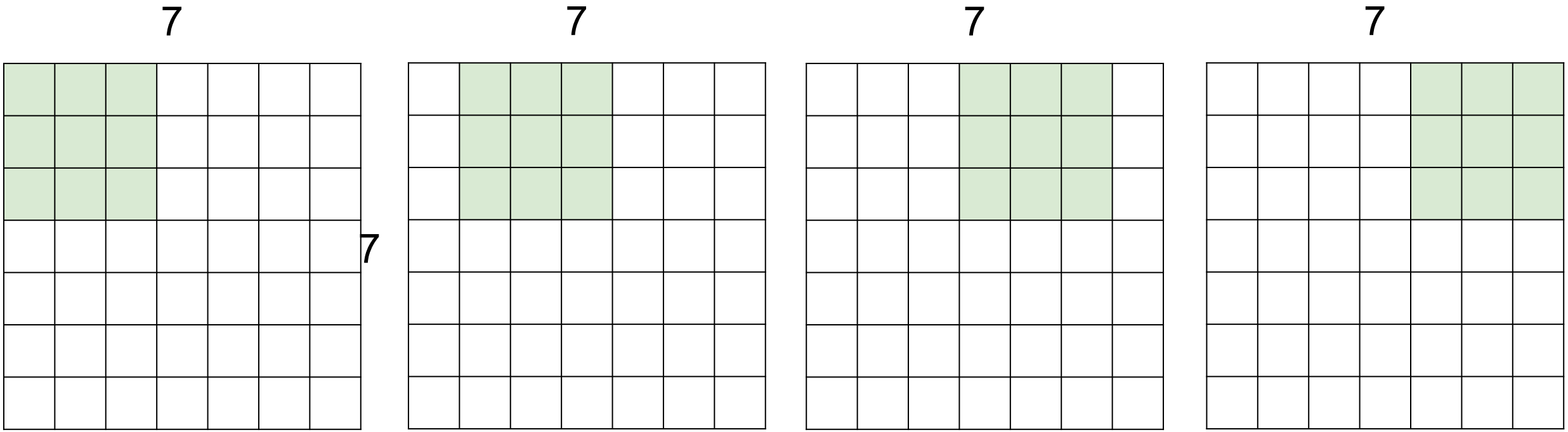
- Convolutional Neural Networks (CNNs / ConvNets)
  - ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture.
    - Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way
    - These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network



filtering an image with two successive filters made of 4x4x3=48 learnable weights each.

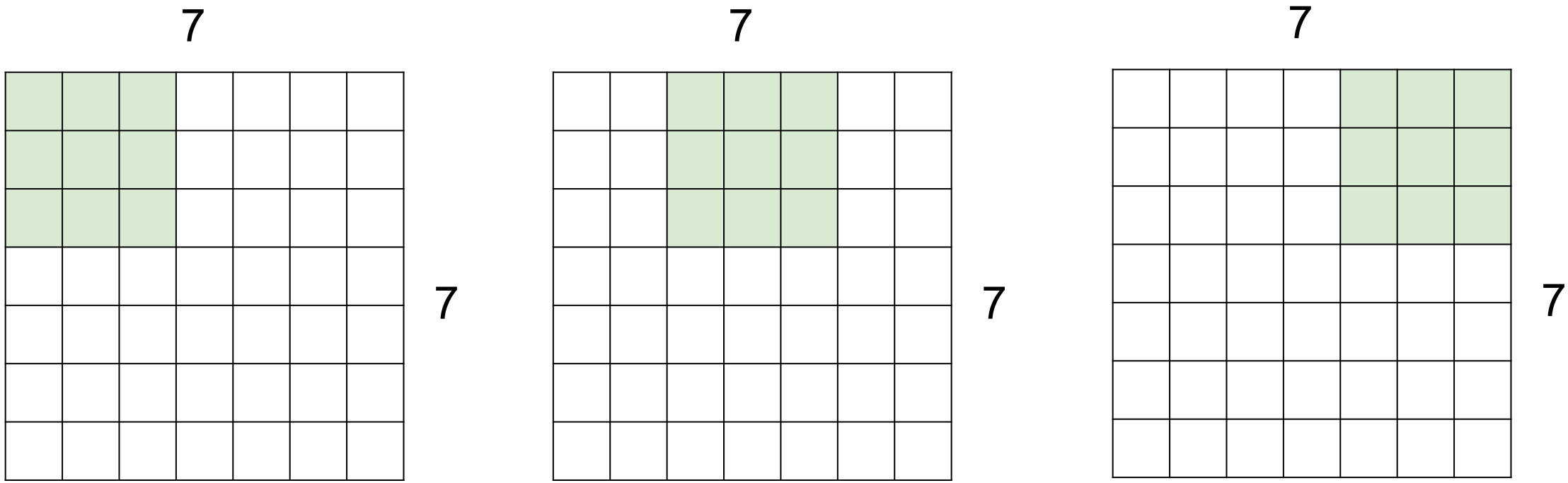
Convolutional Neural Networks

- 7x7 input (spatially) assume 3x3 filter
  - **=> 5x5 output**



Convolutional Neural Networks

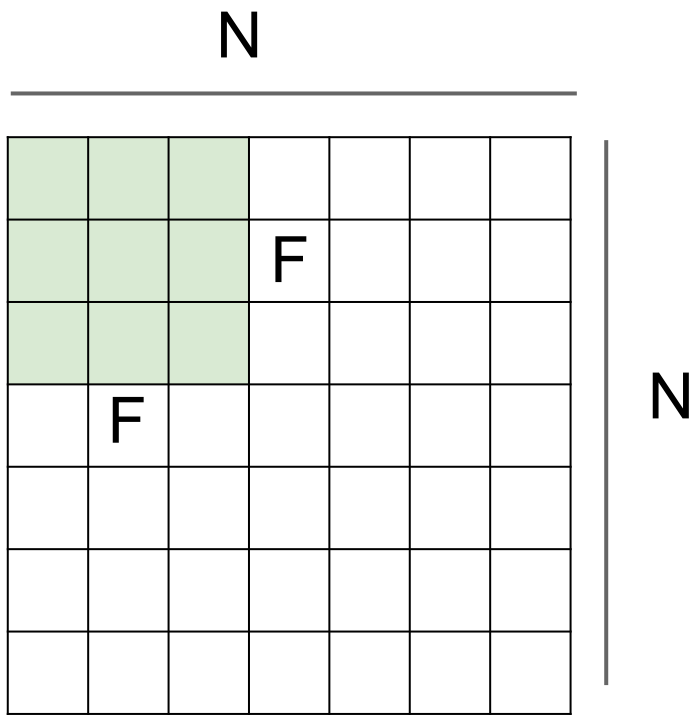
- 7x7 input (spatially) assume 3x3 filter applied **with stride 2**
  - **=> 3x3 output!**



Convolutional Neural Networks

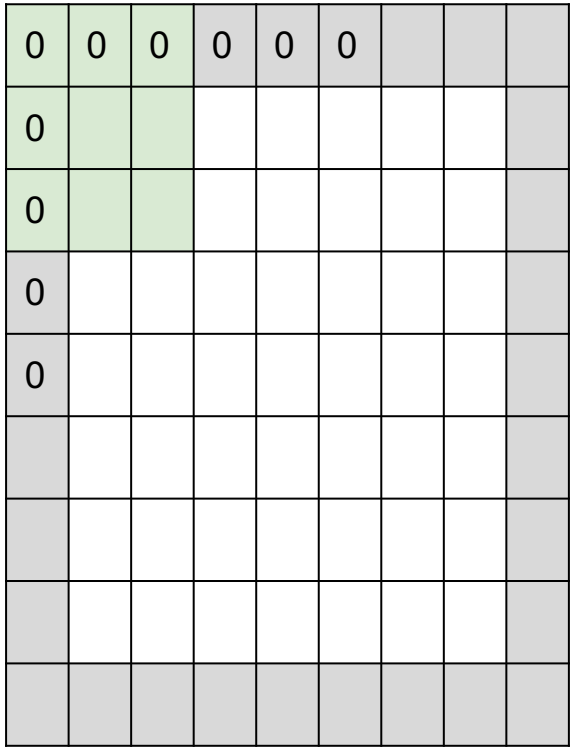
- Output size:
- **(N - F) / stride + 1**

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 :\



Convolutional Neural Networks

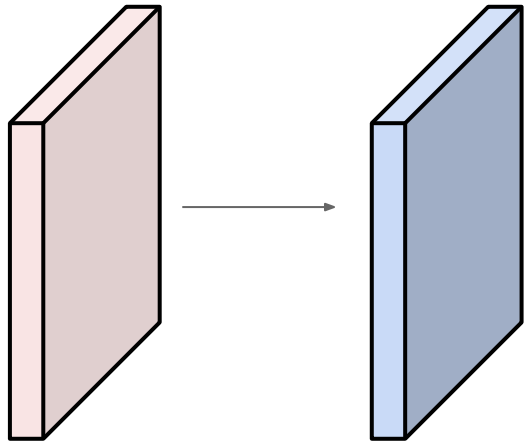
- In practice: Common to zero pad the border
- e.g. input 7x7
- **3x3** filter, applied with **stride 1**
- **pad with 1 pixel** border => what is the output?



- **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

Convolutional Neural Networks

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



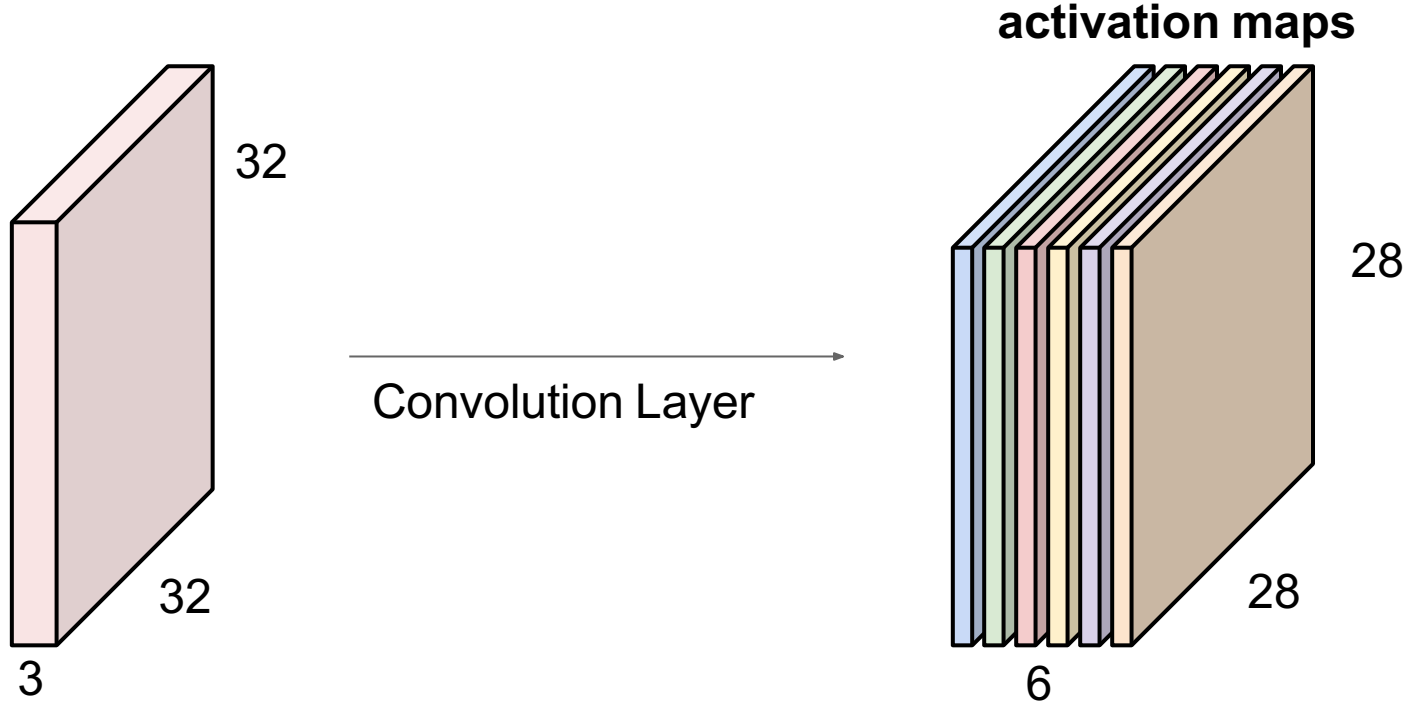
- Output volume size:
- (32+2*2-5)/1+1 = 32 spatially, so
- **32x32x10**
- Number of parameters in this layer? each filter has 5*5*3 + 1 (+1 for bias) = 76 params
- => 76*10 = **760**

Convolutional Neural Networks

- Convolve (slide) over all spatial locations
- 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!

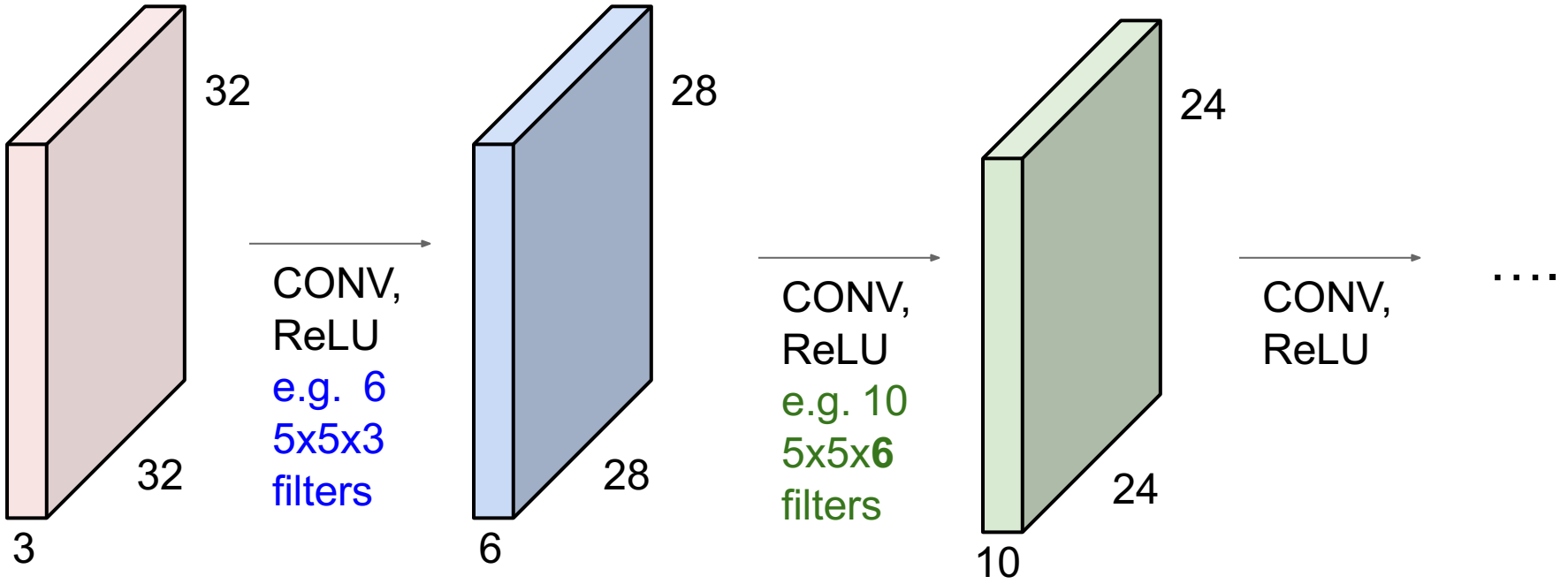
Convolutional Neural Networks

- convolve (slide) over all spatial locations
- 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!



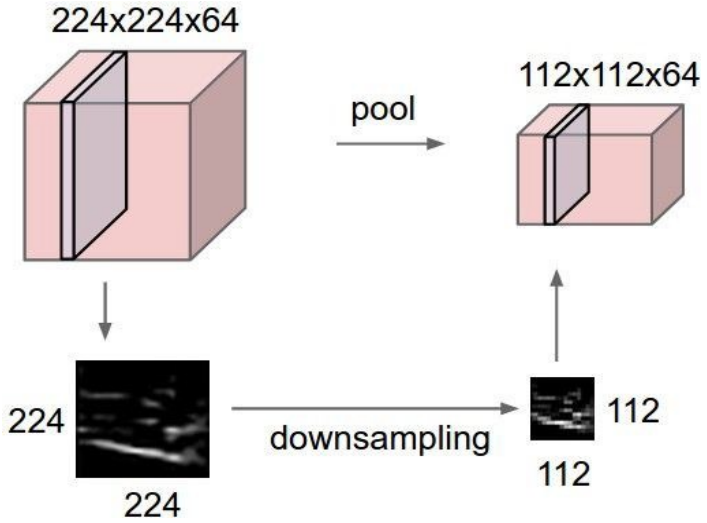
Convolutional Neural Networks

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

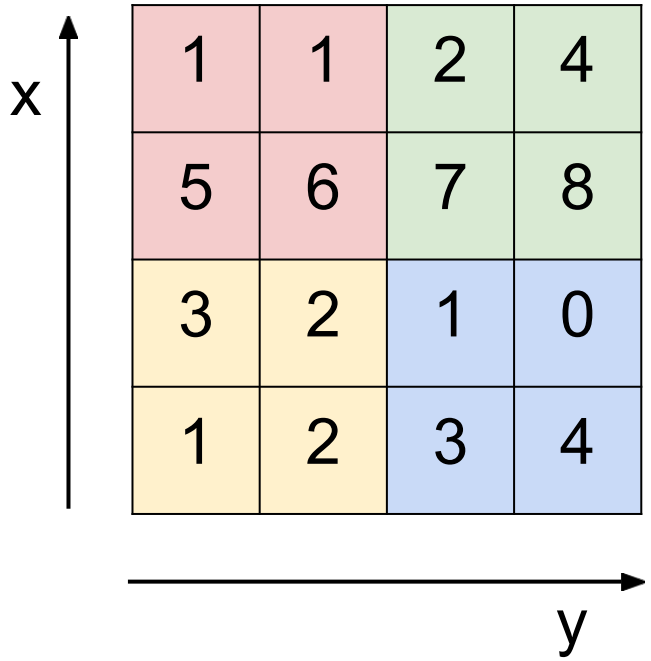


Convolutional Neural Networks

- Pooling layer
  - makes the representations smaller and more manageable
  - operates over each activation map independently:
  - MAX POOLING



Single depth slice

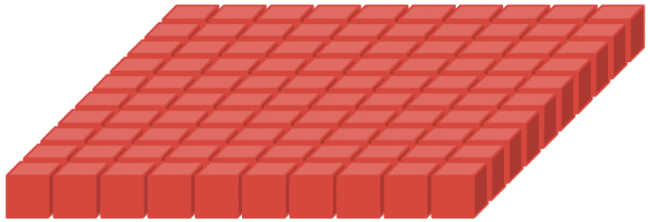


max pool with 2x2 filters and stride 2

6	8
3	4

Convolutional Neural Networks

- Convolutional neural nets
  - Max pooling: a sliding window applying the MAX operation (typically on 2x2 patches, repeated every 2 pixels)



sliding the computing window by 3 pixels results in fewer output values. Strided convolutions or max pooling (max on a 2x2 window sliding by a stride of 2).

Convolutional Neural Networks

- Torch.nn.conv2d: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

CLASS
torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *pad ding_mode='zeros'*, *device=None*, *dtype=None*)

- In the simplest case, the output value of the layer with input size (N,Cin,H,W) and output (N,Cout,Hout,Wout) can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

  - N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

Convolutional Neural Networks

- Torch.nn.conv2d: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

  CLASS

  torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*)

- **stride** controls the stride for the cross-correlation
- **padding** controls the amount of padding applied to the input. It can be either a string {'valid', 'same'} or an int / a tuple of ints giving the amount of implicit padding applied on both sides.
  - padding='valid' is the same as no padding. padding='same' pads the input so the output has the shape as the input. However, this mode doesn't support any stride values other than 1.
- **dilation** controls the spacing between the kernel points
- **groups** controls the connections between inputs and outputs. in_channels and out_channels must both be divisible by groups
  - At groups=1, all inputs are convolved to all outputs.
  - When groups == in_channels and out_channels == K * in_channels, where K is a positive integer, this operation is also known as a "depthwise convolution".

Convolutional Neural Networks

- Torch.nn.conv2d: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

CLASS
torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*)

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Convolutional Neural Networks

- MAXPOOL2D
  - https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d

CLASS torch.nn.MaxPool2d(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*, *return_indices=False*, *ceil_mode=False*)

- In the simplest case, the output value of the layer with input size (*N*,*C*,*H*,*W*), output (*N*,*C*,*Hout*,*Wout*) and kernel_size (*kH*,*kW*) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

Convolutional Neural Networks

- MAXPOOL2D
  - https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d

CLASS torch.nn.MaxPool2d(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*, *return_indices=False*, *ceil_mode=False*)

- Applies a 2D max pooling over an input signal composed of several input planes.
- If padding is non-zero, then the input is implicitly padded with negative infinity on both sides for padding number of points. dilation controls the spacing between the kernel points.

- **kernel_size** (*Union[int, Tuple[int, int]]*) – the size of the window to take a max over
- **stride** (*Union[int, Tuple[int, int]]*) – the stride of the window. Default value is kernel_size
- **padding** (*Union[int, Tuple[int, int]]*) – Implicit negative infinity padding to be added on both sides
- **dilation** (*Union[int, Tuple[int, int]]*) – a parameter that controls the stride of elements in the window
- **return_indices** (*bool*) – if True, will return the max indices along with the outputs. Useful for torch.nn.MaxUnpool2d later
- **ceil_mode** (*bool*) – when True, will use *ceil* instead of *floor* to compute the output shape

Convolutional Neural Networks

- MAXPOOL2D
  - https://pytorch.org/docs/stable/generated/torch.nn.MaxPool2d.html#torch.nn.MaxPool2d

CLASS torch.nn.MaxPool2d(*kernel_size*, *stride=None*, *padding=0*, *dilation=1*, *return_indices=False*, *ceil_mode=False*)

- Input: (N, C, H_{in}, W_{in}) or (C, H_{in}, W_{in})
- Output: (N, C, H_{out}, W_{out}) or (C, H_{out}, W_{out}), where

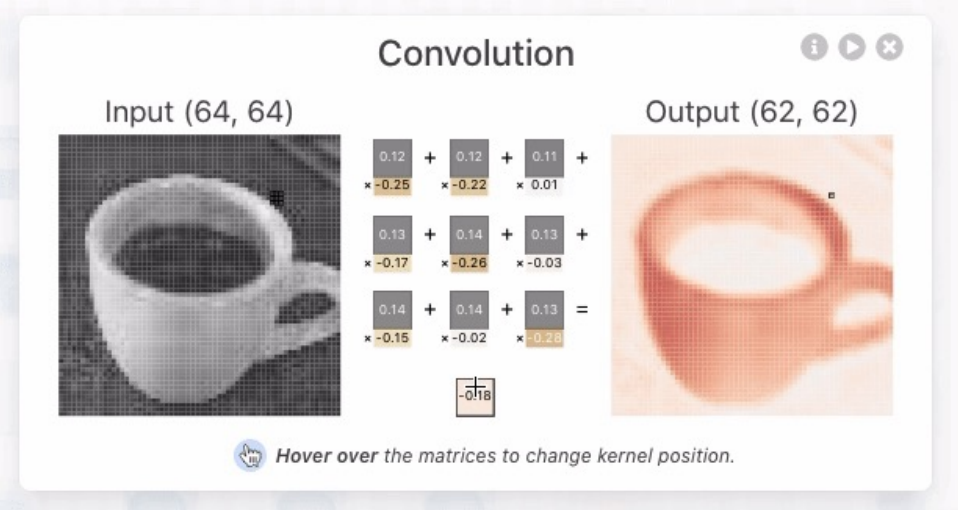
$$H_{out} = \left\lfloor \frac{H_{in} + 2 * \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

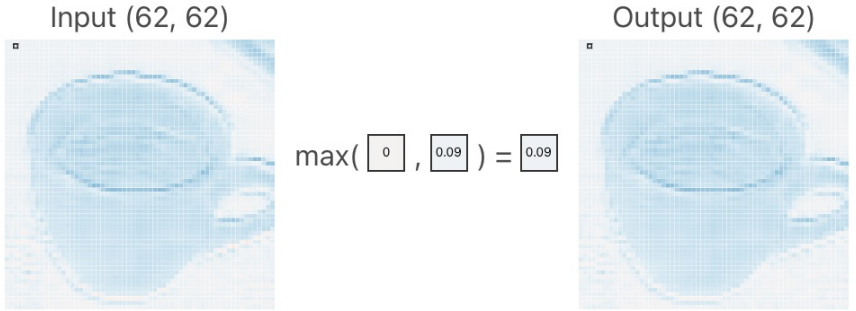
$$W_{out} = \left\lfloor \frac{W_{in} + 2 * \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

```
# pool of non-square window
m = nn.MaxPool2d((3, 2), stride=(2, 1))
```

Convolutional Neural Networks

- Convolve multiple kernels and obtain multiple feature maps or channels
  - The size of kernels is a hyper-parameter specified by the designers of the network architecture.



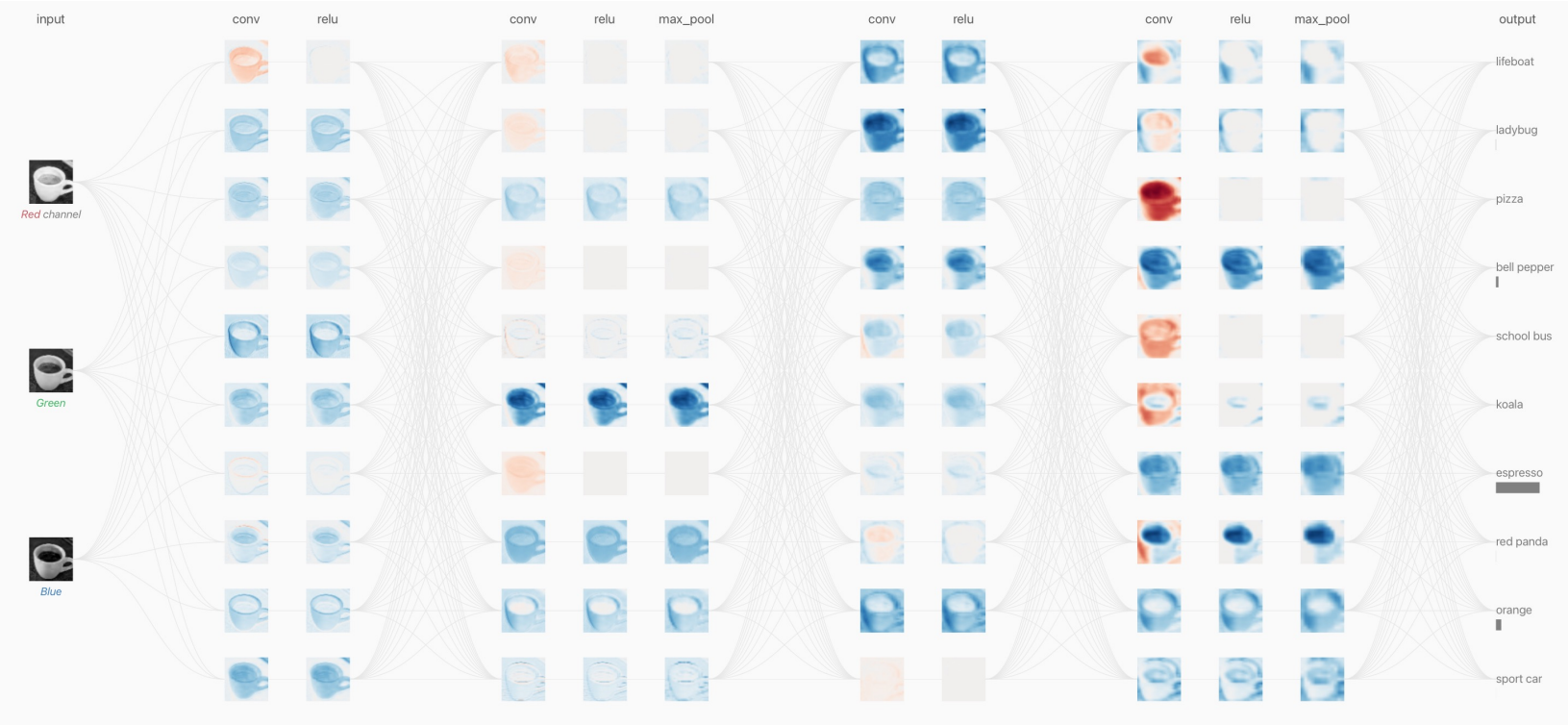


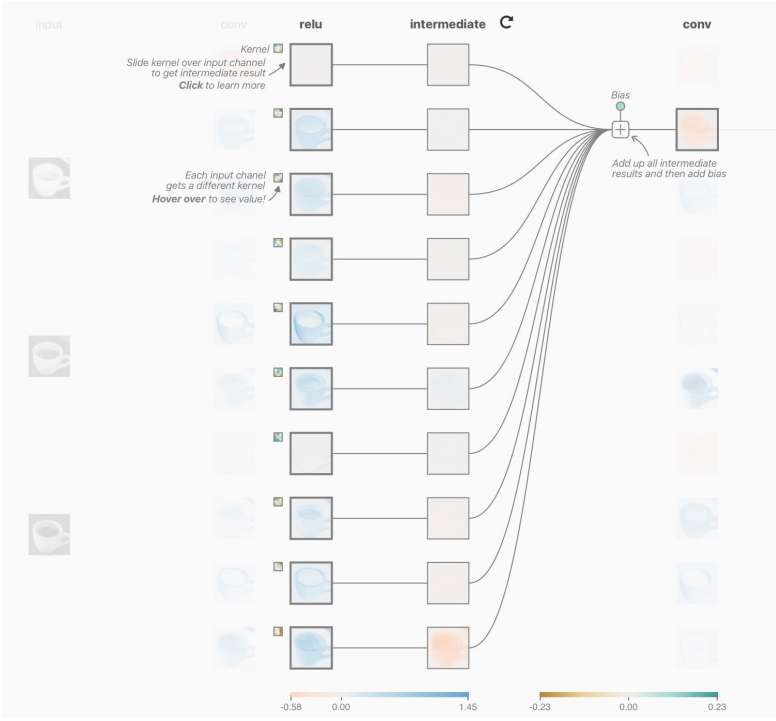
The ReLU activation function is specifically used as a non-linear activation function

https://poloclub.github.io/cnn-explainer/

Convolutional Neural Networks

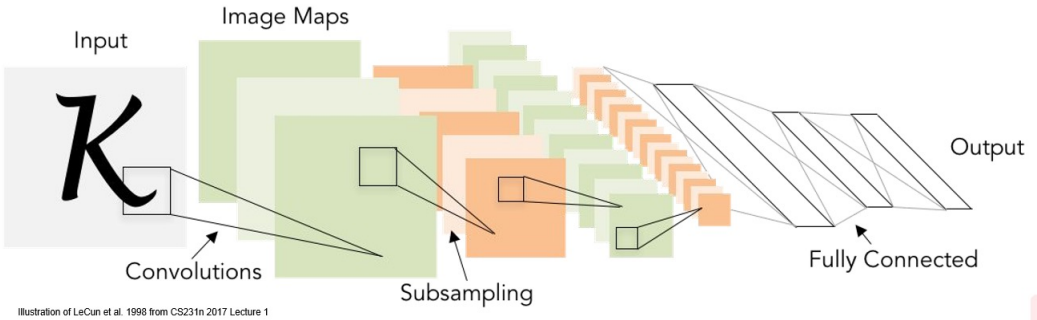
- CNN Explainer: https://poloclub.github.io/cnn-explainer/
- CNN 3D visualization: https://adamharley.com/nn_vis/cnn/3d.html

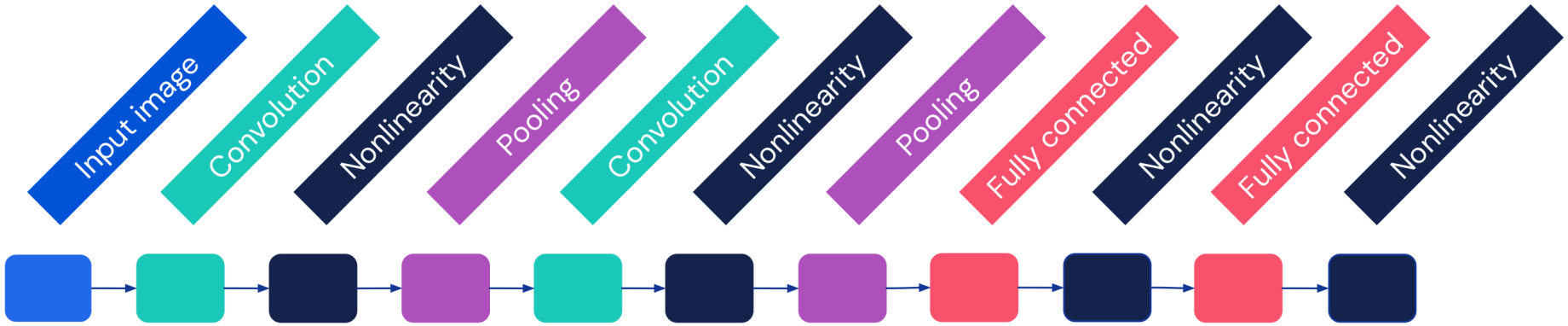




LeNet-5 (1998)

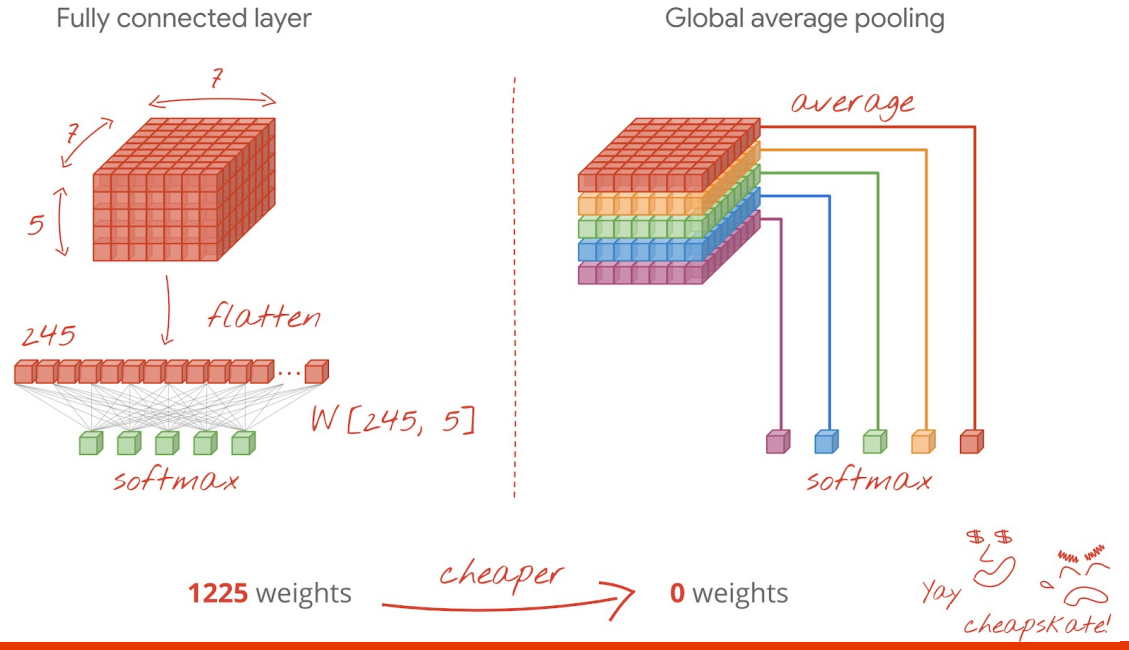
- Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. **Gradient-based learning applied to document recognition**, Proceedings of the IEEE 86(11) (1998)
  - Three main types: Convolutional Layer, Pooling Layer, and Fully-Connected Layer
  - Stack these layers to form a full ConvNet architecture





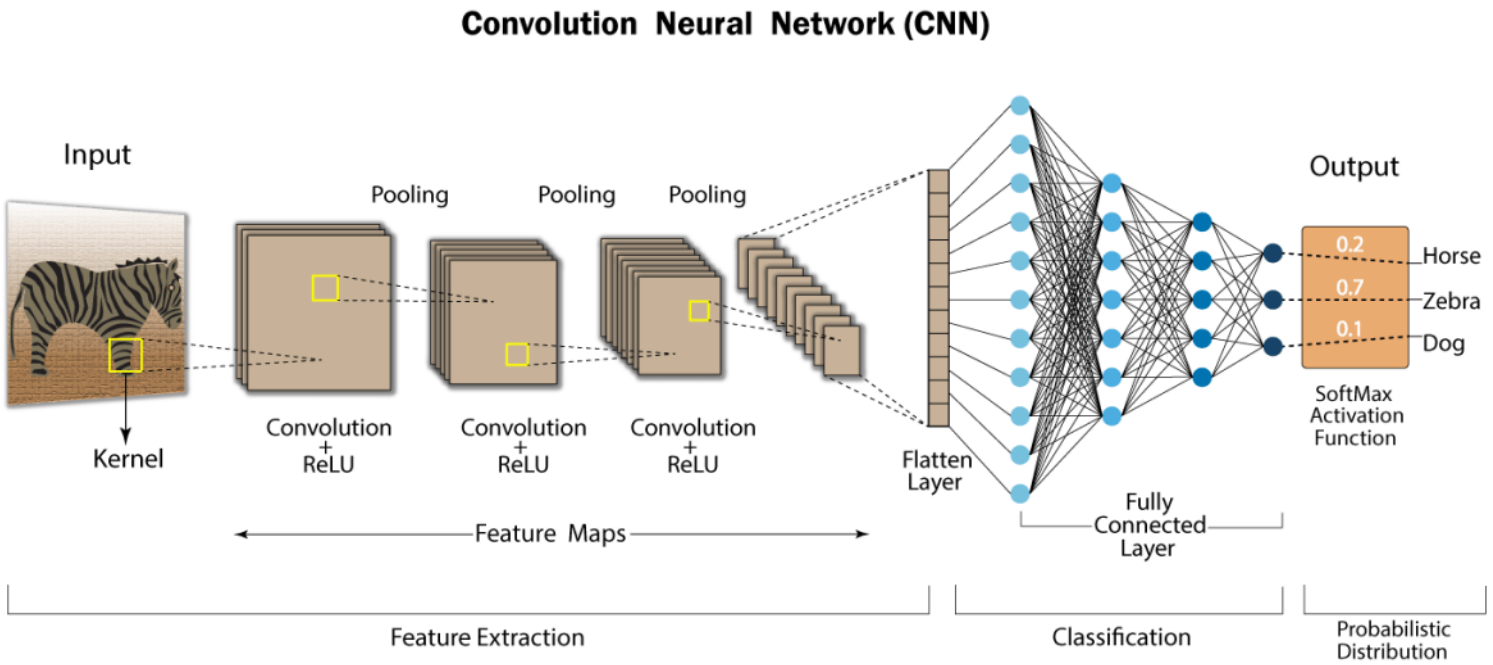
Convolutional Neural Networks

- Global average pooling
  - Instead of using an expensive dense layer at the end of a convolutional neural network, you can split the incoming data "cube" into as many parts as you have classes, average their values and feed these through a softmax activation function. This way of building the classification head costs 0 weights.
    - In Keras, the syntax is tf.keras.layers.GlobalAveragePooling2D()



Transfer Learning

- Transfer learning
  - Transfer learning allows you to take a trained model and re-train it to perform another task.
    - For example, an image classification model could be retrained to recognize new categories of image. Re-training takes less time and requires less data than training a model from scratch.



Thank You

Address:
ENG257, SJSU

Email Address:
Kaikai.liu@sjsu.edu