Neural Networks

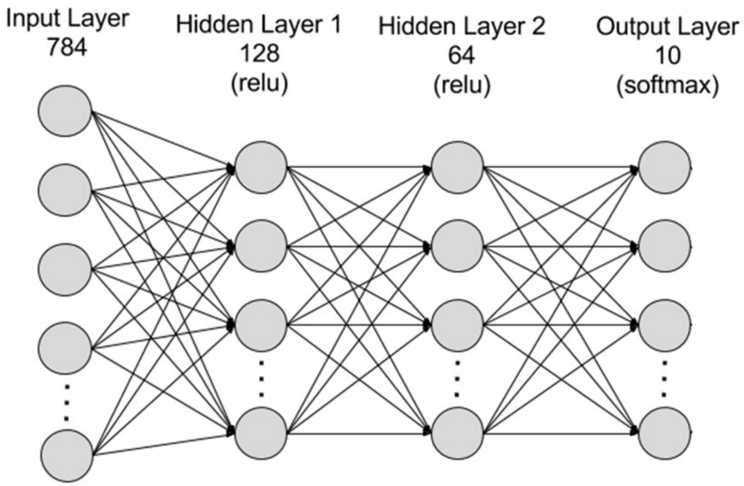
Dr. Jameel Malik

muhammad.jameel@seecs.edu.pk

Images to Digits - A Mapping from 784 to 10 Dimensions

28x28 pixel input images

(784 dim)



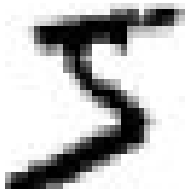
$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

output vector (10 dim)

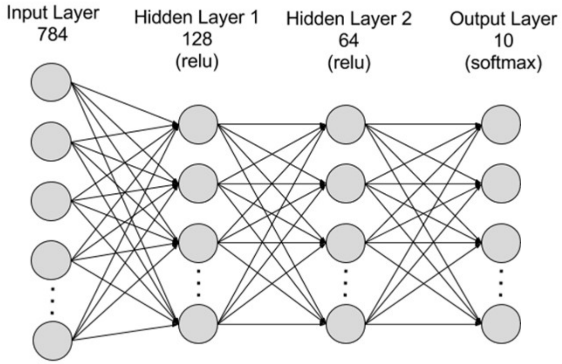
[Partial image courtesy: Nielsen]

Loss Function $L_i\left(\boldsymbol{\theta}\right) \mapsto \mathbb{R}$

$\boldsymbol{x}_i$



data point



network with parameters θ

$$\hat{\boldsymbol{y}}_i = f(\boldsymbol{\theta}, \boldsymbol{x}_i) \quad \begin{bmatrix} 0.2 \\ 0 \\ 0.3 \\ 0.2 \\ 0.2 \\ 0.9 \\ 0 \\ \vdots \end{bmatrix}$$

network output

$$\boldsymbol{y}_i \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

label

Compare output layer to the true label

$$L_i\left(\boldsymbol{\theta}\right) = ||\text{output}_i - \text{label}_i||^2$$

The Parameters We Want

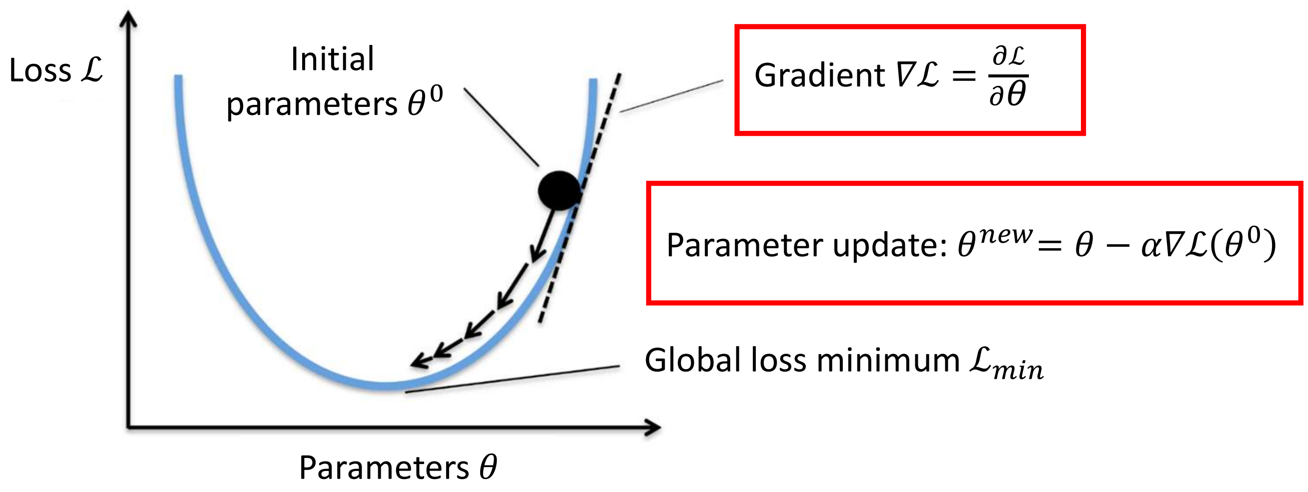
- Parameter $\boldsymbol{\theta}^*$ that minimize the sum of avg. squared losses over all examples

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_i ||f(\boldsymbol{\theta}, \boldsymbol{x}_i) - \boldsymbol{y}_i||^2$$

- The squared loss is only one possible loss, several other options available
- **Goal:** Find the parameter vector $\boldsymbol{\theta}^*$ for the labeled training set $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{I}$ given the loss L

Gradient Descent Algorithm -- Recap

- Steps in the ***gradient descent algorithm***:
  1. Randomly initialize the model parameters, θ^0
  2. Compute the gradient of the loss function at the initial parameters θ^0: $\nabla\mathcal{L}(\theta^0)$
  3. Update the parameters as: $\theta^{new} = \theta^0 - \alpha\nabla\mathcal{L}(\theta^0)$
     - Where α is the learning rate
  4. Go to step 2 and repeat (until a terminating criterion is reached)



Convolutional Neural Networks (CNNs)

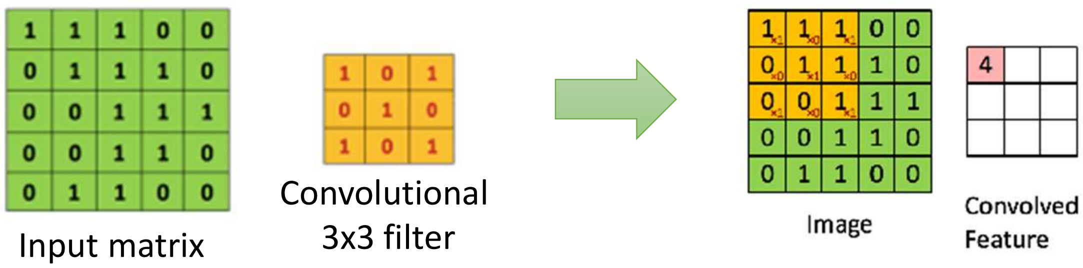
- *Convolutional neural networks* (CNNs) were primarily designed for image data
  - Capture the spatial/contextual information in images.

- CNNs have less parameters than MLPs (fully-connected layers)

- Example:
  - MLP sees or processes a flattened 1D vector (784 dimensions) of an MNIST image.
  - CNN sees or processes the original image of MNIST in 2D

Convolutional Neural Networks (CNNs)

- A convolutional filter slides (i.e., convolves) across the image

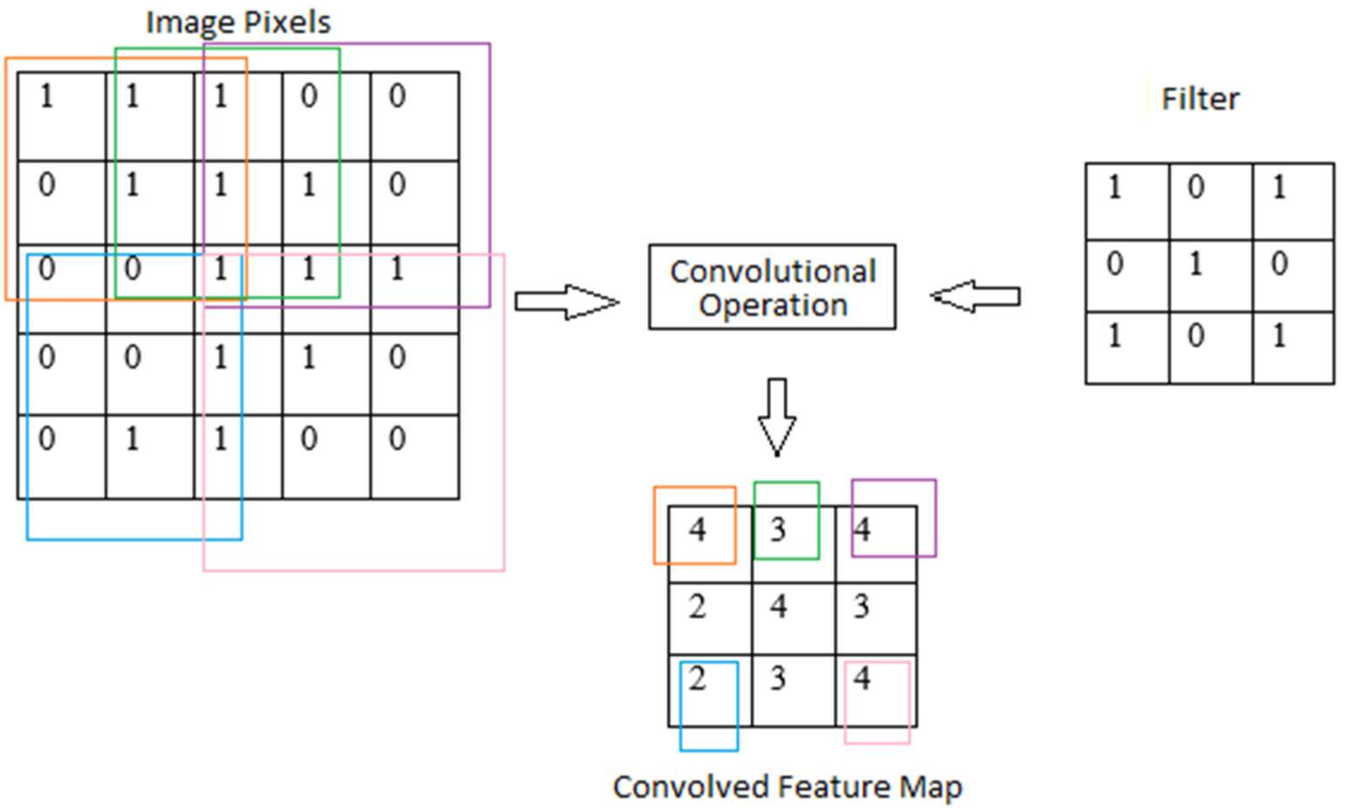


2D convolution

Picture from: http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

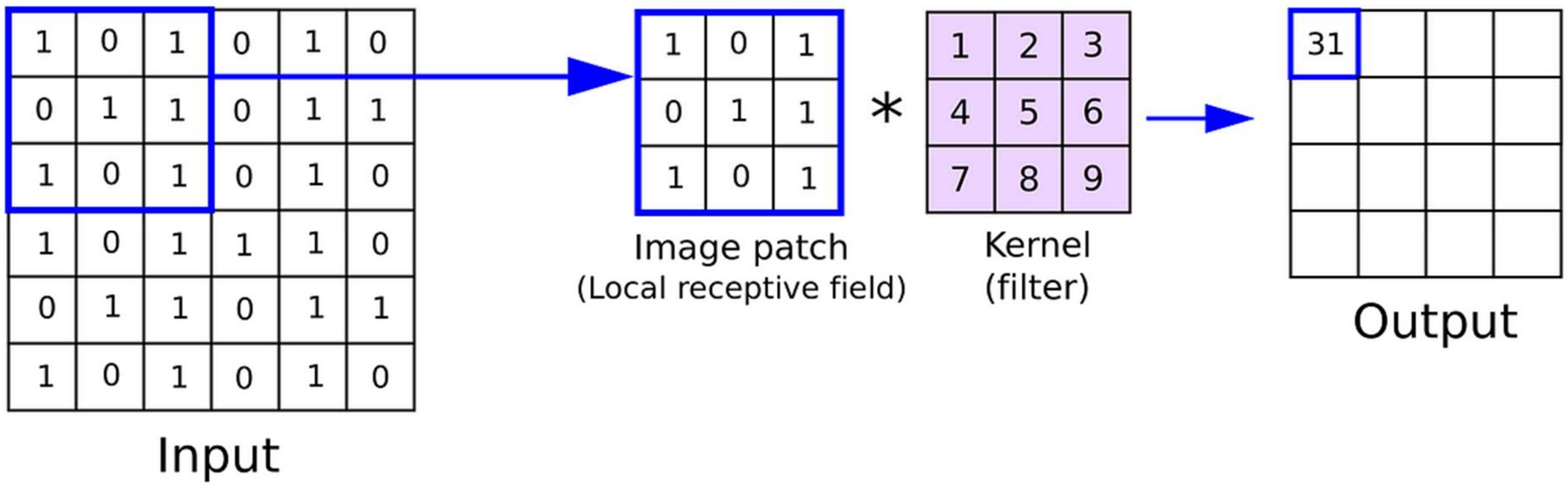
Convolutional Neural Networks (CNNs)

- A convolutional filter slides (i.e., convolves) across the image



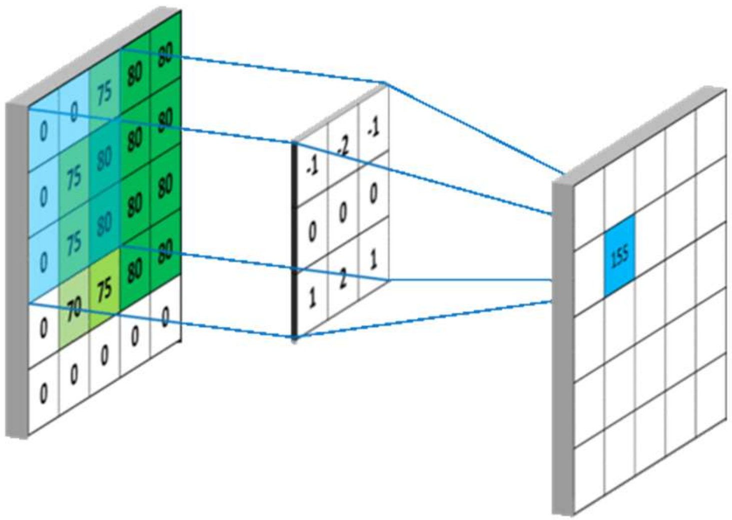
Convolutional Neural Networks (CNNs)

- A convolutional filter slides (i.e., convolves) across the image



2D Convolution

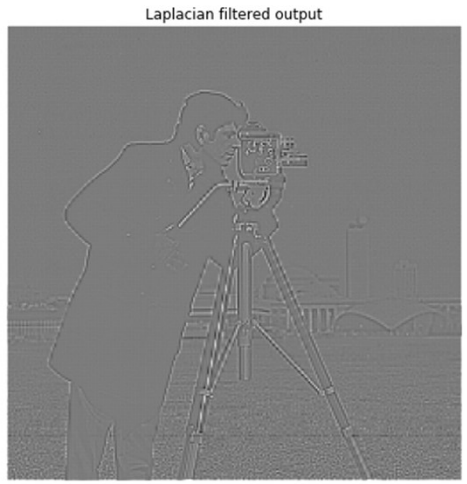
$$y[m,n] = x[m,n] \circledast h[m,n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j]$$



Laplacian Filter (3x3)

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$\circledast$

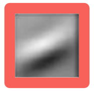


Convolution result
(activation/feature map)

Convolutional Neural Networks (CNNs)

Convolution Operation:

- When the convolutional filters are scanned over the image, they capture useful features
  - E.g., edge detection by convolutions

Filter  $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

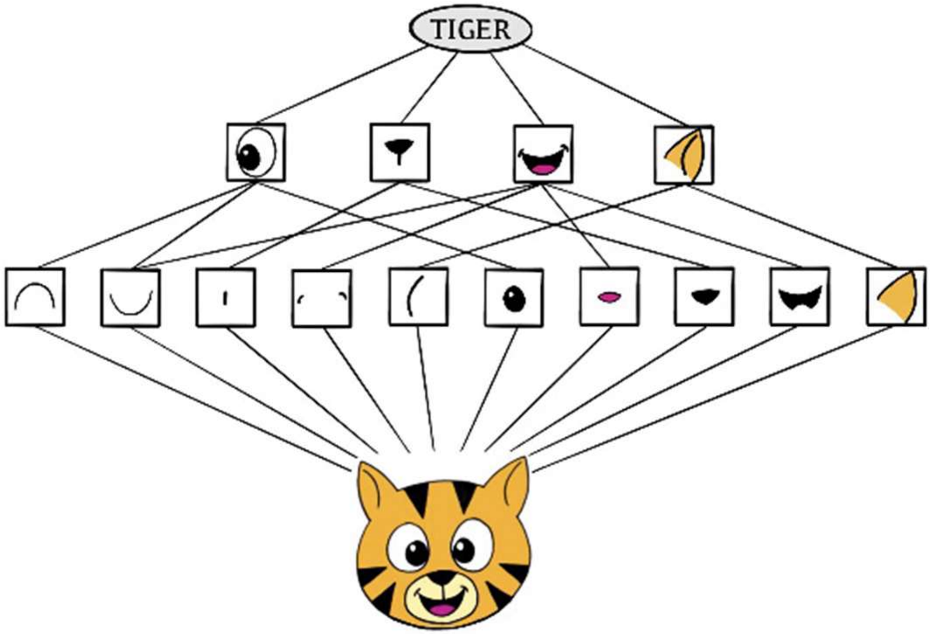
Input Image

Convoluted Image

Slide credit: Param Vir Singh – Deep Learning

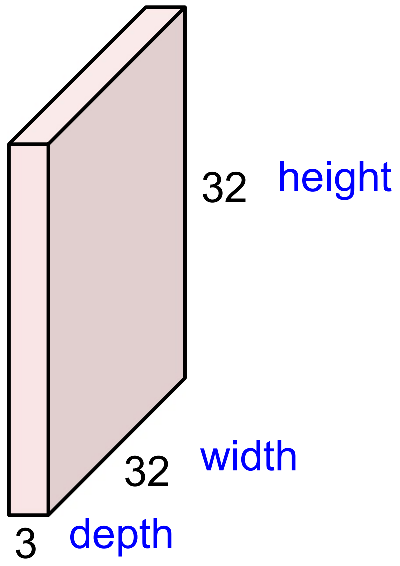
How CNNs Work

- The CNN builds up an image in a hierarchical fashion.
- Edges and shapes (local features) are recognized and pieced together to form more complex shapes (compound features)such as eye and ear, eventually assembling the target image.
- This hierarchical construction is achieved using *convolution* layers.



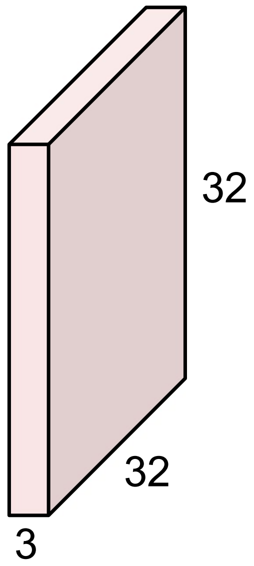
Convolution Layer

32x32x3 image -> preserve spatial structure

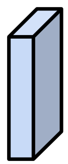


Convolution Layer

- 32x32x3 image



- 5x5x3 filter

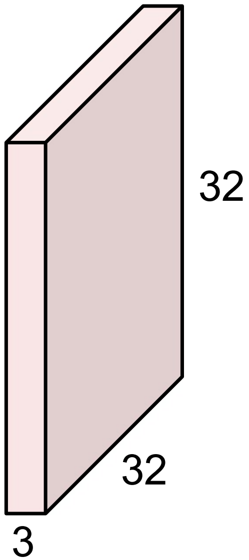


  - **Convolve** the filter with the image
  - i.e. "slide over the image spatially, computing dot products"

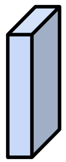
Convolution Layer

Filters always extend the full depth of the input volume

- 32x32x3 image



- 5x5x3 filter

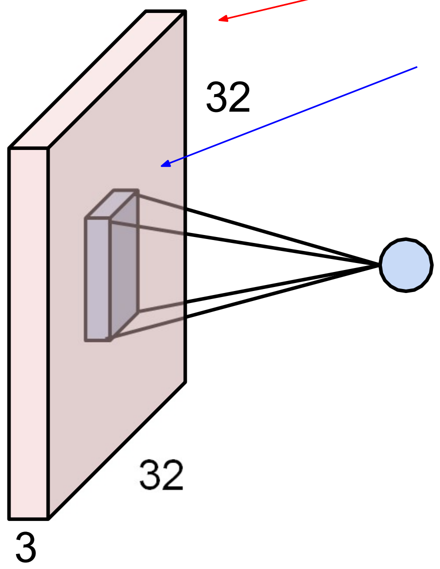


  - **Convolve** the filter with the image
  - i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

5x5x3 filter w



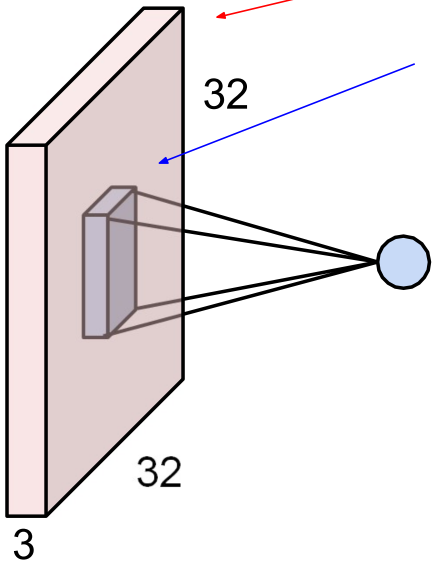
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

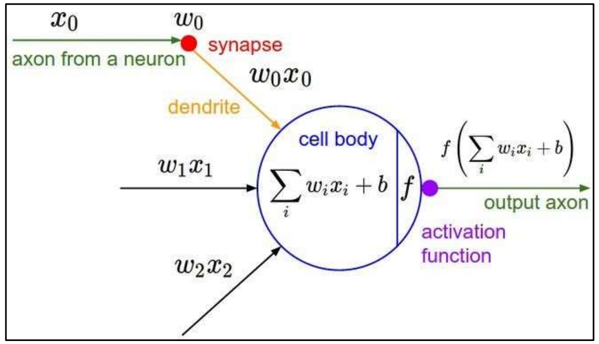
$$w^T x + b$$

Convolution Layer

32x32x3 image

5x5x3 filter w



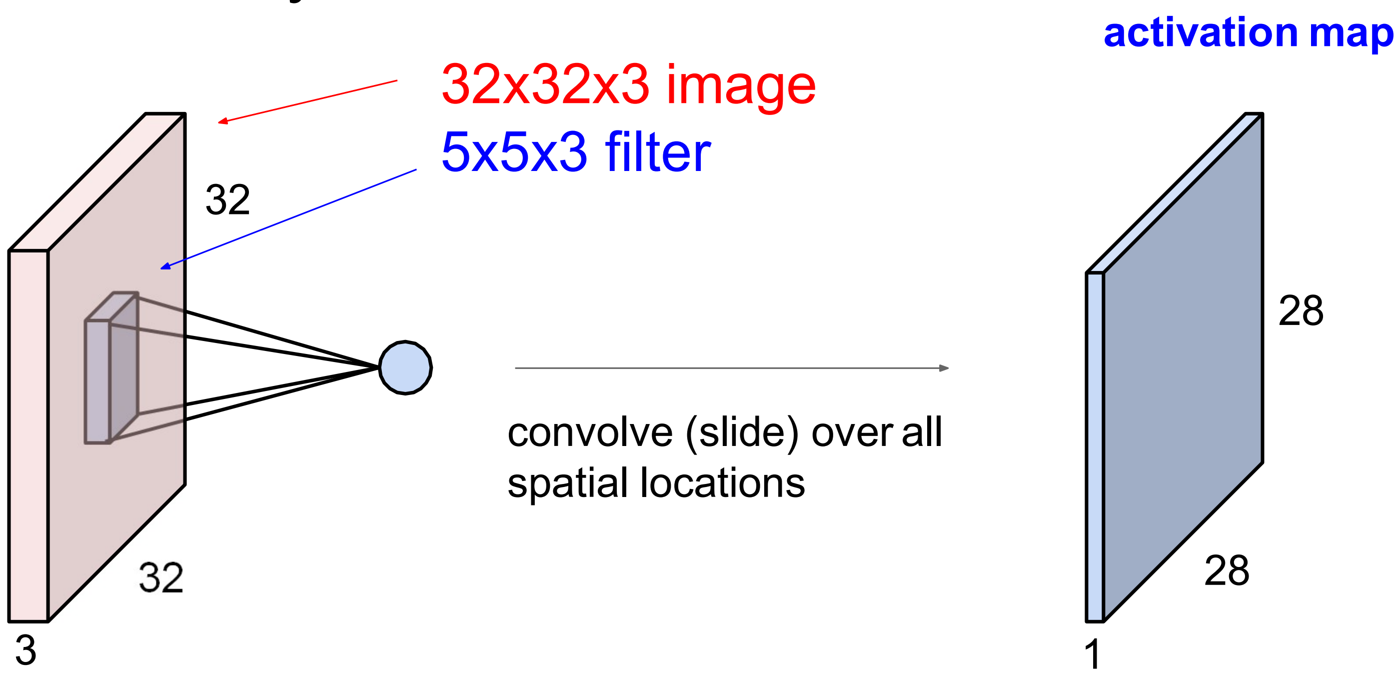


It's just a neuron with local connectivity...

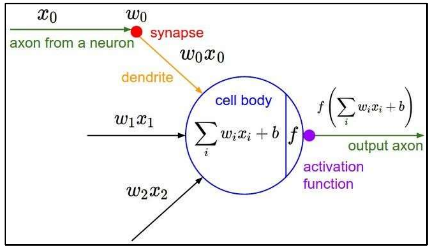
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

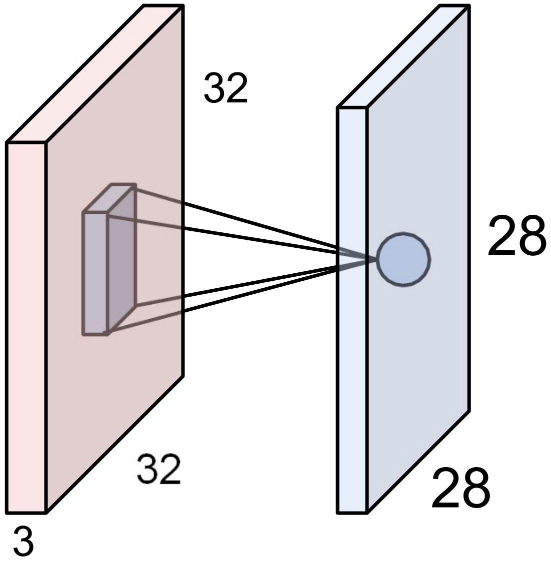
$$w^T x + b$$

Convolution Layer



The brain/neuron view of CONV Layer





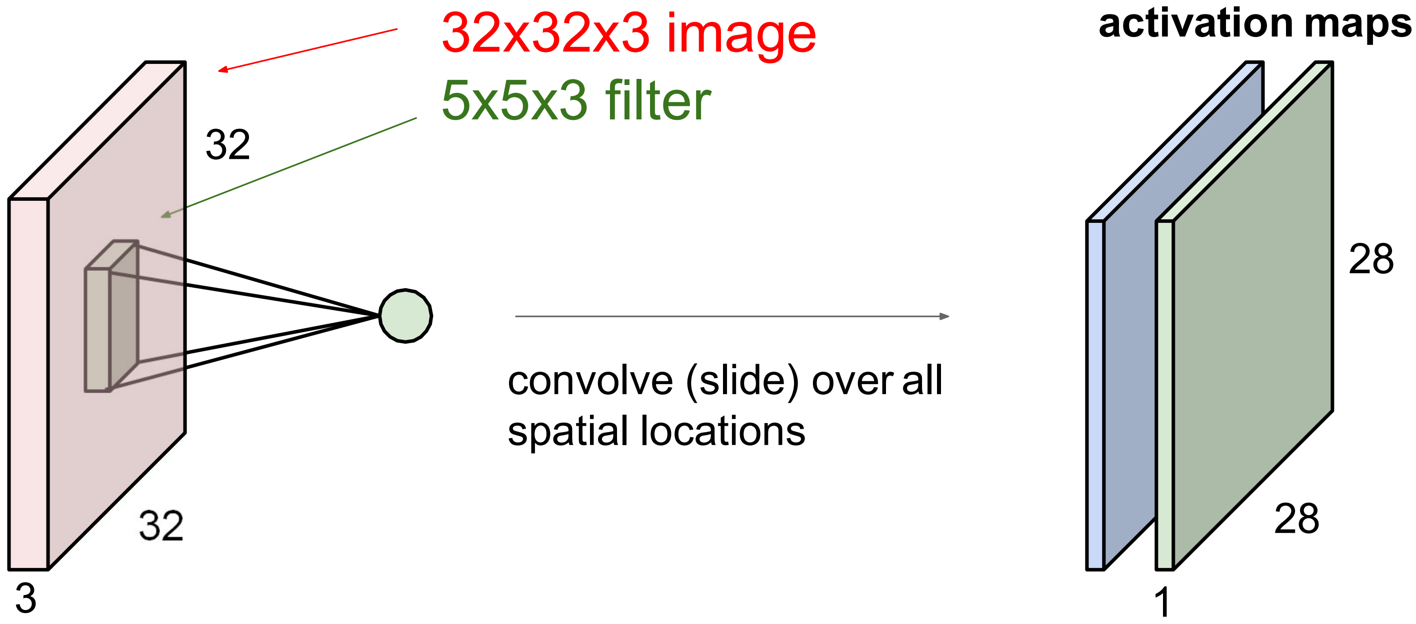
An activation map is a 28x28 sheet of neuron outputs:

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

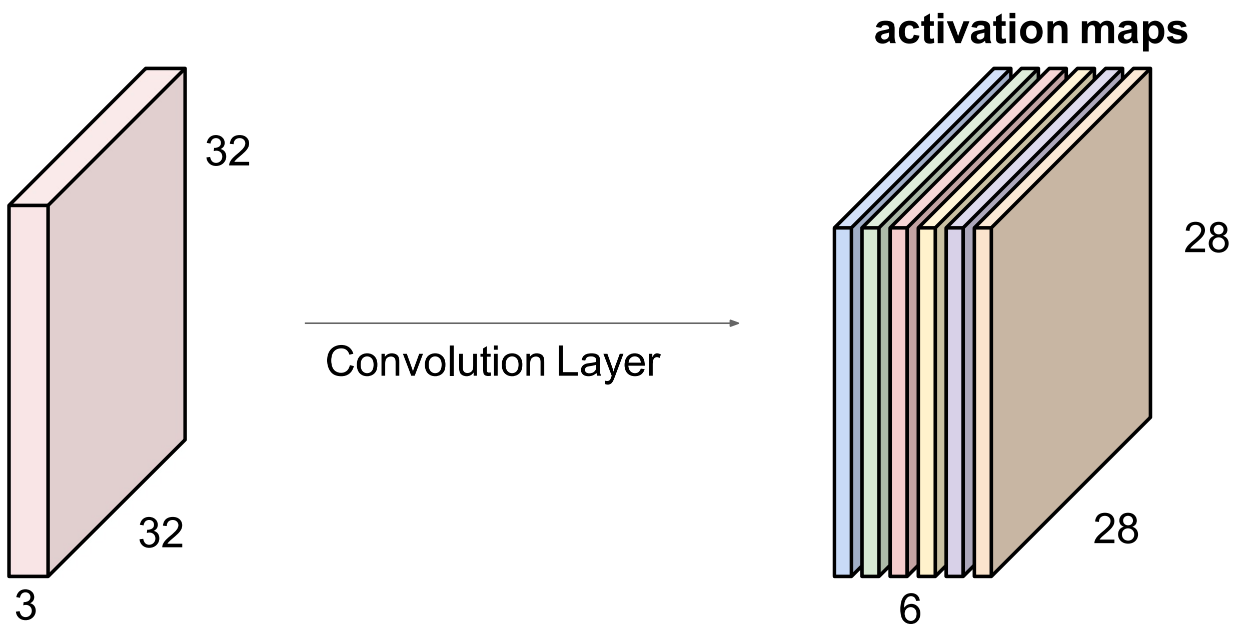
“5x5 filter” -> “5x5 receptive field for each neuron”

Convolution Layer

consider a second, green filter

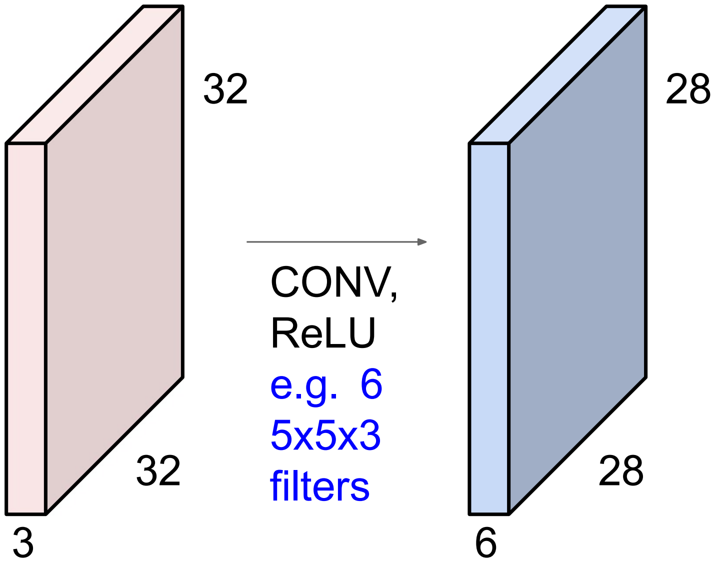


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

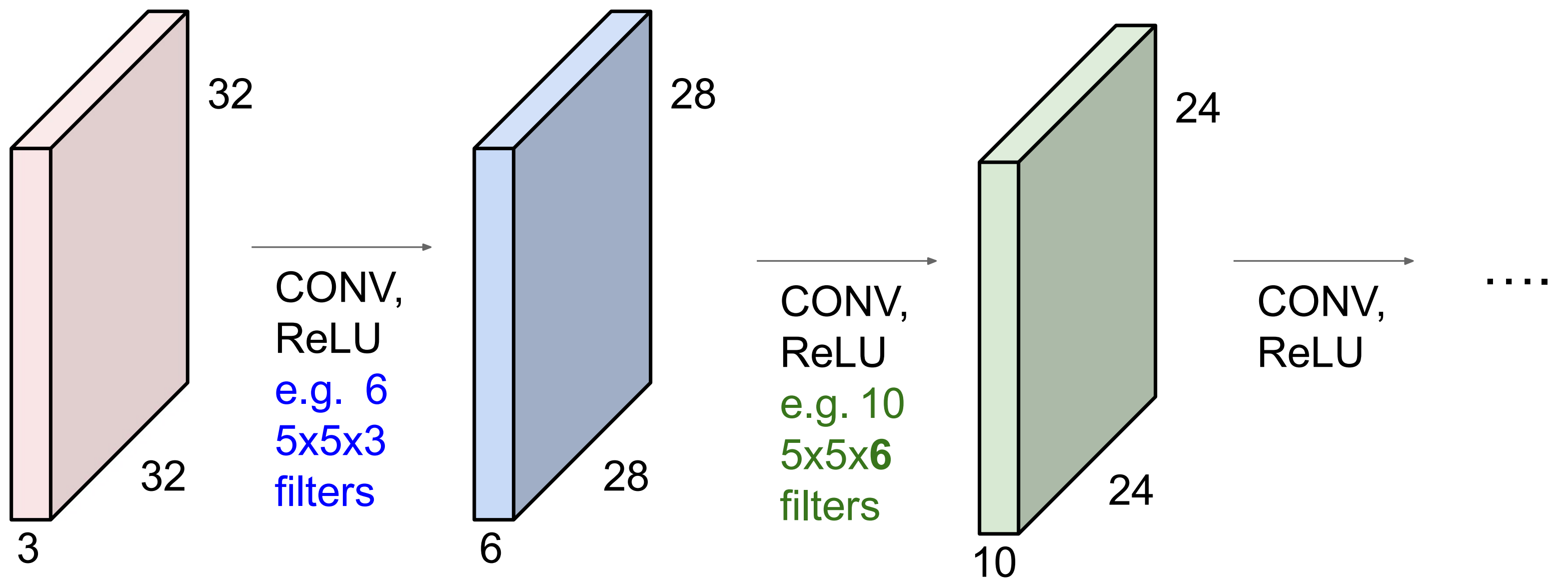


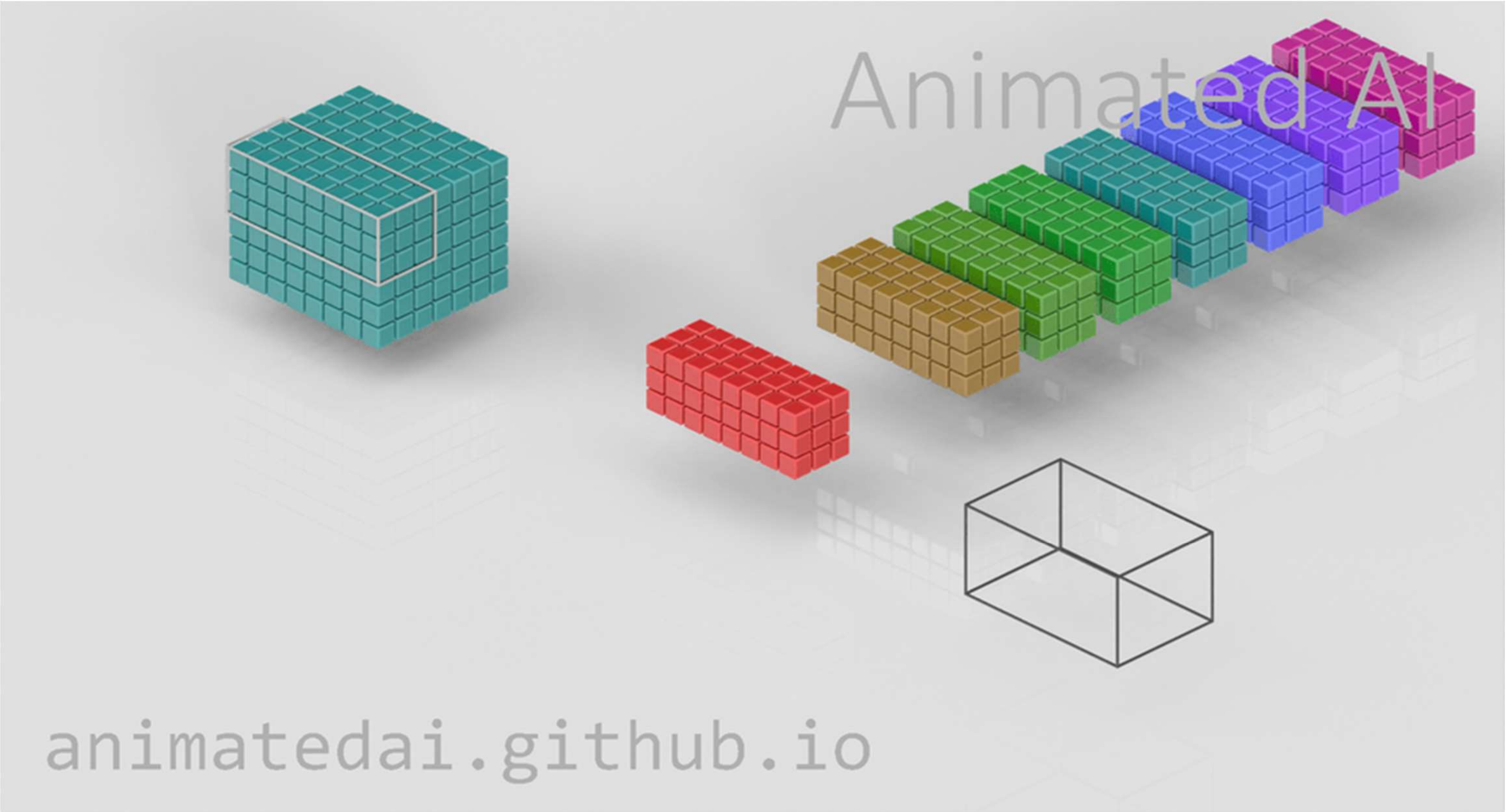
We stack these up to get a "new image" of size 28x28x6!

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

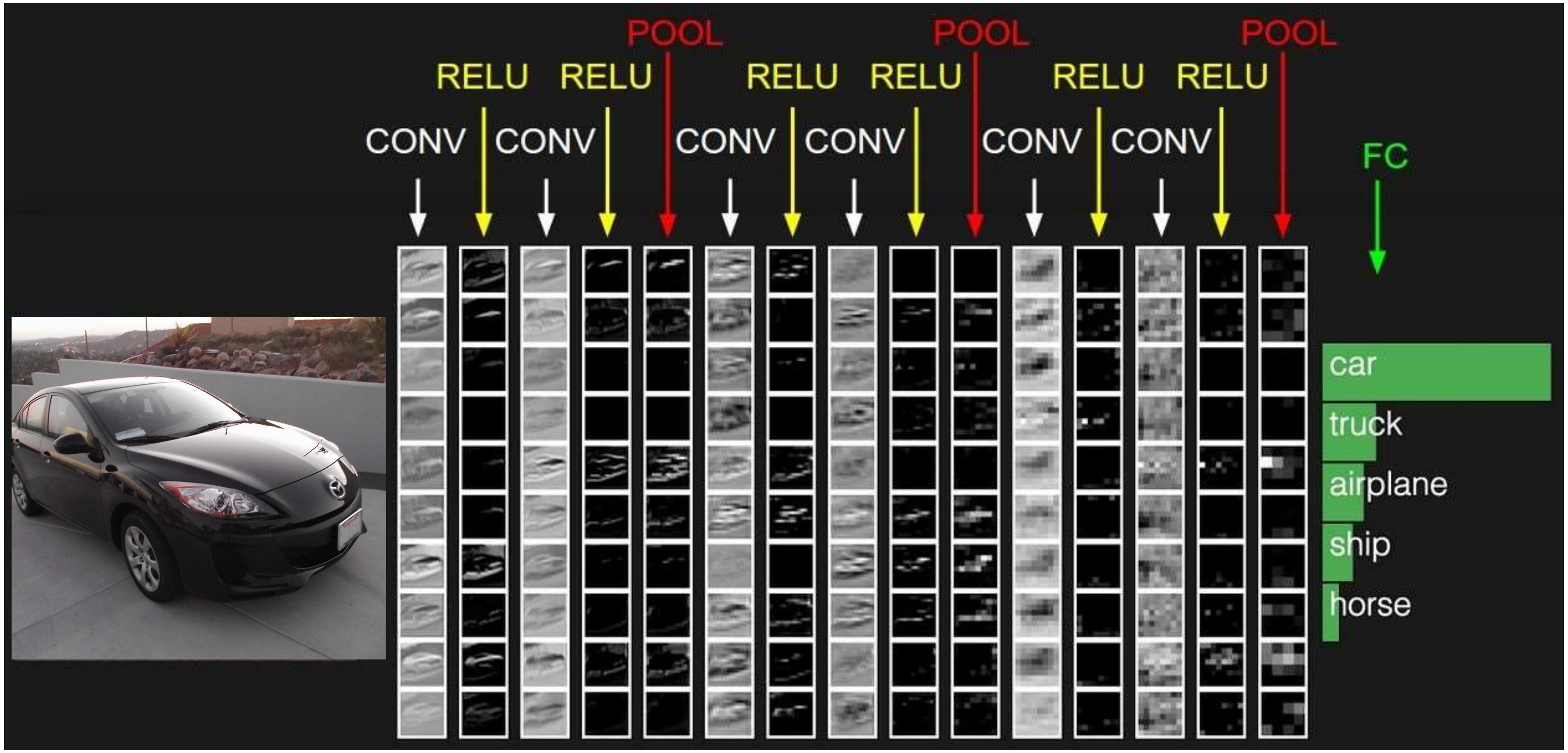


Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions




Animated AI
animatedai.github.io

two more layers to go: POOL/FC



Convolutional Neural Networks (CNNs)

Pooling layer:

- ***Max pooling***: reports the maximum output within a rectangular neighborhood
- ***Average pooling***: reports the average output of a rectangular neighborhood
- Pooling layers reduce the spatial size of the feature maps
  - Reduce the number of parameters, prevent overfitting

MaxPool with a 2×2 filter with stride of 2

1	3	5	3
4	2	3	1
3	1	1	3
0	1	0	4

Input Matrix

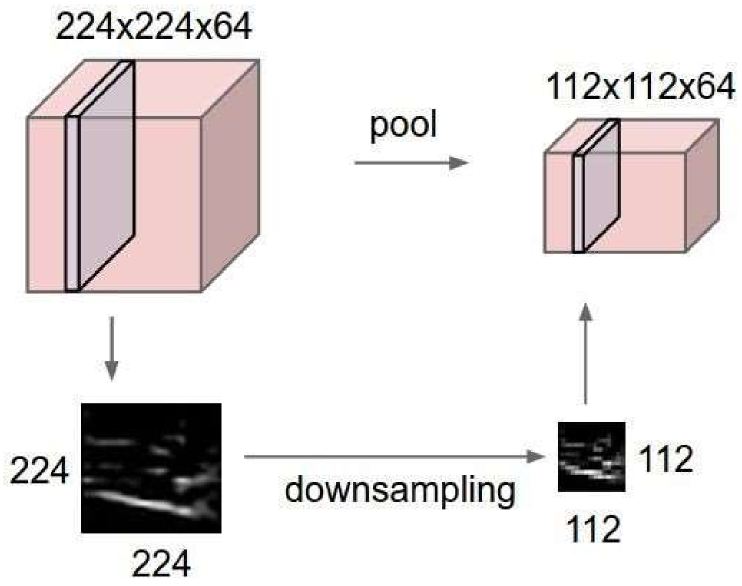
4	5
3	4

Output Matrix

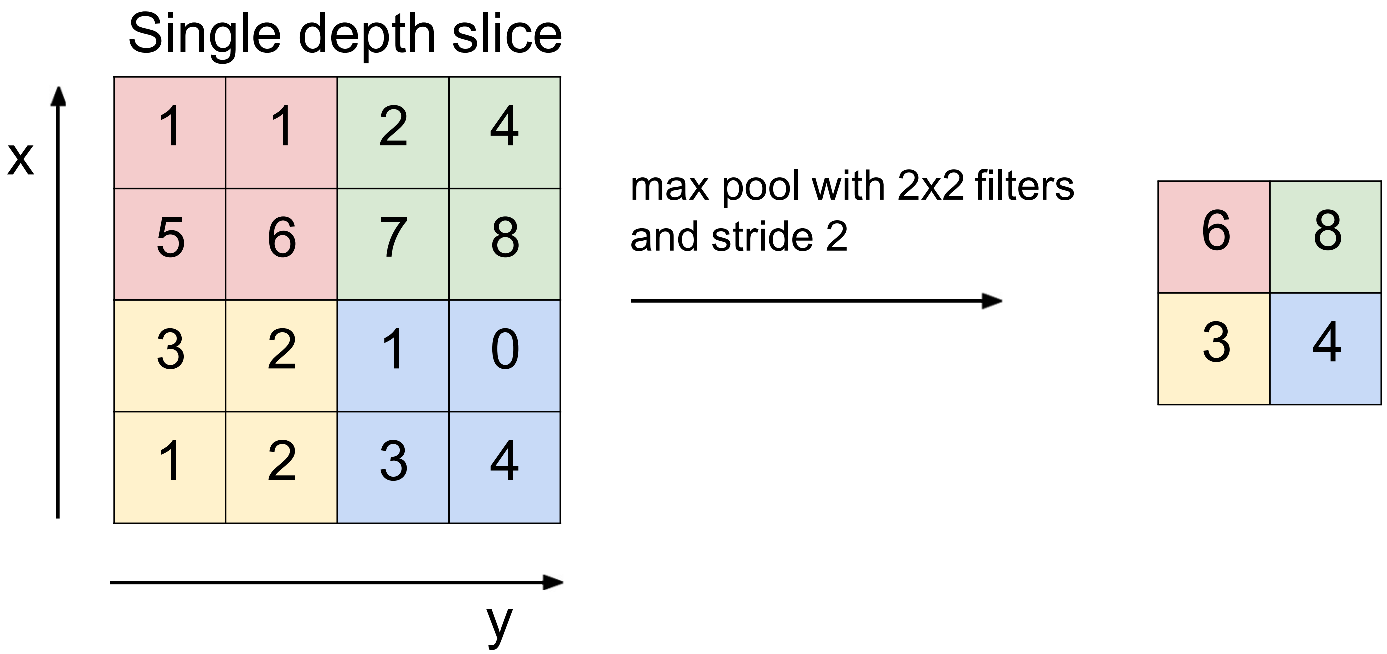
Slide credit: Param Vir Singh – Deep Learning

Pooling layer

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

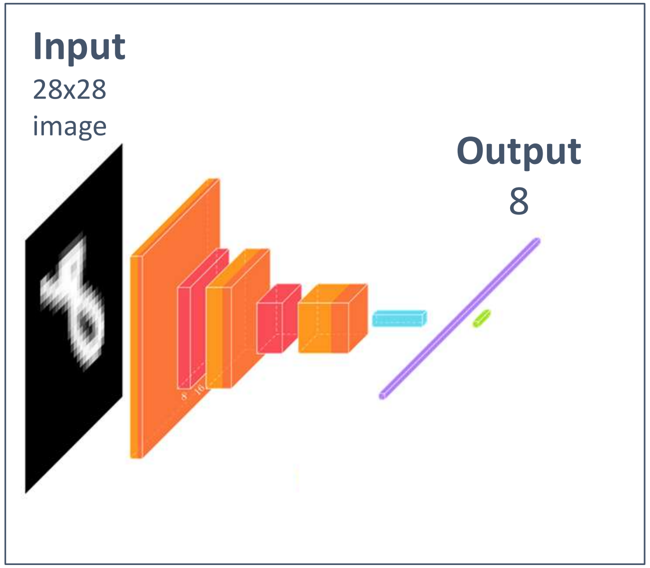


MAX POOLING

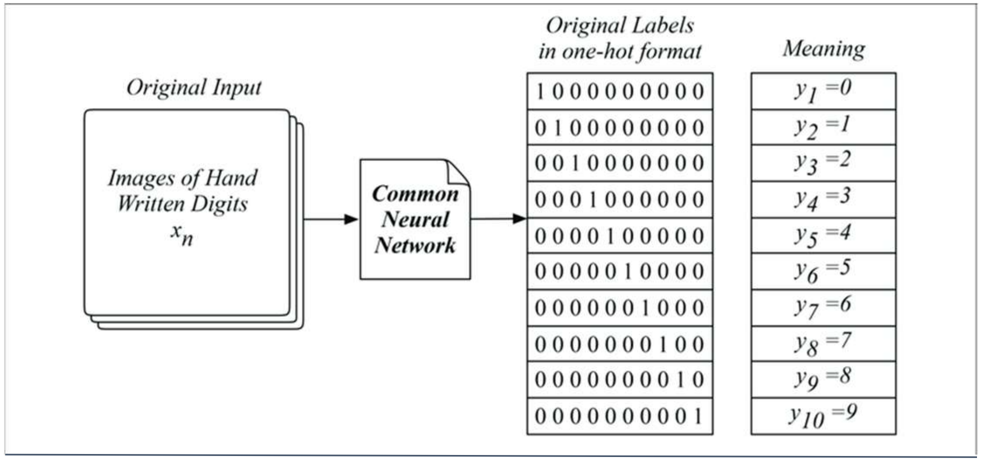


Single depth slice
x
y
1
1
2
4
5
6
7
8
3
2
1
0
1
2
3
4
max pool with 2x2 filters
and stride 2
6
8
3
4

General CNN Architecture for MNIST Digit Classification

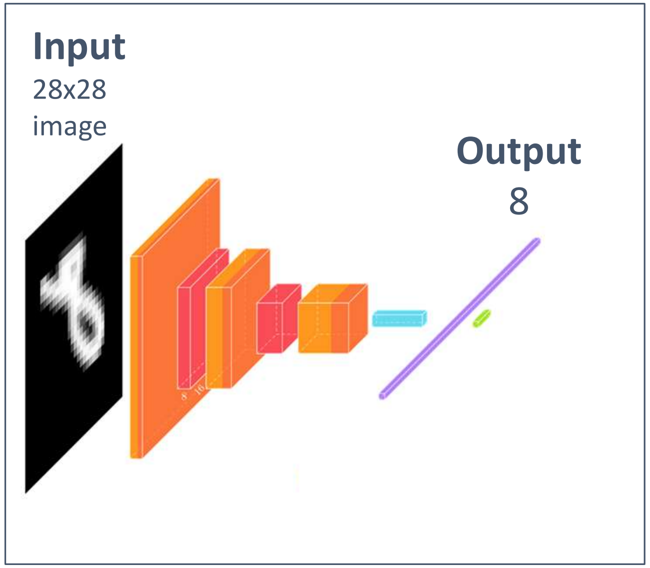


Inference

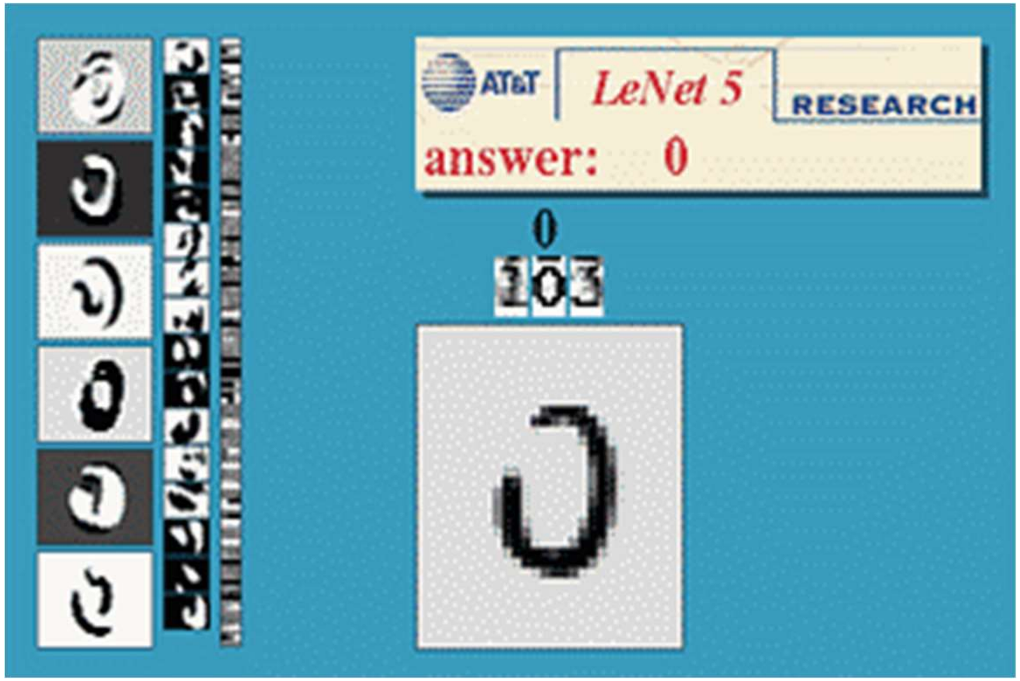


Training

General CNN Architecture for MNIST Digit Classification



Inference



General CNN Architecture for MNIST Digit Classification

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 16)	4,624
conv2d_3 (Conv2D)	(None, 22, 22, 8)	1,160
max_pooling2d (MaxPooling2D)	(None, 11, 11, 8)	0
flatten (Flatten)	(None, 968)	0
dense (Dense)	(None, 10)	9,690

```
model = Sequential()
model.add(Input(shape=(28, 28, 1), name='input_layer'))
model.add(Conv2D(32, (3, 3), activation='relu', name='conv2d_1'))
model.add(Conv2D(16, (3, 3), activation='relu', name='conv2d_2'))
model.add(Conv2D(8, (3, 3), activation='relu', name='conv2d_3'))
model.add(MaxPooling2D(pool_size=(2, 2), name='max_pooling2d'))
model.add(Flatten(name='flatten'))
model.add(Dense(10, activation='softmax', name='dense'))
```