Convolutional neural network
11.11.2024

Outline

- Hour 1
  - Training neural nets continued: stochastic gradient descent
  - Introduction to convolutional neural networks (CNN)
- Hour 2: CNN continued
  - Convolution as a linear map
  - Architecture of CNN

Example application of ML used in Génie mécanique

Prof. Colin Jones: Predictive Control Lab

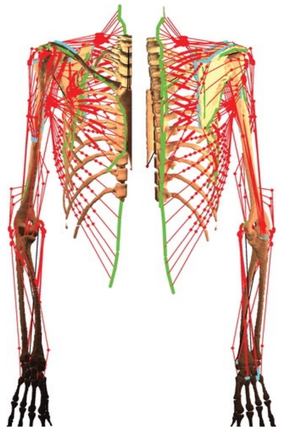
Prof. Pioletti: Laboratory of Biomechanical Orthopedics (LBO)

Prof. Pioletti: Laboratory of Biomechanical Orthopedics (LBO)

Glenohumeral (GH) joint force prediction with a deep learning model (DLM)

- 1000 virtual subjects from a musculoskeletal model (MSM)
- Monte-Carlo simulation for parameter selection
- Parameters (features):
  - Sex, weight, height, activities of daily living, Glenoid version, glenoid inclination, cross sectional area of rotator cuff muscles
- Target: GH joint force

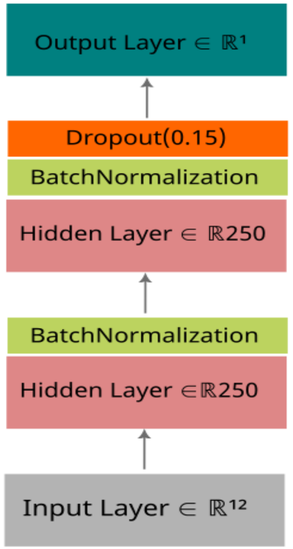
MSM



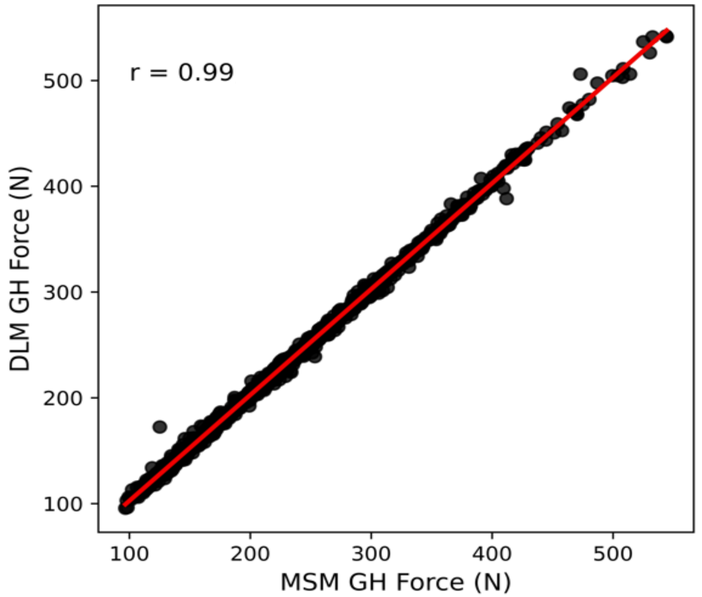
$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = \frac{\partial \Omega}{\partial \dot{q}} M + \lambda_{TS} \frac{\Phi_{TS}}{\partial q} + \lambda_{AI} \frac{\Phi_{AI}}{\partial q}$$

Ehsan Sarshari et al., Sci. Rep., 20806 (2021)

DLM



Performance of DLM:
Mean absolute error of 3.6 (N) on the test data



- DLM prediction more than 1000 time faster than MSM
- DLM easier to integrate with (finite element) patient-specific models

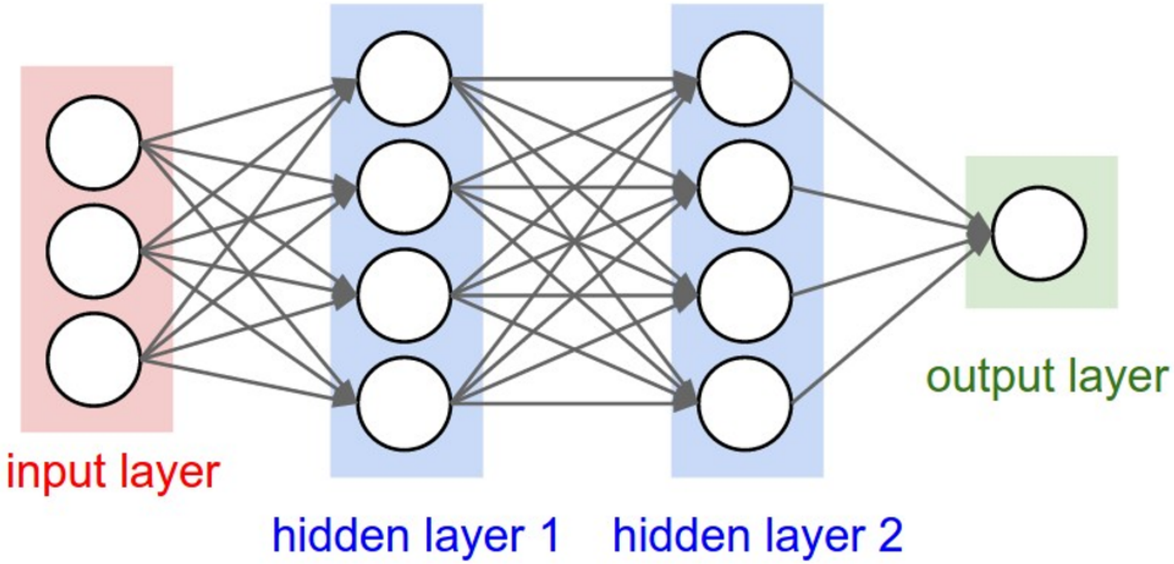
Last time

$\{x^i, y^i\}_{i=1}^N$ training data

we use data to learn a nonlinear prediction

$f_\theta : x \longmapsto f_\theta(x)$

parameters θ : weights & biases of each layer of the neural network



"3-layer Neural Net", or
"2-hidden-layer Neural Net"

Additional resource: https://www.deeplearningbook.org/

Training for NN

Goal of optimization in ML:

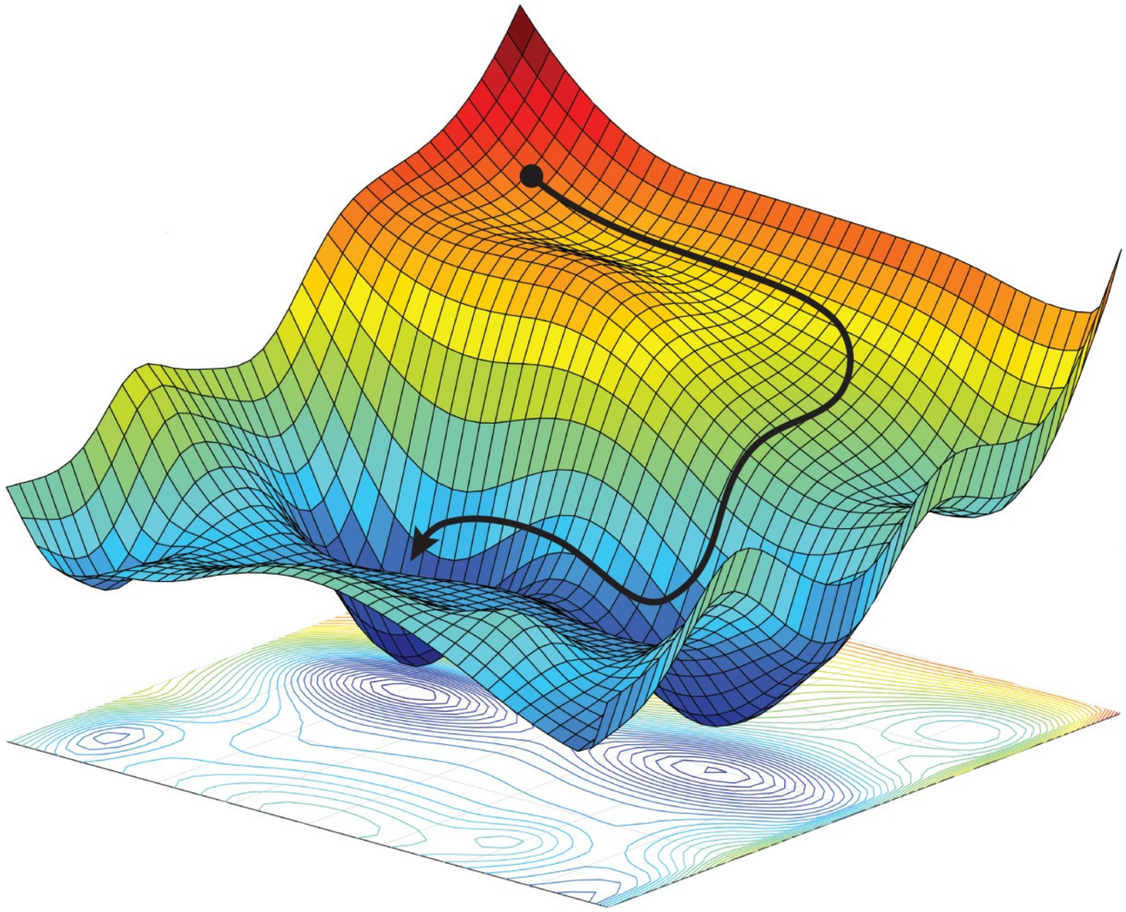
Minimize cost over **training data**: $\sum_{i=1}^{N} L^{(i)}=$

$L(y^{(i)}, \hat{y}^{(i)})$ of the i-th training example

Want the optimization to:

- Converge quickly
- Find a good local minima (or even global minima)

Gradient descent (and variants) is the preferred way to optimize neural networks



A. Amini et al. Spatial Uncertainty Sampling for End-to-End Control, 2019

Training a neural network

Loop:

1. Forward pass to evaluate the the loss function on each data point
2. Backward pass to calculate gradient: backpropogation as shown with a simple example (see Section 5.2 inML4Engineers book.)
3. Update parameters using the gradient

- Forward pass computes result of an operation and save any intermediates needed for gradient computation in memory
- Backward pass applies the chain rule to compute the gradient of the loss function with respect to the inputs: we did an example in class last time

Challenge: gradient computation is slow and memory consuming

Stochastic gradient descent

Training data: $\{x^i, y^i\}_{i=1}^N$,

Loss (cost) $L(\theta)$ θ : set of all W (weights) & b (biases)

$$= \frac{1}{N} \sum_{i=1}^{N} (\hat{y}^i - y^i)^2 \qquad \hat{y}^i = f_\theta(x^i)$$

Gradient of loss

$$\frac{\partial L}{\partial \theta} = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial \theta} (\hat{y}^i - y^i)^2 \qquad \text{(by linearity of differentiation)}$$

Standard gradient descent

$$\theta_{t+1} = \theta_t - \alpha_t \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial \theta} (\hat{y}^i - y^i)^2, \quad \alpha_t : \text{learning rate}$$

Stochastic gradient descent

$$\theta_{t+1} = \theta_t - \alpha_t \frac{\partial}{\partial \theta} (\hat{y}^{n_i} - y^{n_i})^2, \qquad e^{n_i}$$

n_i is a sample from $\{1, 2, \dots, N\}$ uniformly

$$\mathbb{E}\, e^{n_i} = \sum e^{n_i} p(e^{n_i}) = \frac{1}{N} \sum_{i=1}^{N} e^i$$

Mini-batch stochastic gradient descent

Training data: $\{x^i, y^i\}_{i=1}^{N}$

Divide into batches of size $N_b \ll N$

Example : if $N = 500$, $N_b = 20$, we'll have 30 batches;

Mini-batch stochastic gradient descent

$$\theta_{t+1} = \theta_t - \alpha_t \frac{1}{N_b} \sum_{i=1}^{N_b} \frac{\partial}{\partial \theta} \left(\hat{y}^{n_i} - y^{n_i}\right)^2$$ where n_i for $i = 1, 2, \ldots, N_b$

corresponds to the indices of the data points in the batch.

Epoch: # of iterations required to update parameters that let you go through the data set, example : 30 iterations in an epoch

Gradient descent variants

Gradient descent (GD):

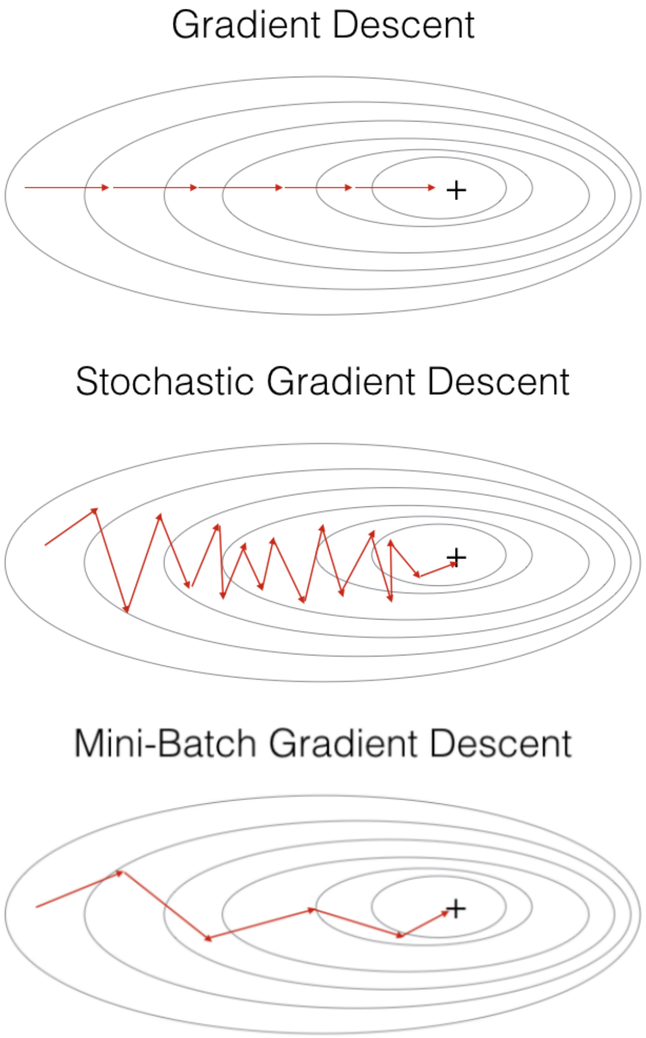
- $J = \frac{1}{N}\sum_{i=1}^{N} L^{(i)}$ weights & biases.
- Weights updated after calculating the gradient over the entire dataset
  - slow
  - requires large memory

Stochastic gradient descent (SGD):

- $J = L^{(i)}$
- Weights updated after calculating the gradient of a single example
  - requires much less memory than GD
  - high variance in parameter updates

Mini-batch Gradient descent

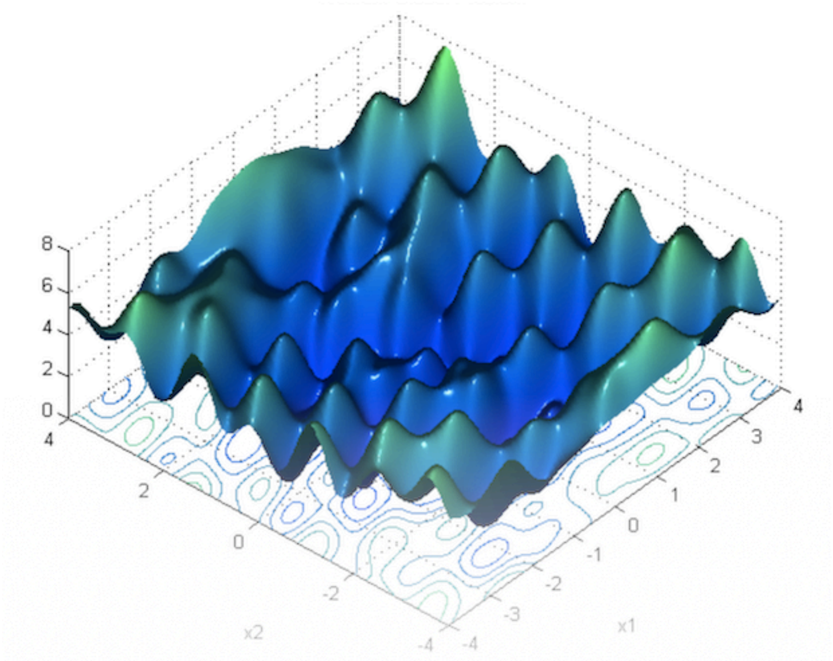
- $J = \frac{1}{N_b}\sum_{i=1}^{N_b} L^{(i)}$
- Weights updated after calculating the gradient over the entire dataset
  - Faster than SGD
  - Reduces variance of gradient estimation



Problems with training

Loss function non-convex

- Choice of initialisation θ_0 (initial weights & biases)
- Choice of learning rate α_t $t = 0, \dots, T$
- Affect the local minimum found



Variants of gradient descent are commonly used in practice to speed-up and improve convergence:

- Momentum update
- Nesterov Accelerated Gradient (NAG)
- Adam
- and more…

Deep learning frameworks
Overview

Deep learning frameworks are used to efficiently define and train neural networks

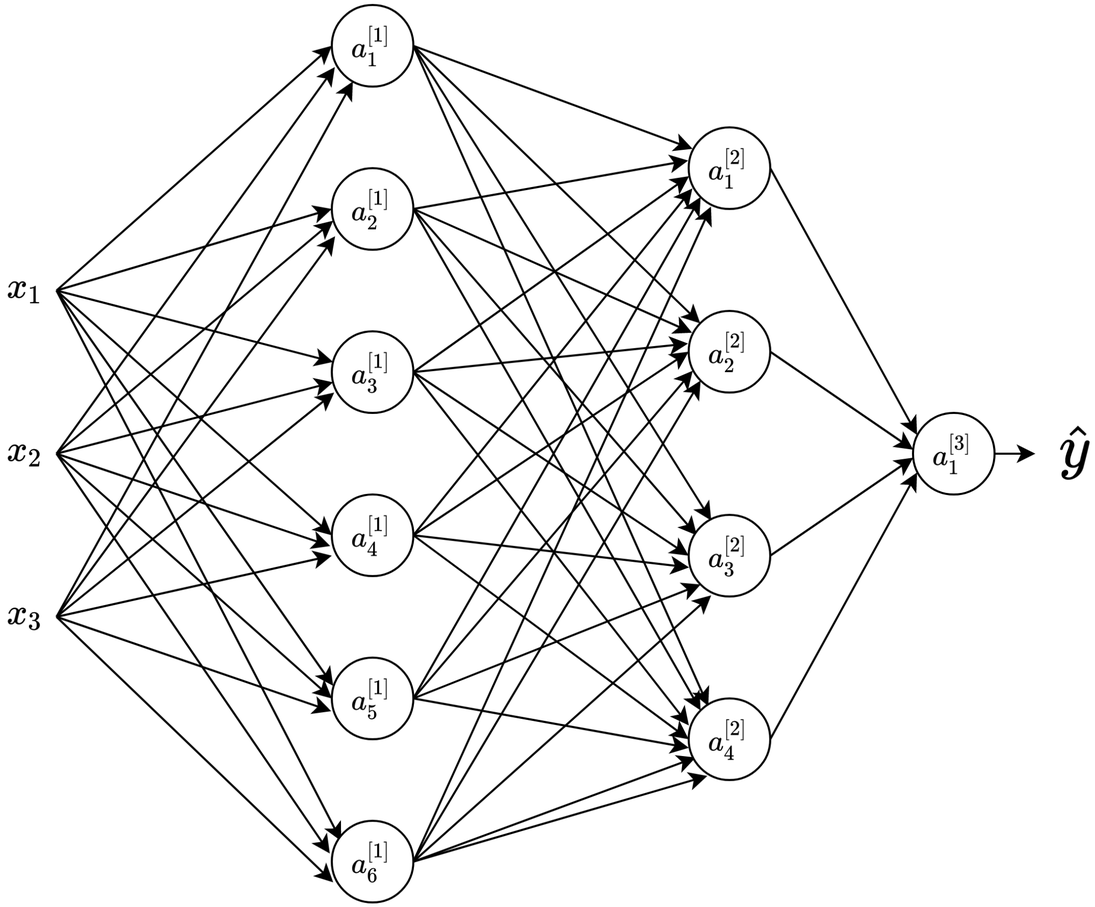
- Support for many types of layers, activations, loss functions, optimizers, …
- Backpropagation computed automatically (e.g. loss.backward() in PyTorch)
- GPU support for faster training

Most popular frameworks today:

- PyTorch (https://pytorch.org)
- TensorFlow (https://www.tensorflow.org/)

Deep learning frameworks

Implementing a simple neural network in PyTorch




```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(3, 6)
        self.fc2 = nn.Linear(6, 4)
        self.fc3 = nn.Linear(4, 1)

    def forward(self, x):
        # First layer
        x = self.fc1(x)
        x = F.relu(x)
        # Second layer
        x = self.fc2(x)
        x = F.relu(x)
        # Output layer
        x = self.fc3(x)
        return x
```

Last week, Python exercise 1 - digit classification

- You created a neural network for hand-written digit classification
- Training data is based on MNIST dataset: 70,000 images containing hand-written digits
- How does the classifier “see” an image? What are the features given to the classifier?
  - What does it mean to flatten an image?
  - How many parameters needed to be determined in training?

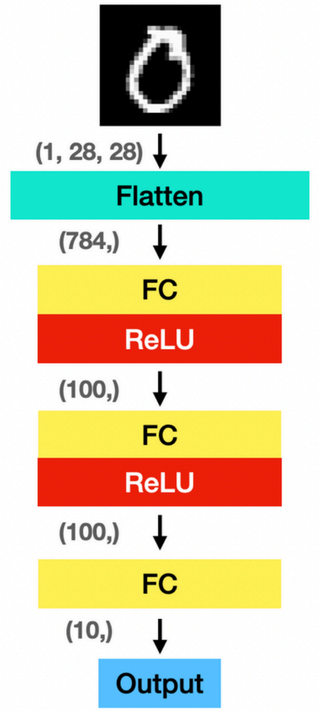
consider first layer:

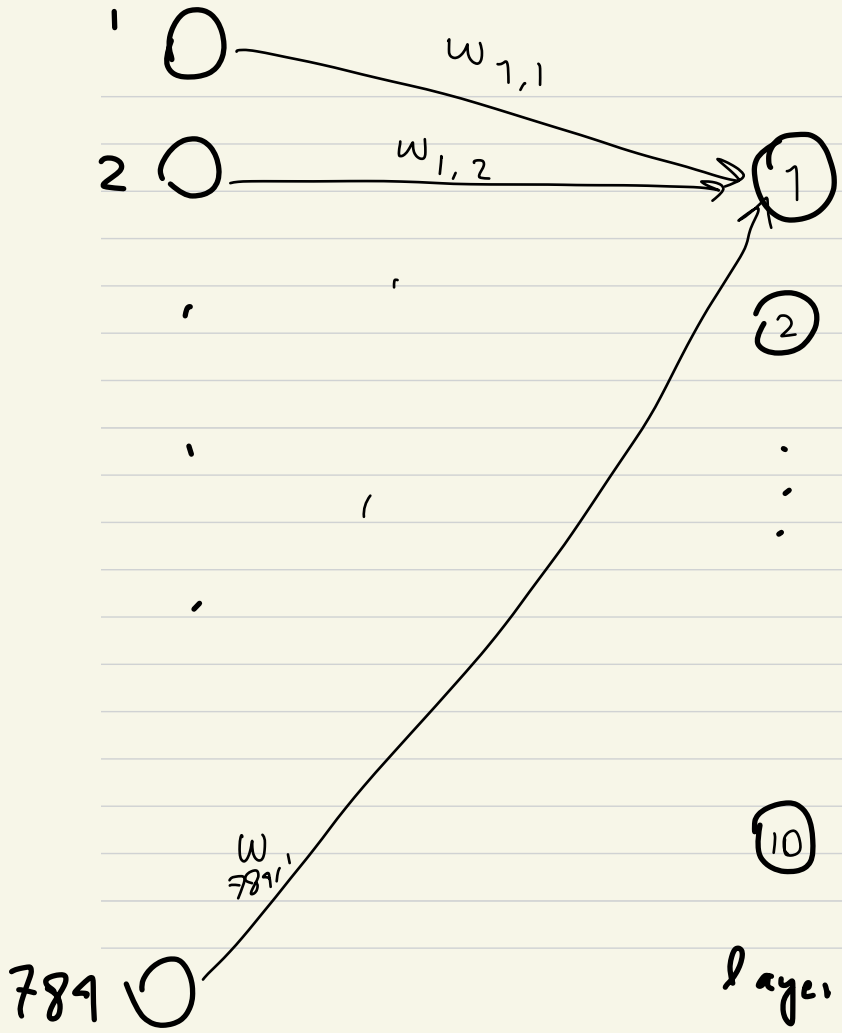
input size 784

$x \in \mathbb{R}^{784}$

suppose we have 10 neurons in first layer.

How many parameters for first layer we have?





$x \in [0,1]^{784}$, $(784 = 28 \times 28)$

$w_i \in \mathbb{R}^{784}$, $i = 1, \dots, 10$,

$b_i \in \mathbb{R}^{10}$

incoming to neuron 1

$w_1^T x + b_1 \in \mathbb{R}$

out from neuron 1

$g(w_1^T x + b_1) \in \mathbb{R}$

{

activation function

Python exercise 2 - fault detection in electric lines

Note: the only difference between two exercises are the datasets

- **Datasets:** Electrical Faults & Dielectric Breakdown
- **Goals:**
  - Detect and classify faults in transmission lines
  - Predict the dielectric breakdown of insulators

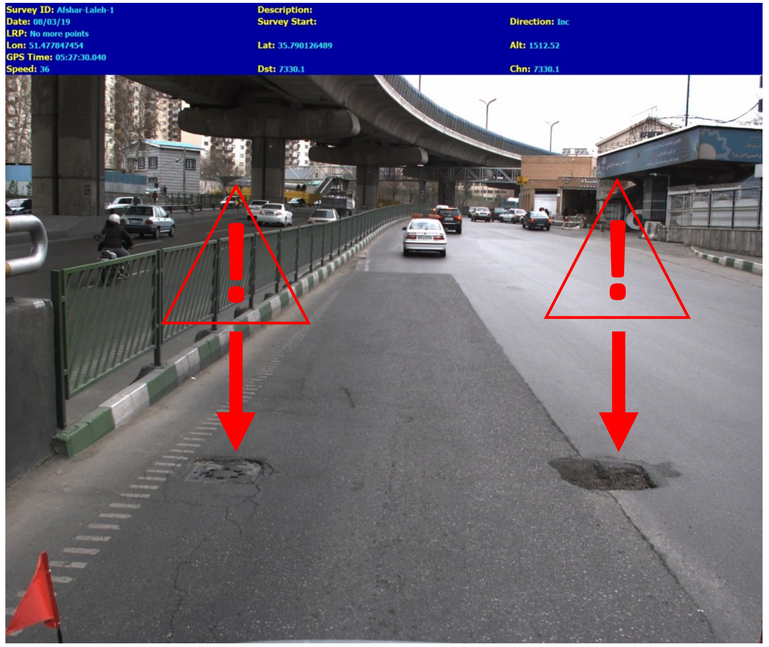
Reflection

- Neural networks are powerful but
  - Computation is intense
    - A lot of time and resources (GPU, etc). Can everyone have equal access to these resources?
    - You might need to use Google Colab or other platforms managed by companies. Are there any data privacy issues?
  - Training neural networks is ENERGY consuming
  - The predictor might not have interpretability

Convolutional Neural Networks

Motivation: real-World Problem

Detecting and Classifying Pavement Distress



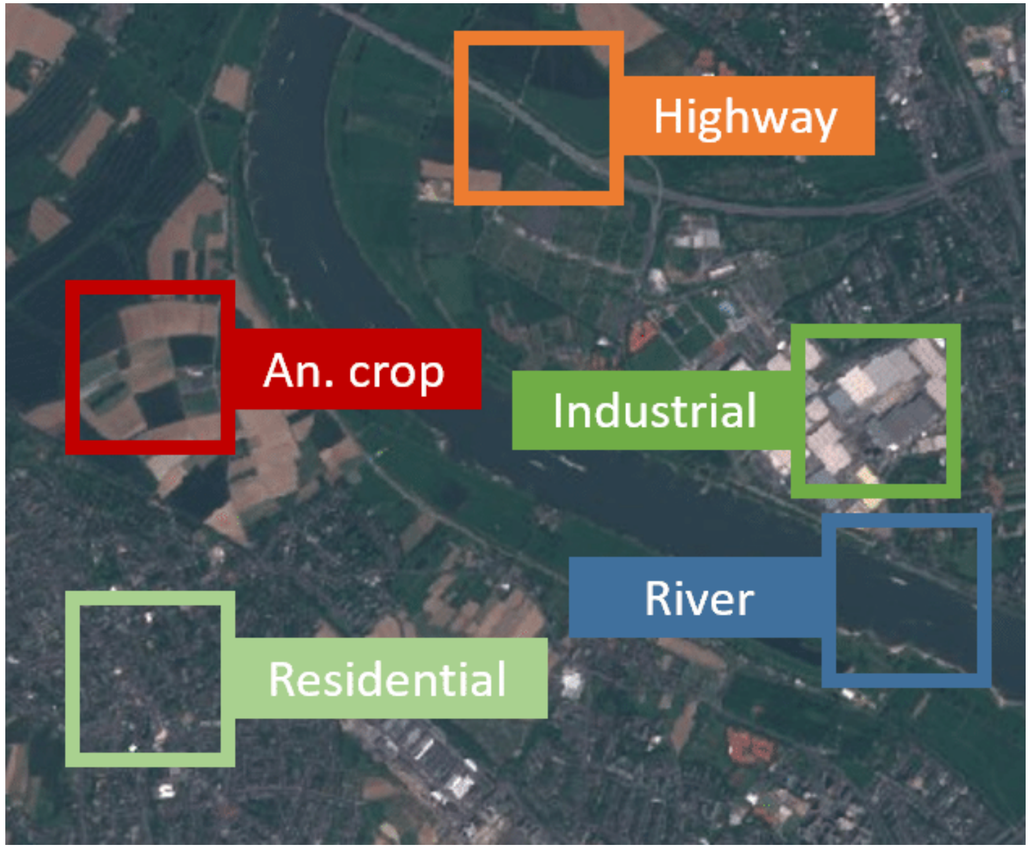
Why? On-time preventive maintenance

Lack of on-time maintenance

- x3 the maintenance cost
- traffic delay
- more fuel consumption
- accidents
- …

Motivations (you will do this in your exercise this week)

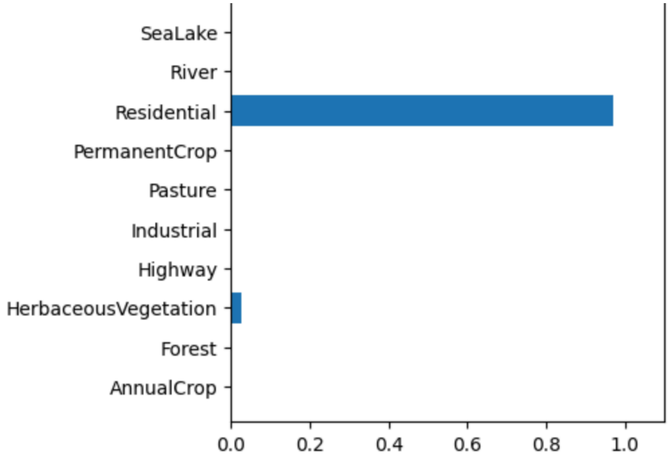
- **Datasets:** EuroSAT (images taken by Sentinel-2 satellite)
- **Goal:** Classify land cover of pictures taken by Sentinel-2

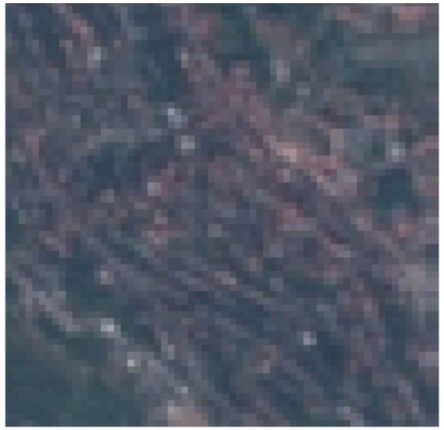


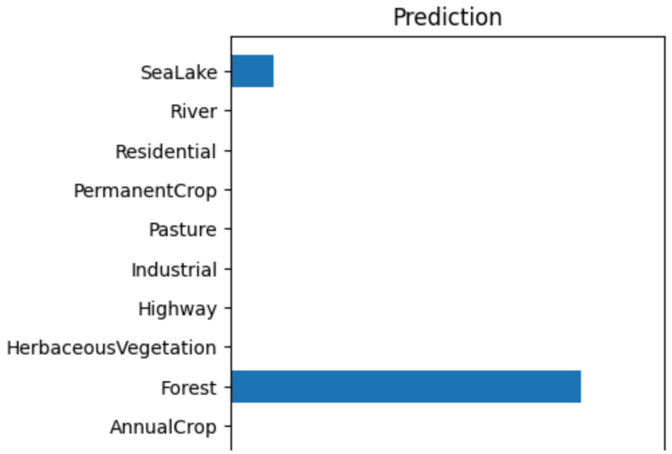

```python
class EuroSAT(ImageFolder):

    classes = [
        "0 - AnnualCrop",
        "1 - Forest",
        "2 - HerbaceousVegetation",
        "3 - Highway",
        "4 - Industrial",
        "5 - Pasture",
        "6 - PermanentCrop",
        "7 - Residential",
        "8 - River",
        "9 - SeaLake",
    ]
```

Convolutional Neural Network dataset 2

- **Comparison:** NN versus CNN in terms of number of parameters and accuracy
- What is the difference between the dimension of the input given to the NN versus CNN?







Model size

```
[ ]: torchsummary.summary(three_layer_net, (3, 64, 64), device="cpu")
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                [-1, 1000]      12,289,000
            Linear-2                 [-1, 100]         100,100
            Linear-3                  [-1, 10]           1,010
================================================================
Total params: 12,390,110
Trainable params: 12,390,110
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.01
Params size (MB): 47.26
Estimated Total Size (MB): 47.32
----------------------------------------------------------------
```

```
[ ]: torchsummary.summary(lenet, (3, 64, 64), device="cpu")
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 18, 64, 64]           1,368
            Conv2d-2          [-1, 54, 28, 28]          24,354
            Linear-3                 [-1, 120]       1,270,200
            Linear-4                  [-1, 84]          10,164
            Linear-5                  [-1, 10]             850
================================================================
Total params: 1,306,936
Trainable params: 1,306,936
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.89
```

Convolutional Neural Networks (CNN)

Intro - Handling images with fully-connected NN

3x32x32 image

height (32)

width (32)

depth (3)

→ for the 3 color channels: R, G, B

Flatten

3072x1 input

By flattening, spatial structure gets lost!

Convolutional Neural Networks

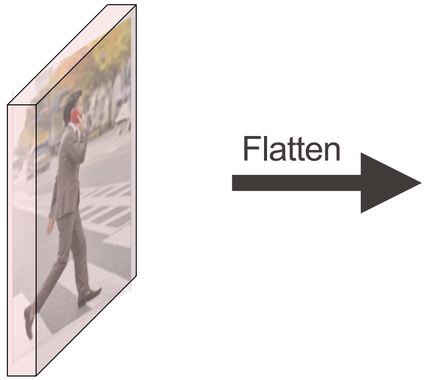
Intro - Handling images with fully-connected NN

A fully-connected neural net:

- Requires flattening the image

  → spatial structure gets lost

- Doesn’t scale well to large images
  - e.g. 1024x1024x3 image results in 3’145’728 weights for each neuron of first hidden layer



How to efficiently model correlation between neighboring pixels?

=> **Convolutional Neural Networks**

Convolution definition

Convolution of a signal with a filter (note: a filter is also referred to as kernel in ML)

Signal $x \in \mathbb{R}^d$

Filter (kernel) $w \in \mathbb{R}^{2m+1}$

Convolution outcome $z \in \mathbb{R}^d$, $z_i = \sum_{j=-m}^{m} w_j x_{i+j}$, $i = m \dots, d-m$

example:

$x = (0, 1, 1, 2, 3, 5, 8, 13) \in \mathbb{R}^8$

$w = (1, -2, 1) \in \mathbb{R}^3$ $\quad m = 1$

$z_4 = 1 \times 1 + -2 \times 2 + 1 \times 3 = 0$

$z_3 =$

$z_5 =$

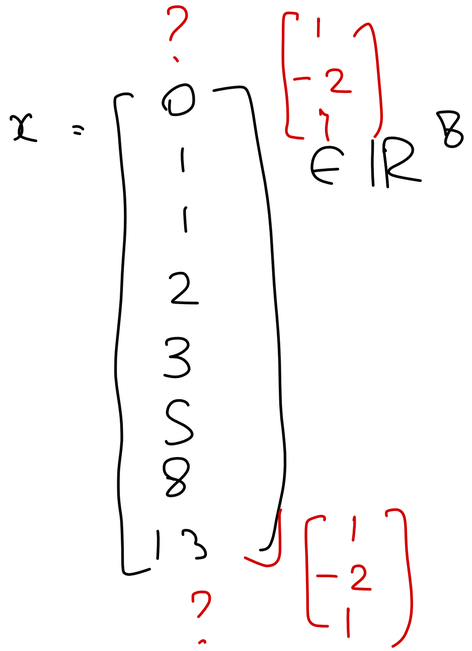
How do we compute z_1 & z_8 ?

Convolution - what to do at boundaries

1) zero-padding: $\bar{x} = (0, 0, 1, 1, 2, 3, 5, 8, 13, 0)$, $w = (1, -2, 1)$

do now $z_1 = 1 \times 0 + -2 \times 0 + 1 \times 1$

$z_8 = \ldots$



2) replikon $\bar{x} = (0, 0, 1, 1, 2, 3, 5, 8, 13, 13)$

$z_1 = 1 \times 0 + -2 \times 0 + 1 \times 1$

$z_8 = \ldots$

3) ignore boundaries & return $z \in \mathbb{R}^{d-2m}$

ex: $z \in \mathbb{R}^{d-2} = \mathbb{R}^6$

Convolution to get features

Filters can approximate what happens in neighborhood with a few numbers:

give average value of signal in a neighborhood $w = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$, $z_i = \frac{x_{i-1} + x_i + x_{i+1}}{3}$

sharpen the signal $w = (-1, 4, -1)$, $z_i = -x_{i-1} + 4x_i + -x_{i+1}$

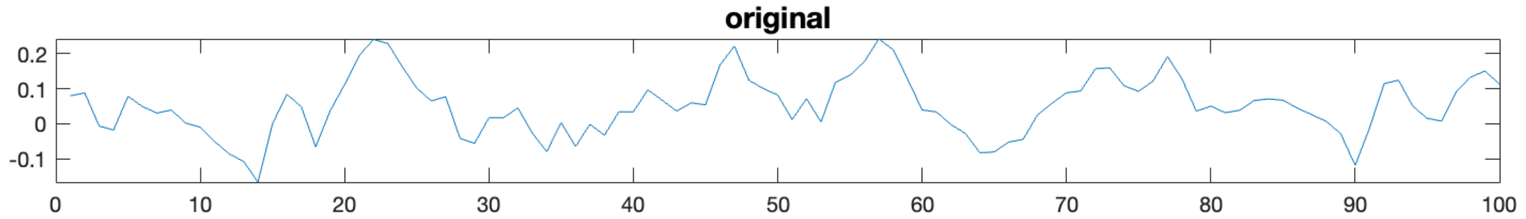
blur the signal $w = \left(e^{-\frac{1}{2\sigma^2}}, e^{0}, e^{\frac{1}{2\sigma^2}}\right)$, σ is a chosen

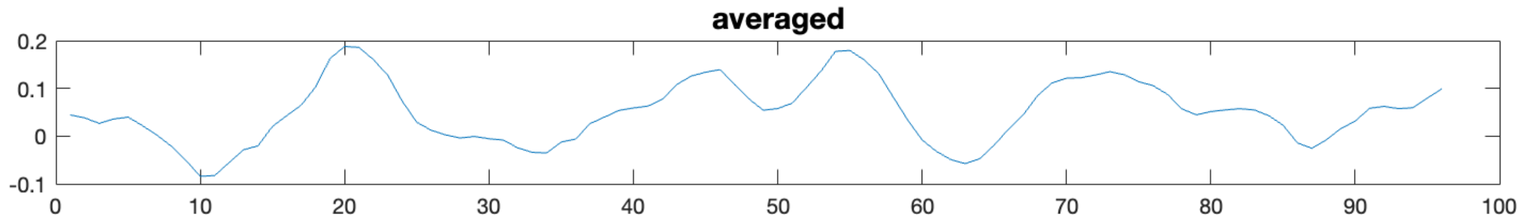
approximate derivative of a signal in a neighborhood: $w = (-1, 0, 1)/2$ $z_i = \frac{x_{i+1} - x_{i-1}}{2}$

approximate second derivative of a signal in a neighborhood $w = (1, -2, 1)$, $z_i = x_{i-1} - 2x_i + x_{i+1}$

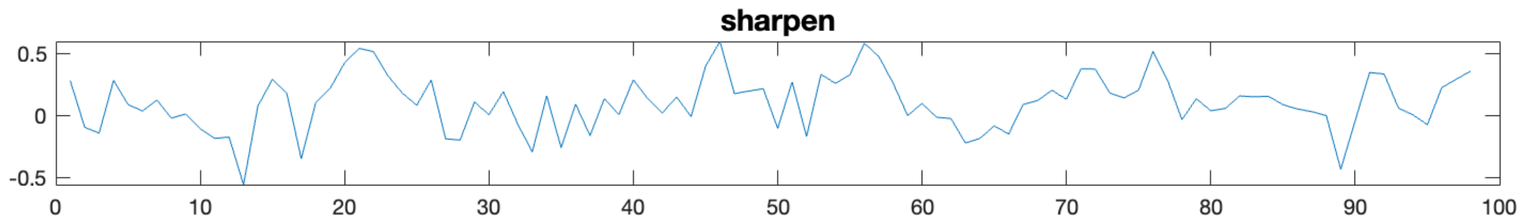
EPFL

$w = \left(\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right)$

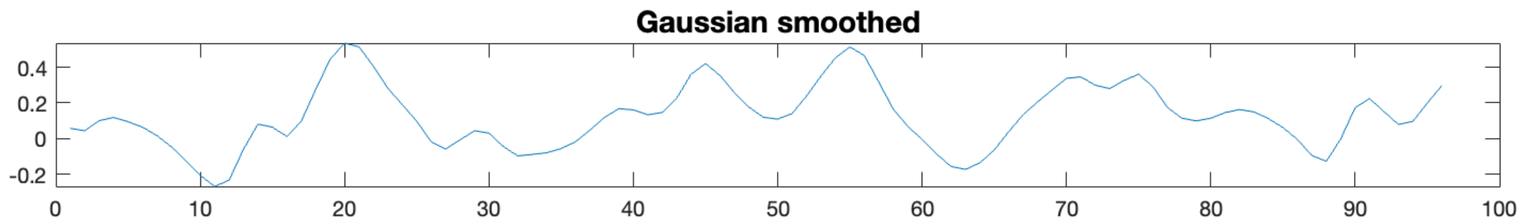




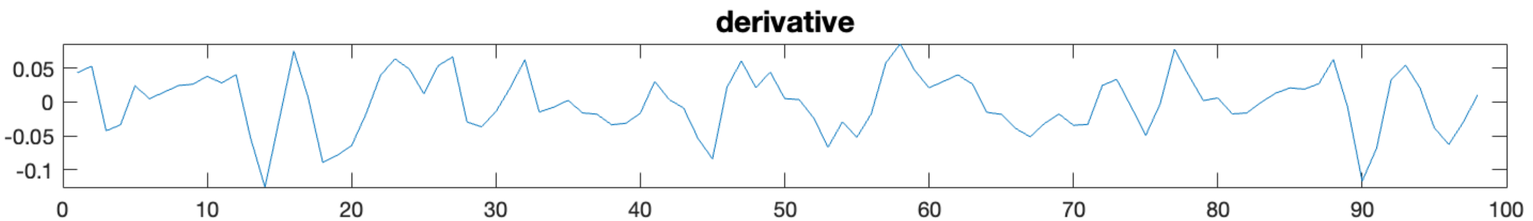
$w = (-1, 4, -1)$



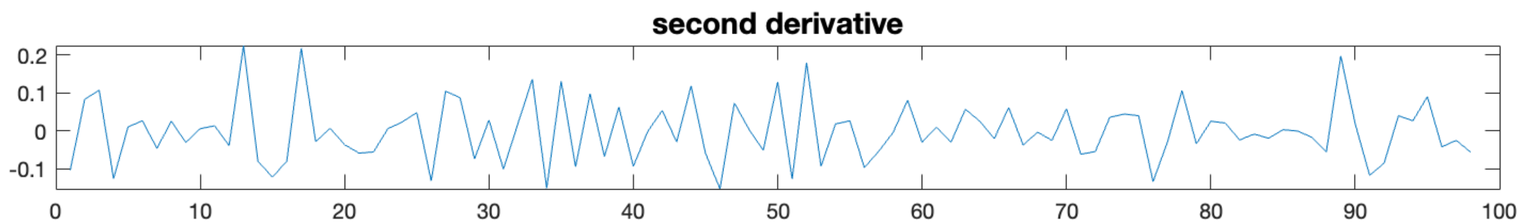
$w = \left(e^{-\frac{2}{2\sigma^2}}, e^{-\frac{1}{2\sigma^2}}, 0, e^{\frac{1}{2\sigma^2}}, e^{\frac{2}{\sigma^2}}\right)$



$w = (-1, 0, 1)$



$w = (1, -2, 1)$



Convolution extension

Continuous signals (functions)

$x : \mathbb{R} \to \mathbb{R}$, $w : \mathbb{R} \to \mathbb{R}$,

$x(t) \in \mathbb{R}$, $w(t) \in \mathbb{R}$

$$z(t) = x * w = \int_{-\infty}^{\infty} x(\tau)\, w(t-\tau)\, d\tau \qquad z(t) \in \mathbb{R}$$

$z : \mathbb{R} \to \mathbb{R}$

Matrices:

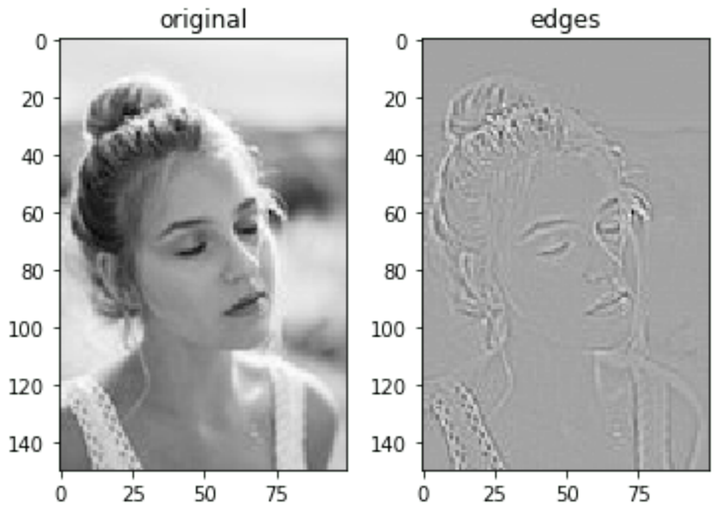
$M \in \mathbb{R}^{d_1 \times d_2}$, $W \in \mathbb{R}^{m_1 \times m_2}$

$M * W$

Example of image convolution

filter / kernel

-1	-1	-1
-1	8	-1
-1	-1	-1



https://muthu.co/basics-of-image-convolution/

Convolutional

2D Convolution computation example

To show how the convolution operation is computed, let's use a simpler example:

5x5 input, **3x3** filter

Input (5x5)

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

Filter (3x3)

1	-1	0
0	2	-3
-1	0	2

Bias: $b = 0$

Convolutional

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

=

2		

Bias: $b = 0$

1x1 + 0x(-1) + 3x0 + 0x0 + 3x2 +
4x(-2) + 1x(-1) + 0x0 + 2x2 + 0
= 2

Convolutional

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

=

2	5	

Bias: $b = 0$

0x1 + 3x(-1) + 0x0 + 3x0 + 4x2 +
0x(-2) + 0x(-1) + 2x0 + 0x2 + 0
= 5

Convolutional
Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

=

2	5	-1

Bias: $b = 0$

3x1 + 0x(-1) + 2x0 + 4x0 + 0x2 +
2x(-2) + 2x(-1) + 0x0 + 1x2 + 0
= -1

Convolutional

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

=

2	5	-1
-15		

Bias: $b = 0$

0x1 + 3x(-1) + 4x0 + 1x0 + 0x2 +
2x(-2) + 8x(-1) + 12x0 + 0x2 + 0
= -15

Convolutional

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

Bias: $b = 0$

=

2	5	-1
-15	-7	

3x1 + 4x(-1) + 0x0 + 0x0 + 2x2 +
0x(-2) + 12x(-1) + 0x0 + 1x2 + 0
= -7

Convolutional

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

=

2	5	-1
-15	-7	2

Bias: $b = 0$

4x1 + 0x(-1) + 2x0 + 2x0 + 0x2 +
1x(-2) + 0x(-1) + 1x0 + 0x2 + 0
= 2

Convolutional

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

=

2	5	-1
-15	-7	2
31		

Bias: $b = 0$

1x1 + 0x(-1) + 2x0 + 8x0 + 12x2
+ 0x(-2) + 0x(-1) + 6x0 + 3x2 + 0
= 31

Convolutional

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

Bias: $b = 0$

=

2	5	-1
-15	-7	2
31	-8	

0x1 + 2x(-1) + 0x0 + 12x0 + 0x2 +
1x(-2) + 6x(-1) + 3x0 + 2x2 + 0
= -8

Convolutional Neural Networks

Convolution computation example

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

1	-1	0
0	2	-2
-1	0	2

=

2	5	-1
-15	-7	2
31	-8	1

Bias: $b = 0$

2x1 + 0x(-1) + 1x0 + 0x0 + 1x2 +
0x(-2) + 3x(-1) + 2x0 + 0x2 + 0
= 1

Convolution example

Example from ML4Engineers book.

Filter based on finite approximation of Laplacian operator:

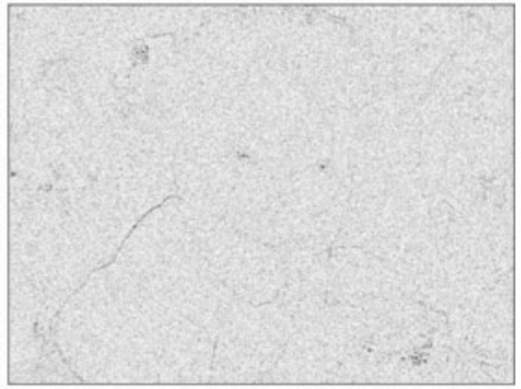
$$\Delta^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

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

(a)

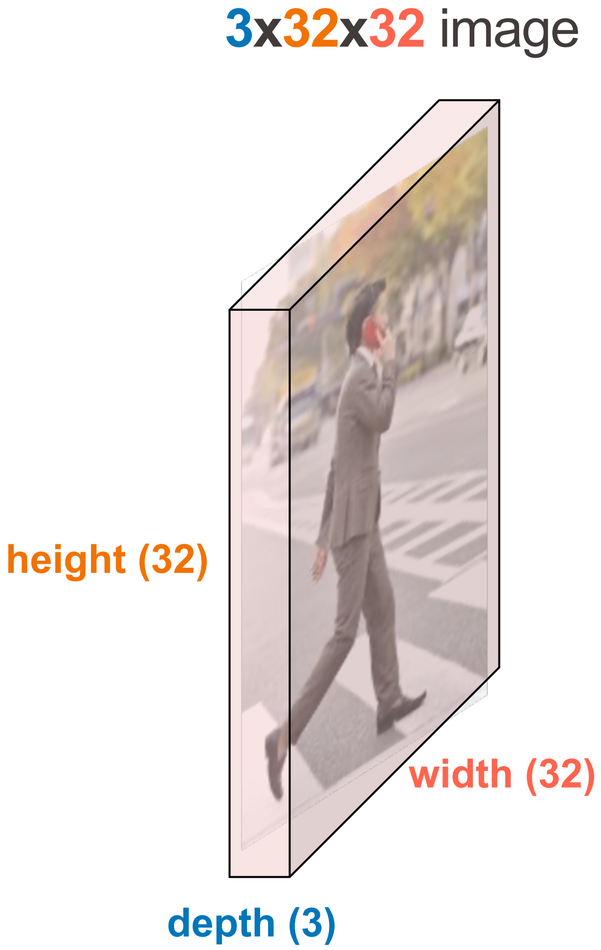
(b)

(c)



(d)

Convolutional Neural Networks (CNN)
Convolution Layer



In PyTorch, images are represented as

(CxHxW)

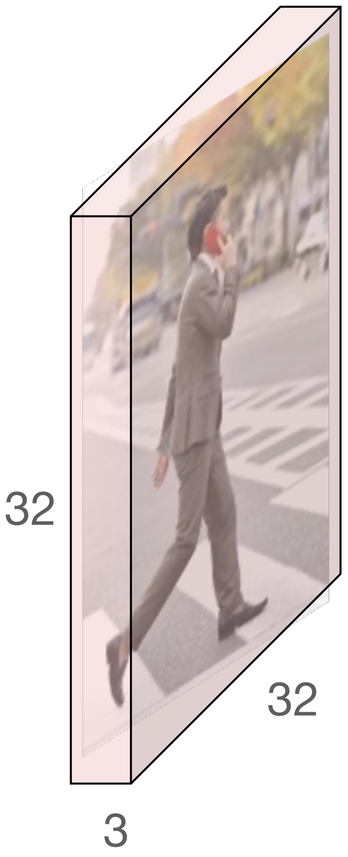
- C: number of channels (depth) colors : red, green, blue
- H: height
- W: width

A pixel can be represented by a vector of 3 color (R, G, B) intensities

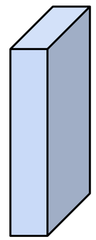
I(c, h, w)

colors height width

Convolutional Neural Networks
Convolution Layer



Filters always extend the full depth of the input volume



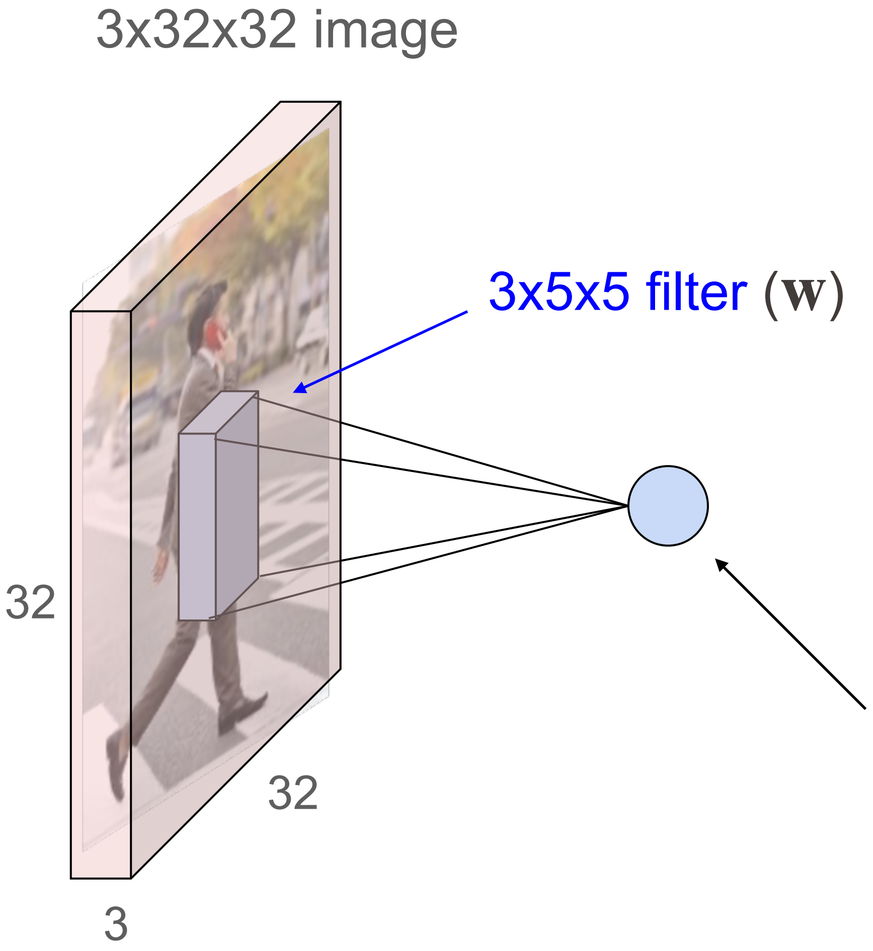
Convolve the filter with the image

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

Note:

Filters are sometimes referred to as **kernels**

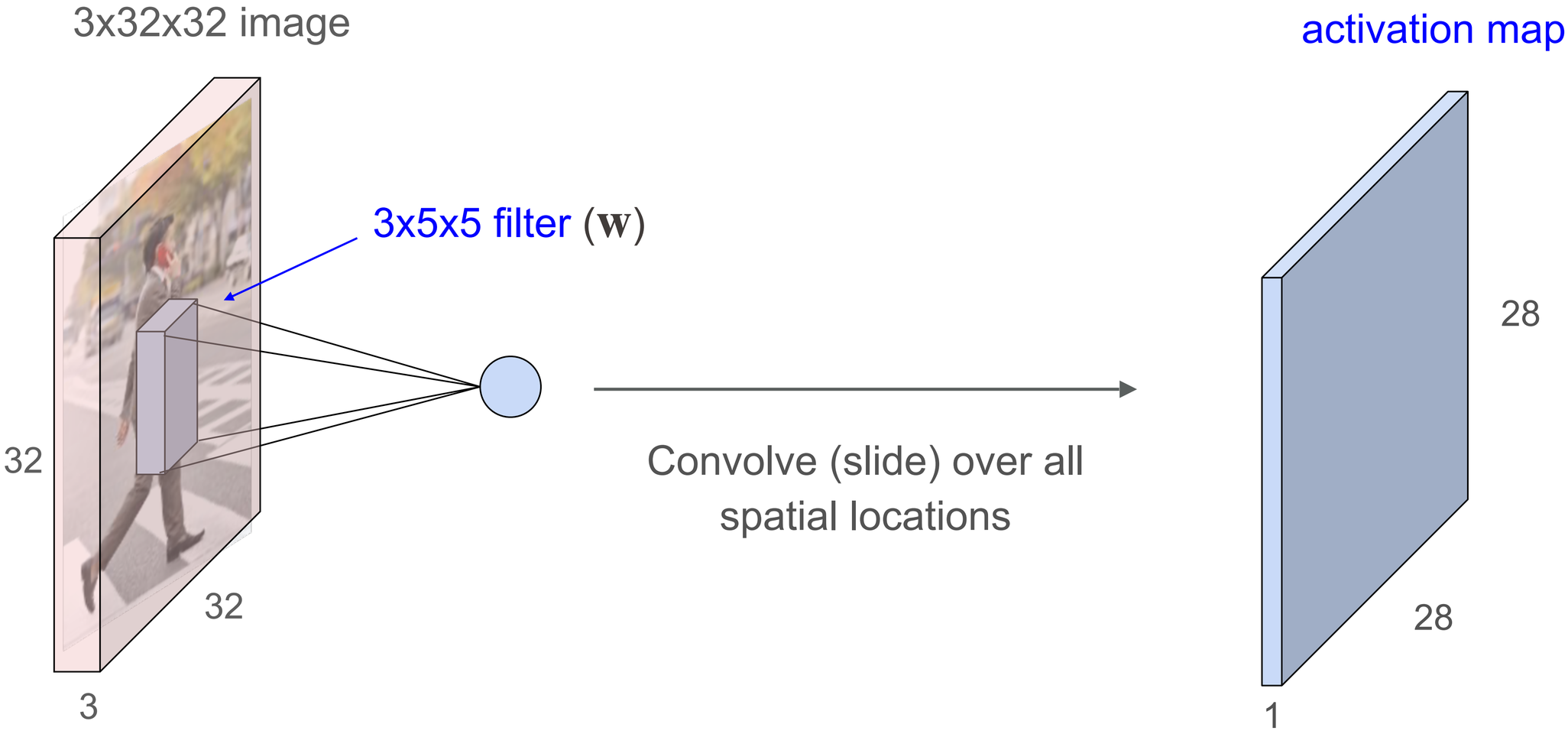
Convolutional Neural Networks
Convolution Layer



1 number:

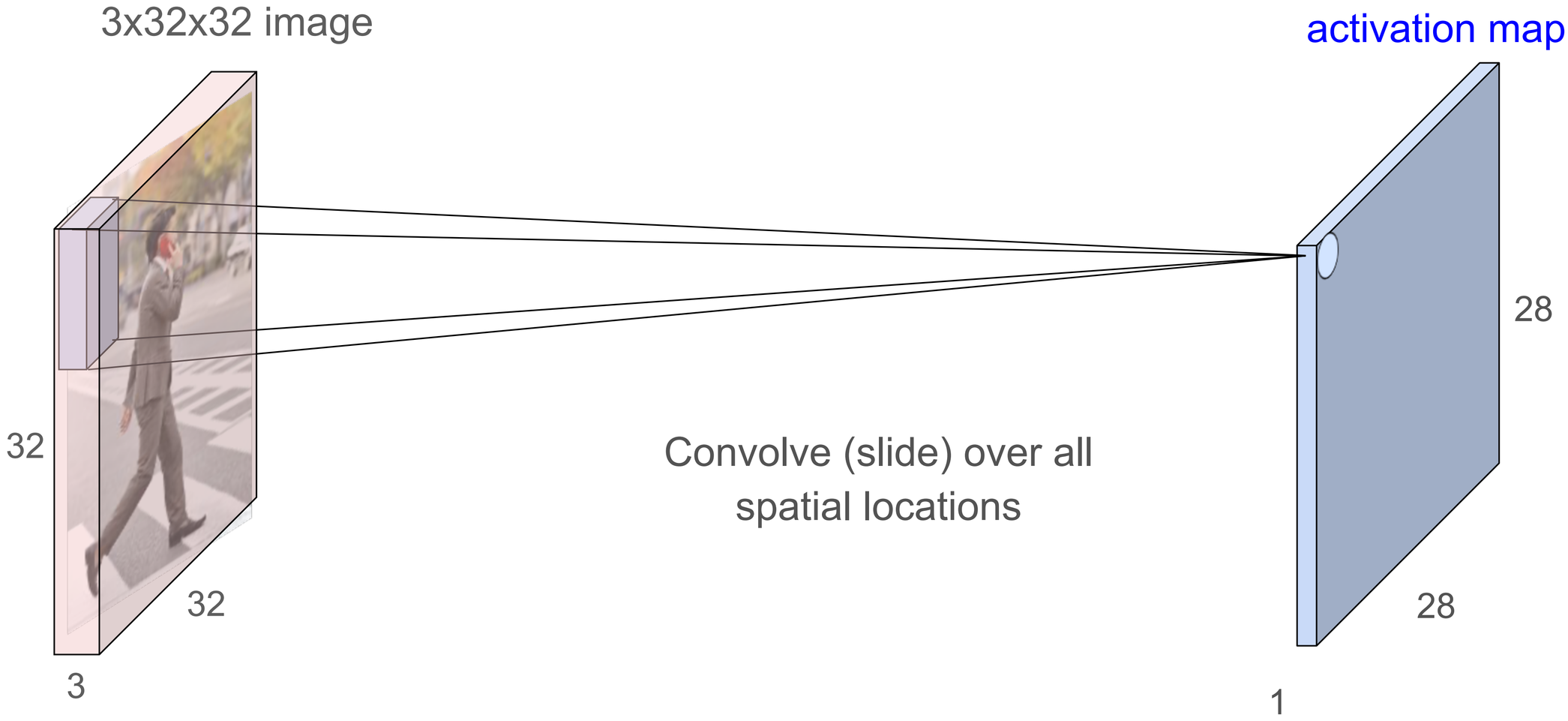
- The result of taking a dot product between the filter and a small 3x5x5 chunk of the image
  - (i.e. 5x5x3=75-dimensional dot product + bias)

Convolutional Neural Networks
Convolution Layer



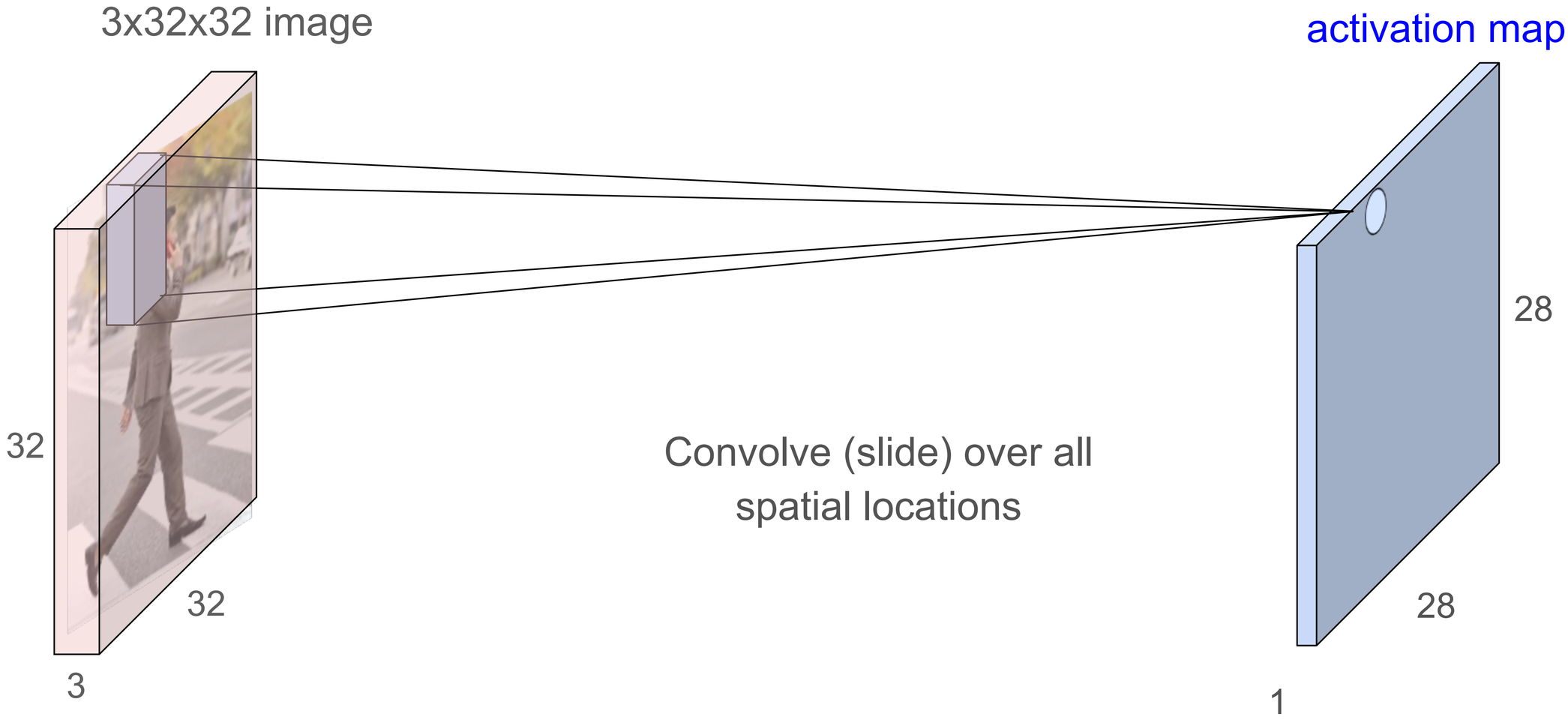
Convolutional Neural Networks

Convolution Layer

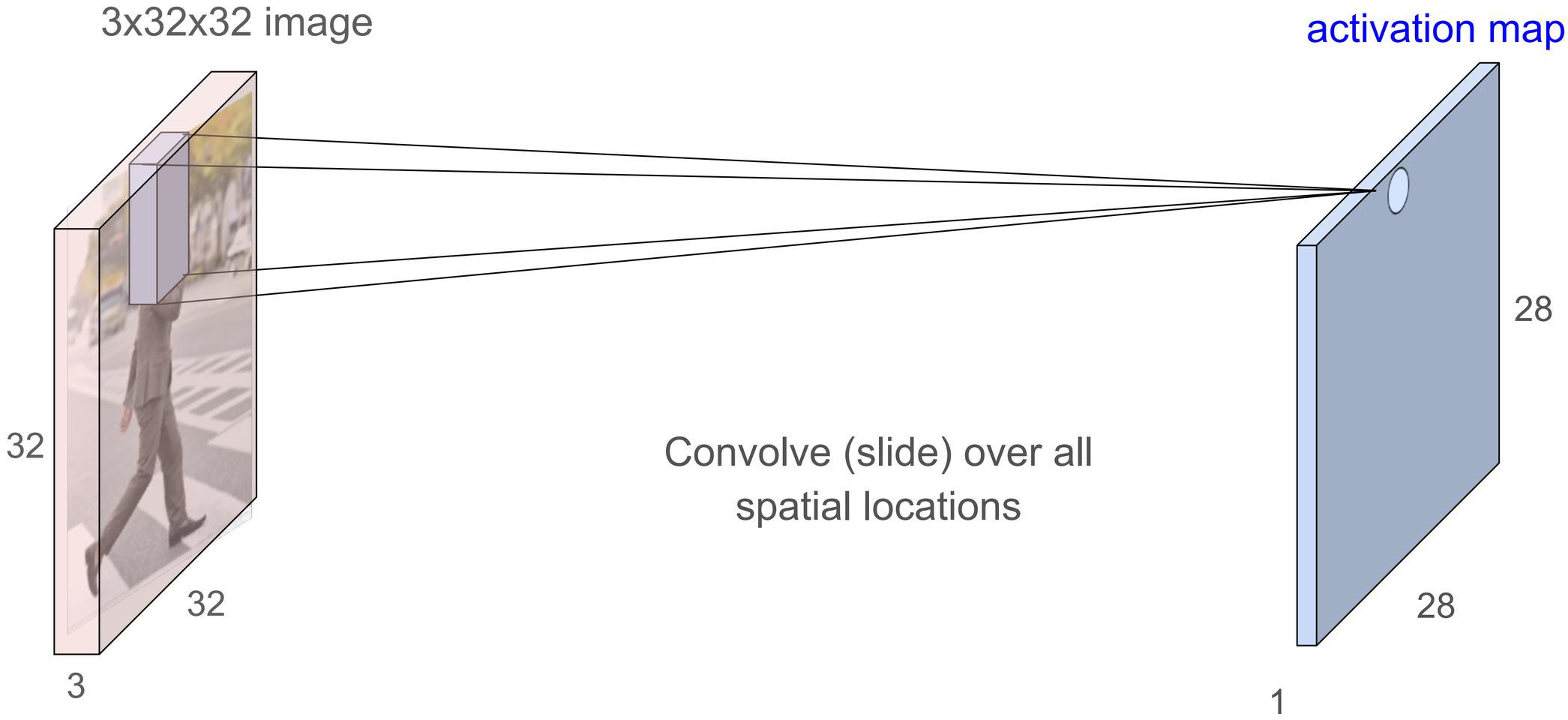


Convolutional Neural Networks

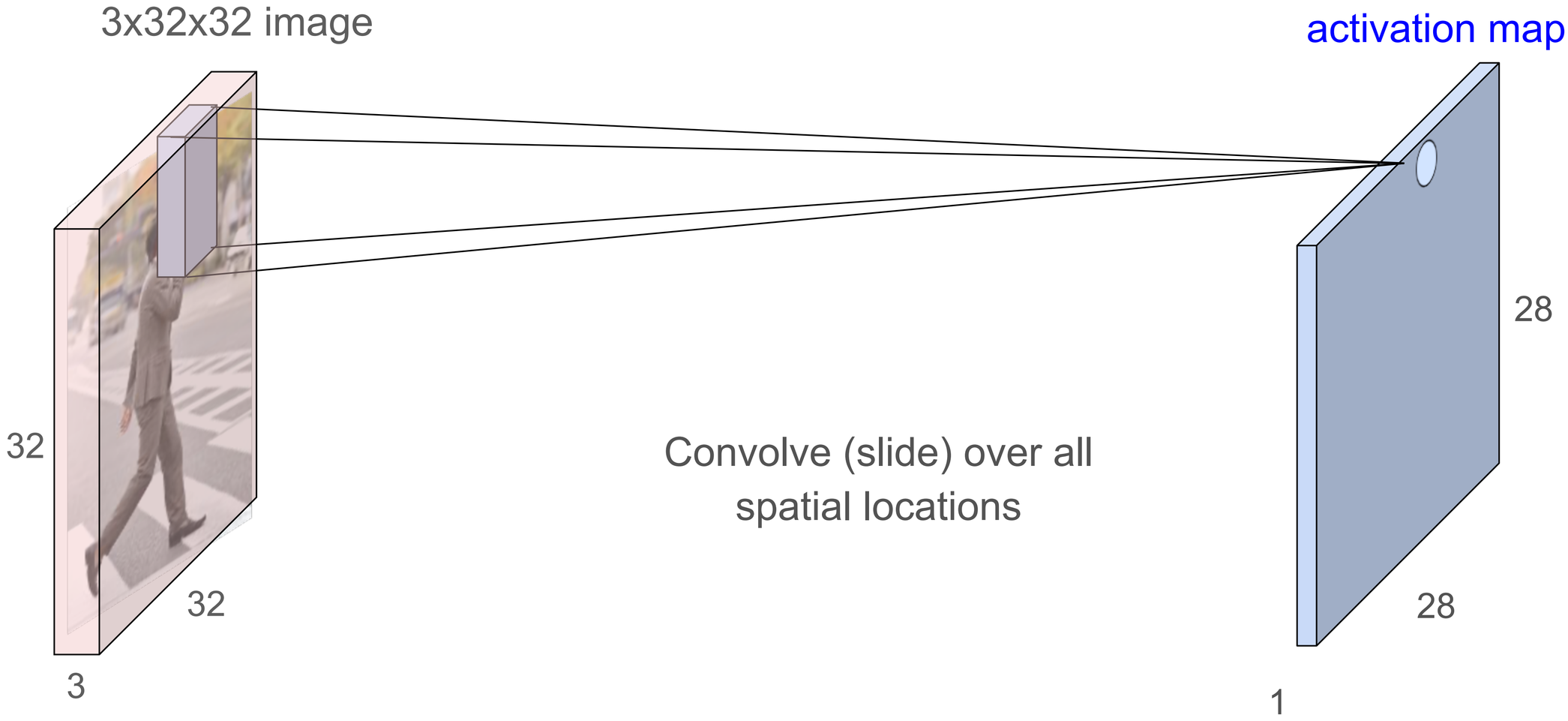
Convolution Layer



Convolutional Neural Networks
Convolution Layer

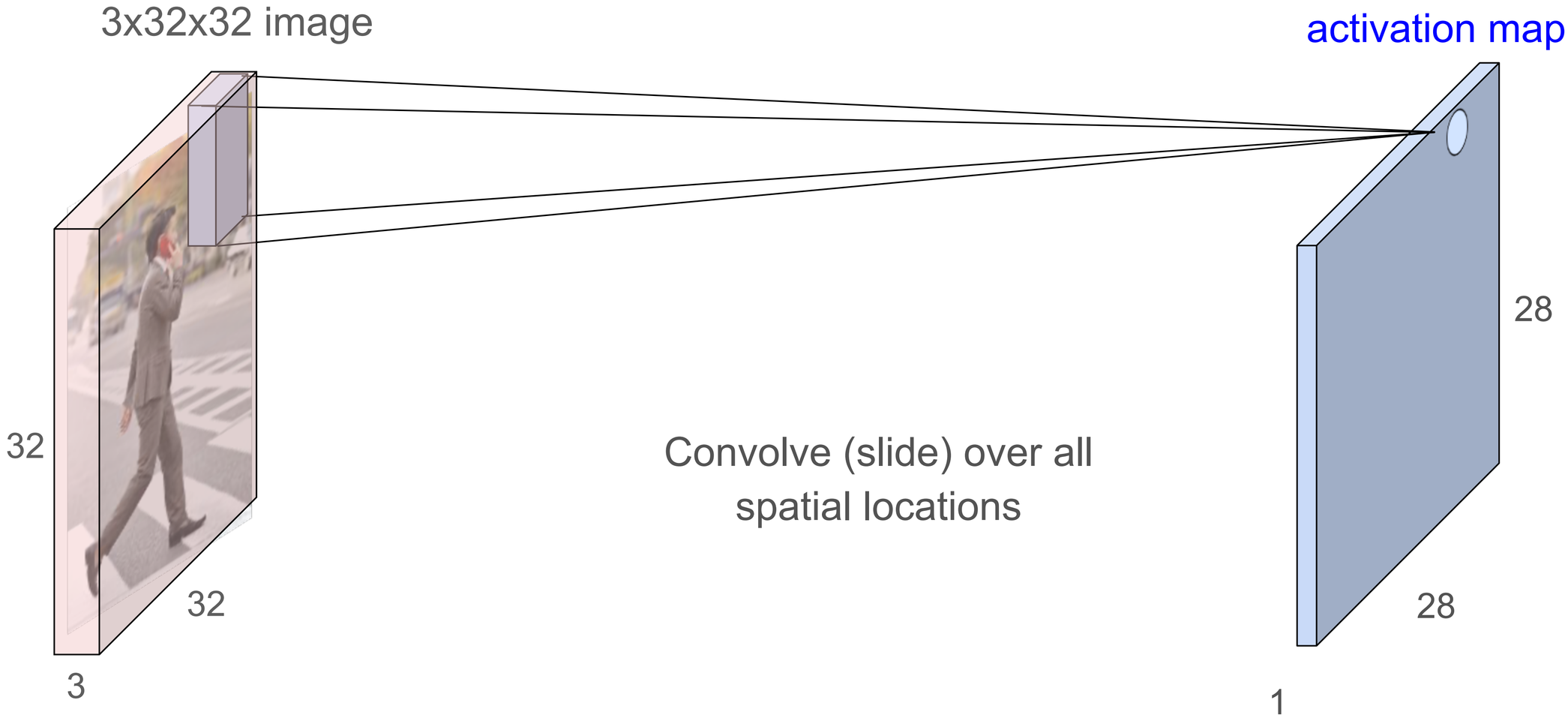


Convolutional Neural Networks
Convolution Layer



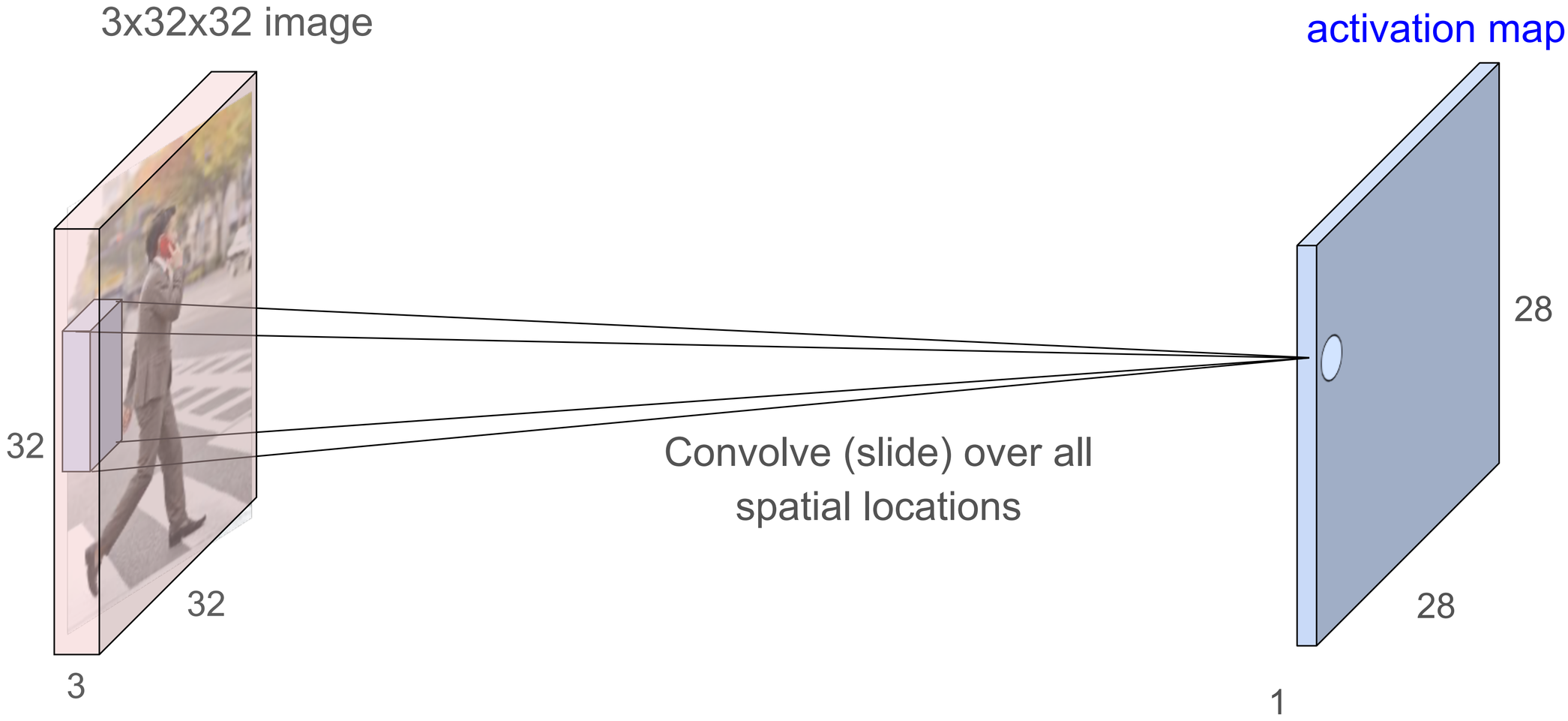
Convolutional Neural Networks

Convolution Layer



Convolutional Neural Networks

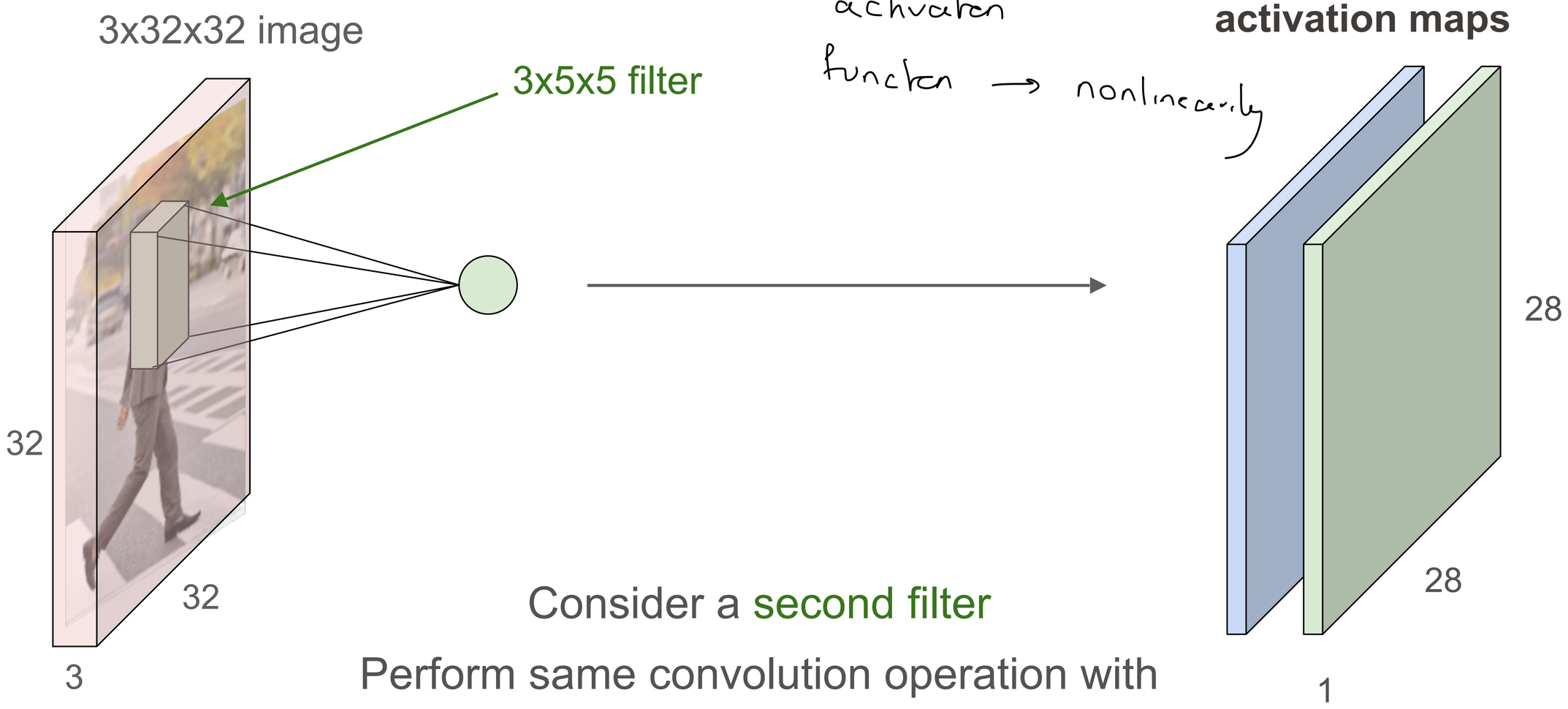
Convolution Layer



Convolutional Neural Networks

Convolution Layer

convolution : linear operation,
activation function → nonlinearity



Consider a second filter

Perform same convolution operation with this filter to get a second activation map

Pooling

Applying a function in a non-overlapping way to a signal

Need to determine the type of function and the ‘stride’

Example: $x = (0, 1, 1, 2, 3, 5, 8, 13)$

average-pooling of stride 4 :

$$z = (\text{avg}(x_{1:4}), \text{avg}(x_{5:8})) = (1, 7.25) \in \mathbb{R}^2$$

max-pooling of stride 4

$$z = (\max(x_{1:4}), \max(x_{5:8})) = (2, 13) \in \mathbb{R}^2$$

Pooling example

Max-pooling, stride of 2

Input (1x6x6)

3	0	1	0	2	4
0	1	8	12	0	0
4	0	0	3	2	2
2	0	1	0	1	1
3	2	0	6	0	5
1	0	6	0	0	9

Output (1x3x3)

3	12	4
4	3	2
3	6	9

Pooling example

Example from ML4Engineers book.

Max-pooling of stride 9 applied to (a)

(a)

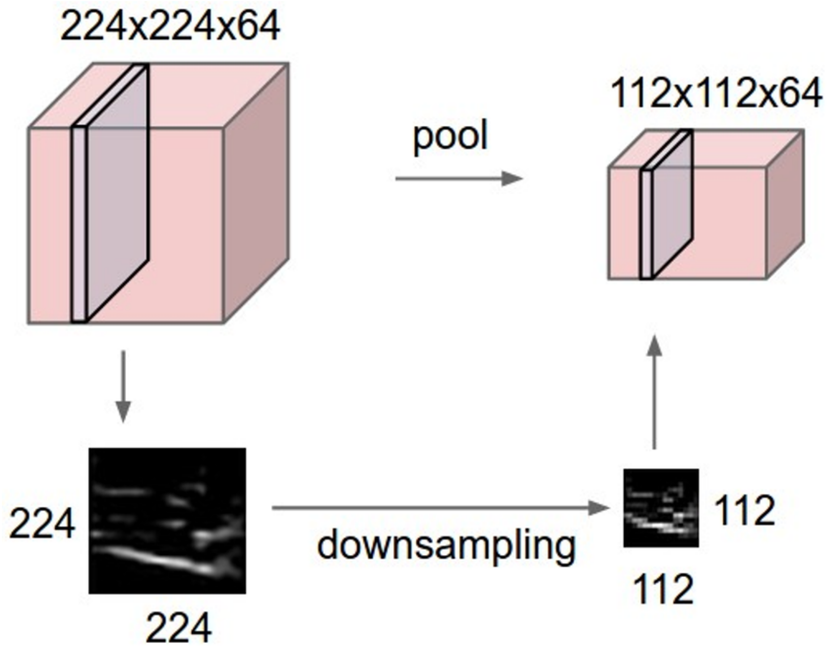
(b)

Convolutional Neural Networks
Pooling layer

CNNs may include pooling layers **to reduce the spatial size of the representation**

Pooling layers require two hyper-parameters: their spatial extent F and their stride S

- Most common layer uses 2x2 filters of stride 2 ($F = 2,\ S = 2$)



Convolutional applied with a stride

We can also apply convolution with a **stride of 2**

Input (1x5x5)

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

Filter (1x3x3)

1	-1	0
0	2	-3
-1	0	2

Bias: $b = 0$

Convolutional Neural Networks

Changing the stride

Back to our simple example, but change to **stride of 2**

Input (1x5x5)

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

Filter (1x3x3)

1	-1	0
0	2	-2
-1	0	2

Bias: $b = 0$

=

Output (1x2x2)

2	

1x1 + 0x(-1) + 3x0 + 0x0 + 3x2 +
4x(-2) + 1x(-1) + 0x0 + 2x2 + 0
= 2

Convolutional Neural Networks

Changing the stride

Back to our simple example, but change to **stride of 2**

Input (1x5x5)

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

Filter (1x3x3)

1	-1	0
0	2	-2
-1	0	2

Bias: $b = 0$

=

Output (1x2x2)

2	-1

3x1 + 0x(-1) + 2x0 + 4x0 + 0x2 +
2x(-2) + 2x(-1) + 0x0 + 1x2 + 0
= -1

Convolutional Neural Networks

Changing the stride

Back to our simple example, but change to **stride of 2**

Input (1x5x5)

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

Filter (1x3x3)

1	-1	0
0	2	-2
-1	0	2

Bias: $b = 0$

=

Output (1x2x2)

2	-1
31	

1x1 + 0x(-1) + 2x0 + 8x0 + 12x2
+ 0x(-2) + 0x(-1) + 6x0 + 3x2 + 0
= 31

Convolutional Neural Networks

Changing the stride

Back to our simple example, but change to **stride of 2**

Input (1x5x5)

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

•

Filter (1x3x3)

1	-1	0
0	2	-2
-1	0	2

Bias: $b = 0$

=

Output (1x2x2)

2	-1
31	1

2x1 + 0x(-1) + 1x0 + 0x0 + 1x2 +
0x(-2) + 3x(-1) + 2x0 + 0x2 + 0
= 1

Convolutional Neural Networks
Zero-padding

Height and width shrink quite quickly due to the repeated convolutions

To avoid this, we can add zero-padding:

Input (1x5x5)

1	0	3	0	2
0	3	4	0	2
1	0	2	0	1
8	12	0	1	0
0	6	3	2	0

Zero-padding = 1

Zero-padded input (1x7x7)

0	0	0	0	0	0	0
0	1	0	3	0	2	0
0	0	3	4	0	2	0
0	1	0	2	0	1	0
0	8	12	0	1	0	0
0	0	6	3	2	0	0
0	0	0	0	0	0	0

If we use a 3x3 filter with a stride of 1 on the padded input, we get a 5x5 output

→ same size as input

Convolutional Neural Networks
Convolution layer summary

The convolution layer:

- Accepts a volume of size $C_{in} \times H_1 \times W_1$

- Requires four hyper-parameters:
  - Number of filters K
  - Spatial extent of filters F
  - Stride S
  - Amount of zero padding P

- Produces of a volume of size $C_{out} \times H_2 \times W_2$ where:
  - $C_{out} = K$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $W_2 = (W_1 - F + 2P)/S + 1$

Convolutional Neural Networks
Convolution layer summary

The convolution layer:

- Accepts a volume of size $C_{in} \times H_1 \times W_1$

- Requires four hyper-parameters:
  - Number of filters K
  - Spatial extent of filters F
  - Stride S
  - Amount of zero (repetition) padding P

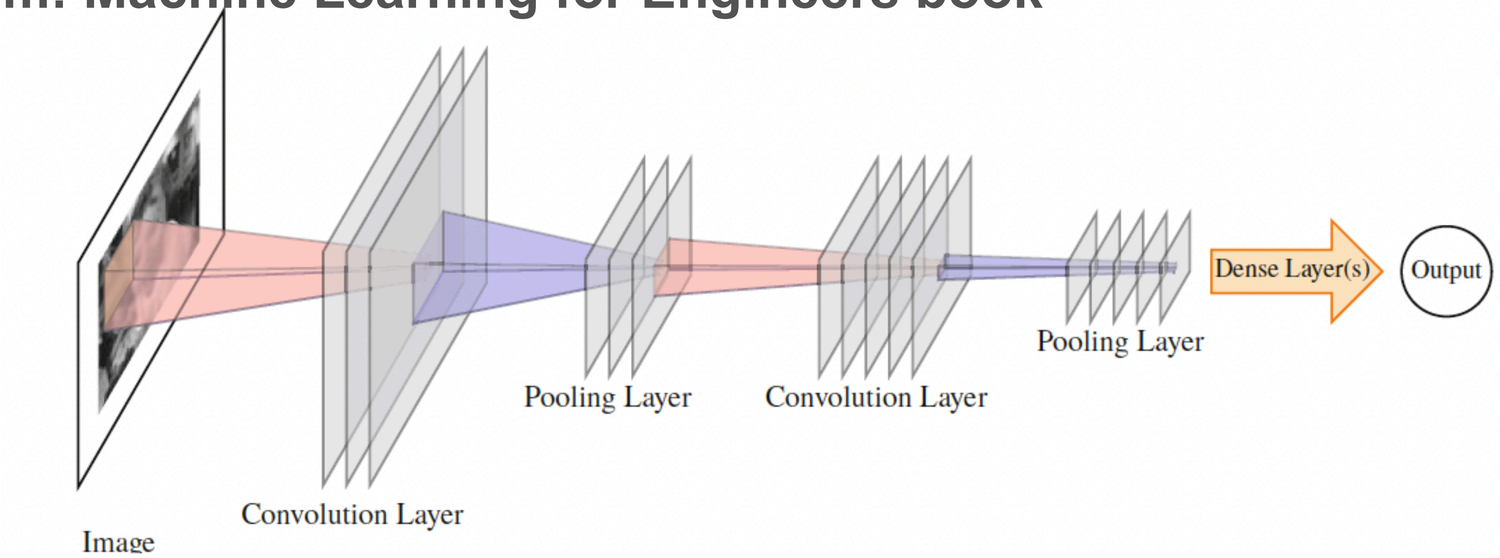
- Produces of a volume of size $C_{out} \times H_2 \times W_2$ where:
  - $C_{out} = K$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $W_2 = (W_1 - F + 2P)/S + 1$

Note:

There are $F \cdot F \cdot C_{in}$ weights per filter,
for a total of $(F \cdot F \cdot C_{in}) \cdot K$ weights
and K biases per layer

Convolutional neural net

From: Machine Learning for Engineers book



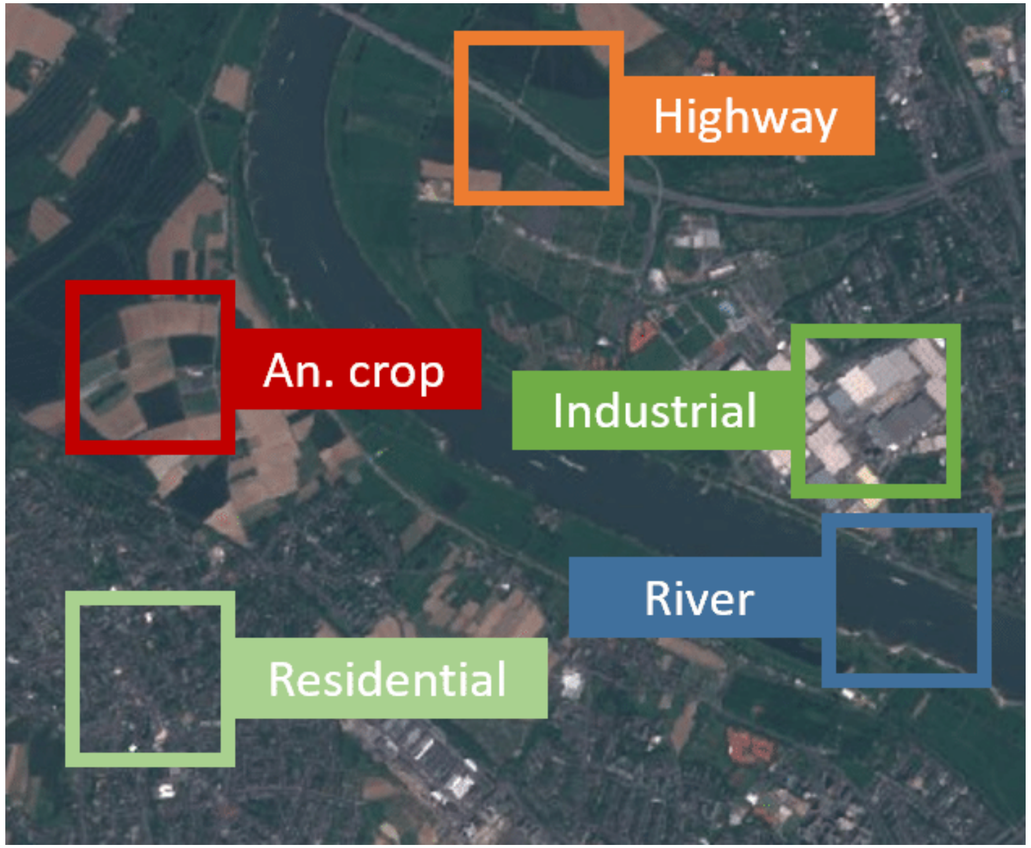
Usually the architecture is fixed for a given problem (object classification) based on trial and error

examples: LeNet-5, LeCun et al. ,1998, AlexNet Krizhevsky et al., 2012, GoogLeNet (Inception v1), etc.

It can be applied to transfer learning to a new problem

Python exercises this week

- You will apply CNN to two datasets
  - Dataset 1: from last week, digit recognition
  - Dataset 2: satellite images classification (bonus)



Summary

Neural networks: nonlinear function approximations (predictors)

- Structure: compositions of linear functions and nonlinear activation function ->
  - Strong function approximation property
  - Gradient computation

Convolutional neural networks: can help keeping spatial/temporal structure

- Image or audio processing

Disadvantages:

- Highly non-convex loss functions
- Energy and time for training
- Interpretability

Additional reading (optional): transfer learning

Train network for a task

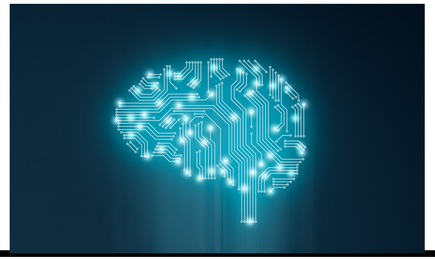
- Example: image classification
- Requires large number of training data, and training resources

Modify the trained network for a different task (transfer learning)

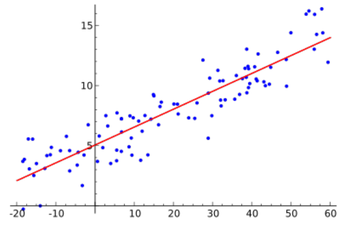
- Why? Can address limited data and time/resources for training

Case study 6.5 from Machine Learning for Engineers book: “Finding volcanos on Venus with pre-fit models”

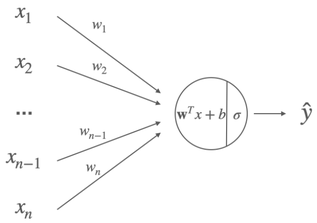
Introduction



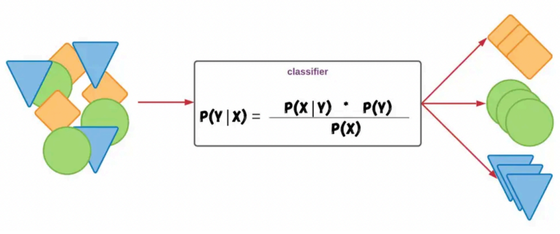
Linear regression



Logistic regression

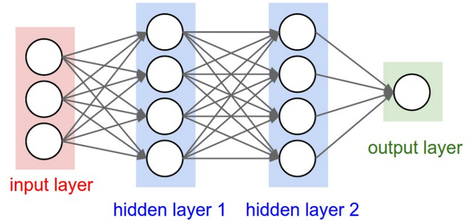


Naive Bayes

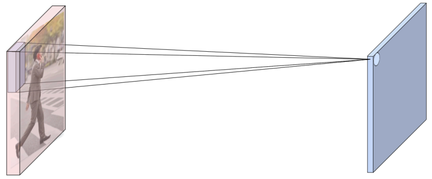


KNN

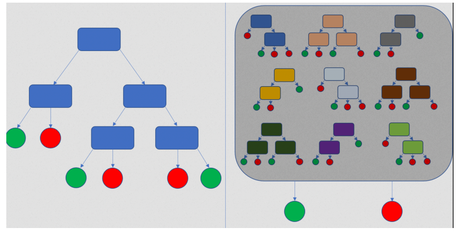
Neural networks



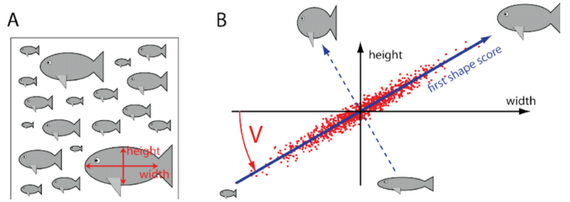
Convolutional neural networks



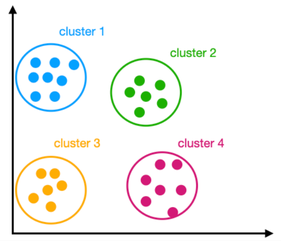
Decision-trees



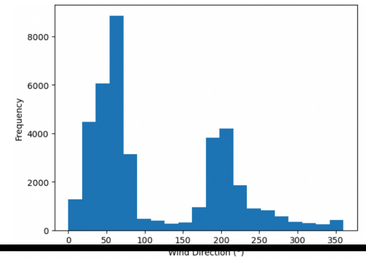
Dimensionality reduction



Clustering



Reinforcement learning



AI ethics