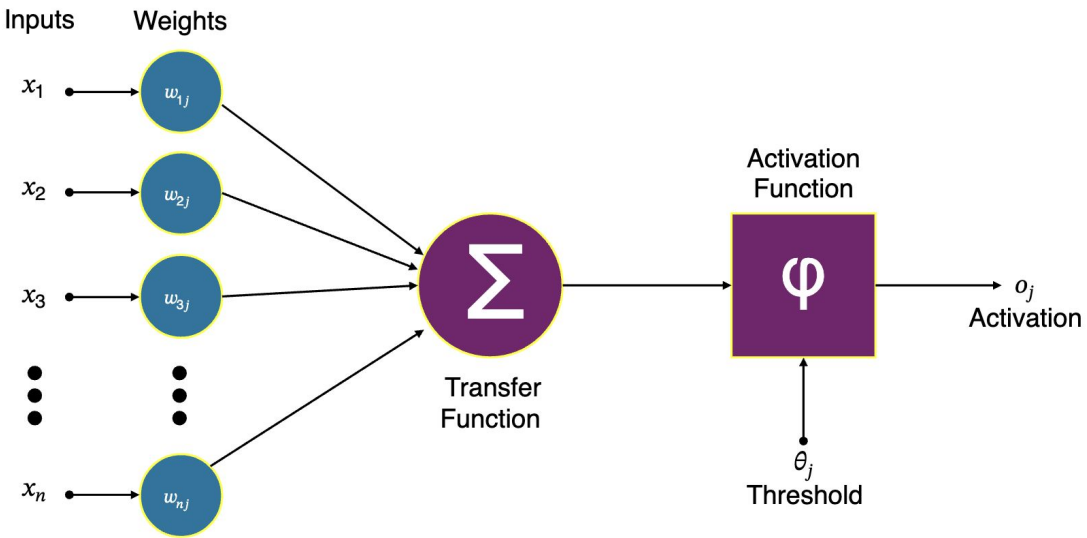
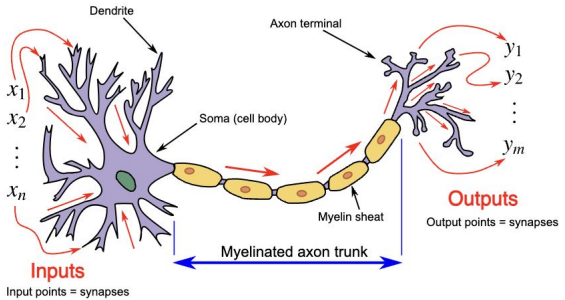
Week 8
Deep Learning

Content

- Feedforward Neural Networks (FFNN)
- Back-Propagation
- Convolutional Neural Network (CNN)
- Training ConvNets
- Example: CNN in Python

Recap: Artificial neuron structure

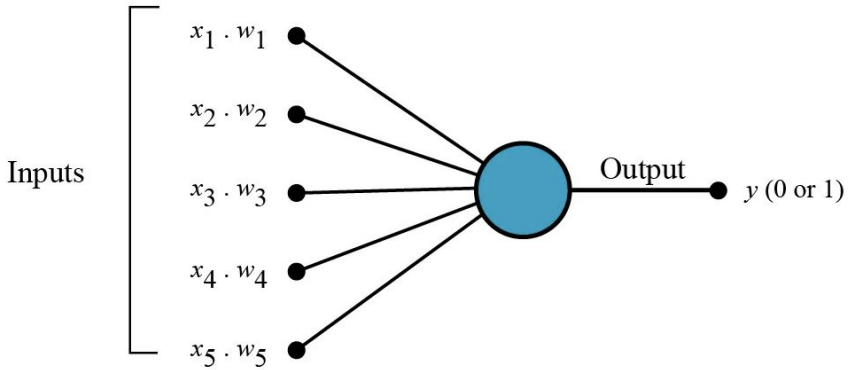


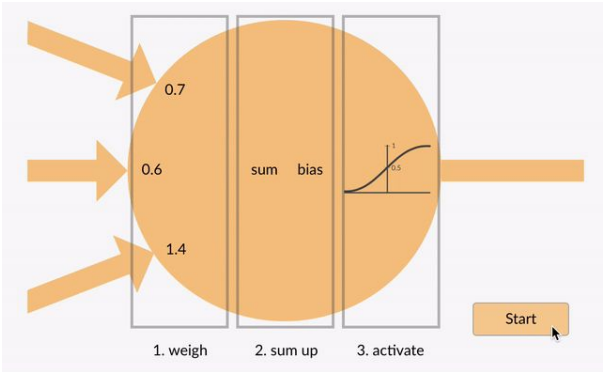


Biological neuron model

https://en.wikipedia.org/wiki/Artificial_neuron

Recap: A single Neuron from an Artificial Neural Network (ANN)





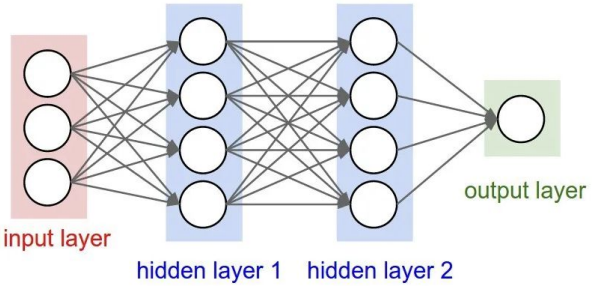
One of the first ANNs was known as the perceptron, and it consisted of only a single neuron.

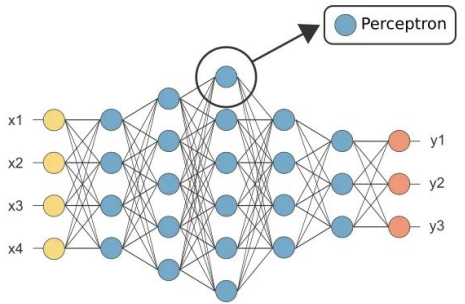
https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d

Feedforward Neural Networks (FFNN)

A fully-connected feed-forward neural network (FFNN) — aka A multi-layered perceptron (MLP)

- A type of artificial neural network where connections between nodes do not form a cycle. They are called “feedforward” because data flows in one direction—from input to output.
- Consists of an input layer, one or more hidden layers, and an output layer.
- Key Components
    - The **Perceptron**: one of the first Neural Networks, is made of just a single Neuron
    - **Neurons**: Basic units of the network, performing computations.
    - **Layers**:
        - **Input** Layer: Receives the initial data.
        - **Hidden** Layers: Intermediate layers that perform computations and feature extraction.
        - **Output** Layer: Produces the final output.



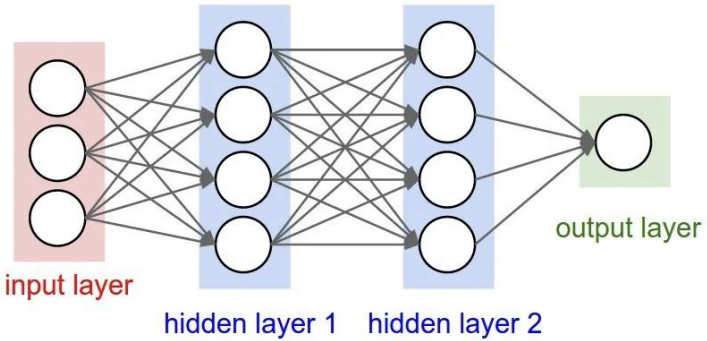


https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a309d

Feedforward Neural Networks (FFNN)

A network of artificial neurons with multiple layers

- A type of artificial neural network where connections between nodes do not form a cycle. They are called “feedforward” because data flows in one direction—from input to output.
- Consists of an input layer, one or more hidden layers, and an output layer.
- Key Components
    - Neurons: Basic units of the network, performing computations.
    - Layers:
        - Input Layer: Receives the initial data.
        - Hidden Layers: Intermediate layers that perform computations and feature extraction.
        - Output Layer: Produces the final output.
- Activation Functions
    - Purpose: Introduce non-linearity into the model, enabl
    - Common Types:
        - Sigmoid
        - ReLU (Rectified Linear Unit)
        - Tanh



Feedforward Neural Networks (FFNN)

Example: Learning XOR

- **XOR Function ("Exclusive or")**

x_1	x_2	$x_1\ XOR\ x_2$
0	0	0
0	1	1
1	0	1
1	1	0

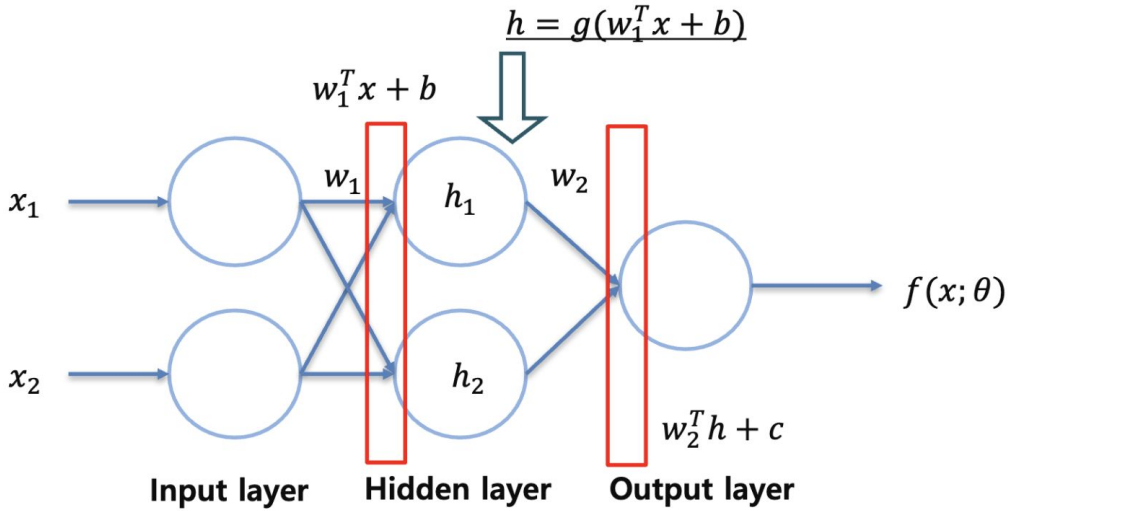
- **X = {[0, 0]^T, [0, 1]^T, [1, 0]^T, [1, 1]^T}**
- **Mean squared error (MSE) loss function**

$$J(\boldsymbol{\theta}) = \frac{1}{4}\sum_{\mathbf{x}\in\mathbb{X}}\left(f^*(\boldsymbol{x}) - f(\boldsymbol{x};\boldsymbol{\theta})\right)^2$$

- $f^*(x)$: *correct answer*
- $f(x;\theta)$: *calculated result by neural network*

Feedforward Neural Networks (FFNN)

Example: Learning XOR



(g = activation function, b, c = bias)

$$f(x;\theta) = f(x;w_1,b,w_2,c) = w_2^T g(w_1^T x + b) + c$$

Feedforward Neural Networks (FFNN)

Gradient-based Learning

- **Loss function of neural network is non-convex**
  - Trained by using iterative, gradient-based optimizers
- **Cost function**
  - Learning conditional distribution
  - Learning conditional statistics
- **Output layers**
  - Linear units for Guassian output distributions
  - Sigmoid units
  - Softmax units

Feedforward Neural Networks (FFNN)

Learning Conditional Distribution

- **Negative log-likelihood as cross-entropy between training data and model distribution**

$$J(\boldsymbol{\theta}) = H\left(\hat{\boldsymbol{p}}_{data}, \boldsymbol{p}_{model}(\boldsymbol{y}|\boldsymbol{x})\right) = -\sum_{\boldsymbol{x},\boldsymbol{y}\sim\hat{\boldsymbol{p}}_{data}} \hat{\boldsymbol{p}}_{data} \boldsymbol{log}\, \boldsymbol{p}_{model}(\boldsymbol{y}|\boldsymbol{x})$$

$$J(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{\boldsymbol{p}}_{data}} \boldsymbol{log} \boldsymbol{p}_{model}(\boldsymbol{y}|\boldsymbol{x})$$

- $\hat{\boldsymbol{p}}_{data}$: $\boldsymbol{training\ data - generating\ distribution}$
- $\boldsymbol{p}_{model}(\boldsymbol{y}|\boldsymbol{x})$: $\boldsymbol{probability\ distribution\ estimating}\ \hat{\boldsymbol{p}}_{data}$
- $H\left(\hat{\boldsymbol{p}}_{data}, \boldsymbol{p}_{model}(\boldsymbol{y}|\boldsymbol{x})\right)$: $\boldsymbol{cross\ entropy\ between}\ \hat{\boldsymbol{p}}_{data}\ \boldsymbol{and}\ \boldsymbol{p}_{model}$

- ***log*** **function undoes the exp of some output units**

Feedforward Neural Networks (FFNN)

Learning Conditional Distribution

- **Mean Square Error (MSE)**

$$f^* = \arg\min_f \mathbb{E}_{x,y\sim p_{data}} \left|\left|y - f(x)\right|\right|^2$$

- **Mean Absolute Error (MAE)**

$$f^* = \arg\min_f \mathbb{E}_{x,y\sim p_{data}} \left|\left|y - f(x)\right|\right|_1$$

- **MSE, MAE often lead to poor results when used with gradient-based learning**

Feedforward Neural Networks (FFNN)

Linear Units for Gaussian Output Distributions

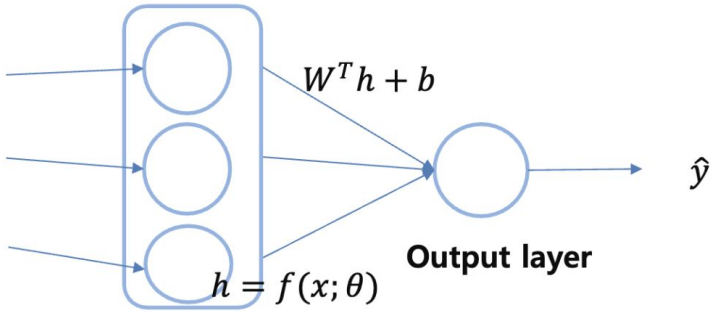
- **Feature h, produce a vector y hat**

$$\hat{y} = W^T h + b$$

- **The produce the mean of a conditional Guassian distribution**

$$p(y|x) = \mathcal{N}(y; \hat{y}, I)$$

- **No saturation → little difficult to gradient-based optimization**



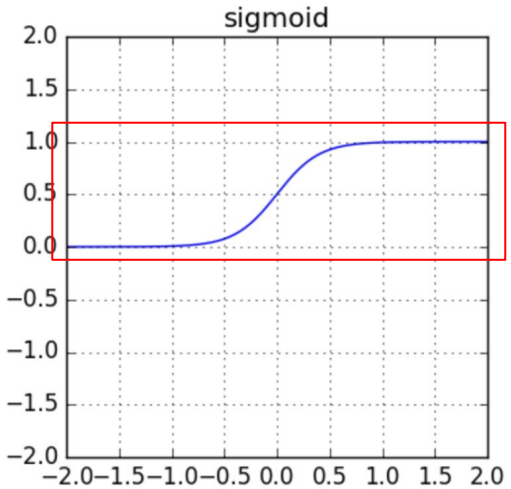
Feedforward Neural Networks (FFNN)

Sigmoid Units (Two plots, “soft” meanings)

- Binary cl $\hat{y} = \sigma(w^T h + b)$
- Output

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

- Saturate to 0 and 1



Feedforward Neural Networks (FFNN)

Softmax Units

- **Multiclass classification**
- **To generalize to the case of a discrete variable with n values, vector y hat, with** $\hat{y}_i = P(y = i|x)$

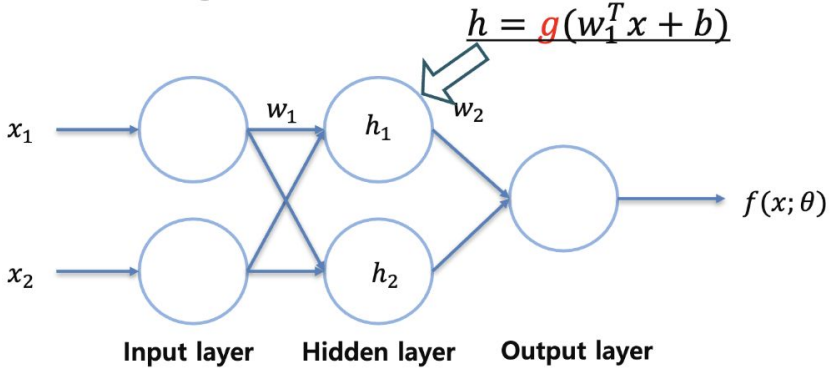
$$z = W^T h + b \left(z_i = log\, \tilde{P}(y = i|x)\right)$$

$$softmax(z)_i = \frac{exp(z_i)}{\Sigma_j exp(z_j)}$$

Feedforward Neural Networks (FFNN)

Hidden Units (Which active functions always use)

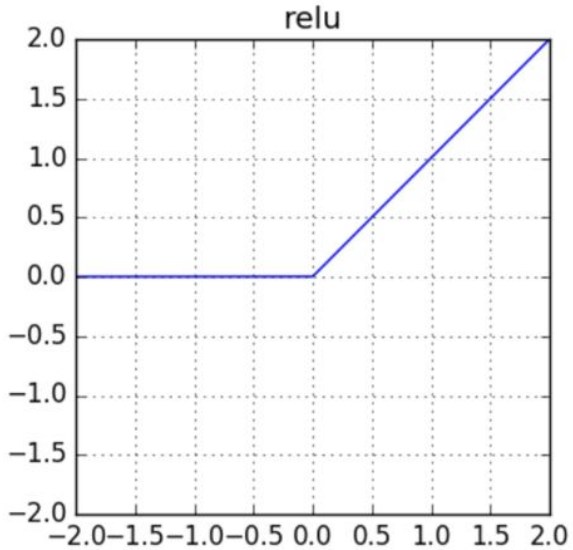
- **How to choose the type of hidden unit to use in the hidden layers of the model**
- **Input:** $\boldsymbol{z = W^T x + b}$
- **Activation function g(z)**
  - Rectified Linear Units (ReLU)
  - Logistic Sigmoid and Hyperbolic Tangent



Feedforward Neural Networks (FFNN)

Rectified Linear Units (ReLU)

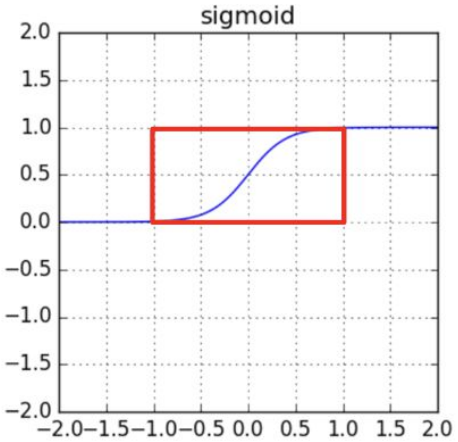
- **Activation function g(z) = max(0, z)**
- **If model's behavior is closer to linear, models are easier to optimize**

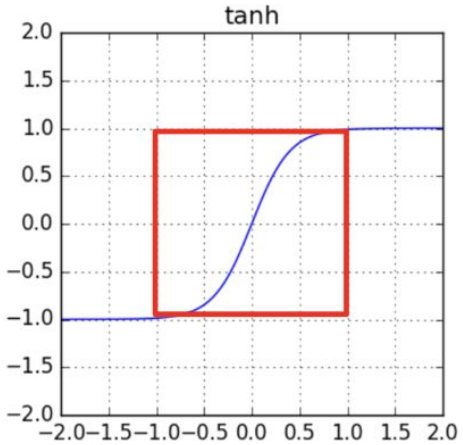


Feedforward Neural Networks (FFNN)

Logistic Sigmoid and Hyperbolic Tangent

- **Activation function**
  - $g(z) = \sigma(z)$
  - $g(z) - tanh(z) = 2\sigma(2z) - 1$
- **Saturation of sigmoidal units make gradient-based learning difficult**

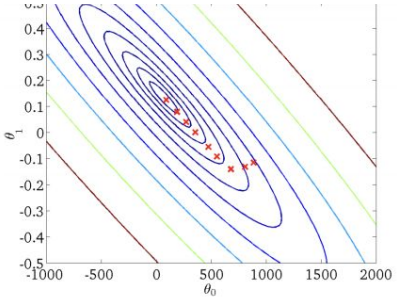




Back-Propagation

Recap: Gradient Descent

- Recall: gradient descent moves opposite the gradient (the direction of steepest descent)



- Weight space for a multilayer neural net: one coordinate for each weight or bias of the network, in all the layers
- Conceptually, not any different from what we've seen so far — just higher dimensional and harder to visualize!
- We want to compute the cost gradient dE/dw, which is the vector of partial derivatives.
    - This is the average of dL/dw over all the training examples, so in this lecture we focus on computing dL/dw.

Back-Propagation

Recap: Univariate Chain Rule (Add figure of “Movement”)

- We’ve already been using the univariate Chain Rule.
- Recall: if f (x) and x(t) are univariate functions, then

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t)) = \frac{\mathrm{d}f}{\mathrm{d}x} \frac{\mathrm{d}x}{\mathrm{d}t}$$

Back-Propagation

Recap: Univariate Chain Rule

- Recall: Univariate logistic least squares model

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

- Let’s compute the loss derivatives.

Back-Propagation

Recap: Univariate Chain Rule

$$\mathcal{L} = \frac{1}{2}(\sigma(wx+b)-t)^2$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\ &= \frac{1}{2}\frac{\partial}{\partial w}(\sigma(wx+b)-t)^2 \\ &= (\sigma(wx+b)-t)\frac{\partial}{\partial w}(\sigma(wx+b)-t) \\ &= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial w}(wx+b) \\ &= (\sigma(wx+b)-t)\sigma'(wx+b)x\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial}{\partial b}\left[\frac{1}{2}(\sigma(wx+b)-t)^2\right] \\ &= \frac{1}{2}\frac{\partial}{\partial b}(\sigma(wx+b)-t)^2 \\ &= (\sigma(wx+b)-t)\frac{\partial}{\partial b}(\sigma(wx+b)-t) \\ &= (\sigma(wx+b)-t)\sigma'(wx+b)\frac{\partial}{\partial b}(wx+b) \\ &= (\sigma(wx+b)-t)\sigma'(wx+b)\end{aligned}$$

What are the disadvantages of this approach?

Back-Propagation

Recap: Univariate Chain Rule

- A more structured way to do it

Computing the loss

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives

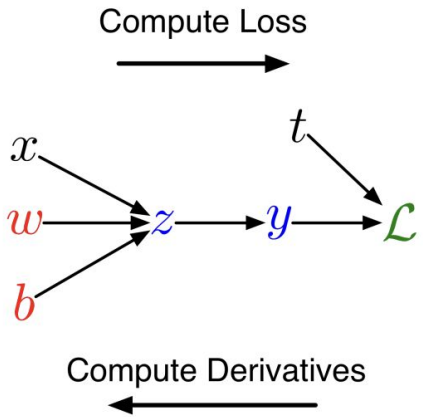
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = y - t$$
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\sigma'(z)$$
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}x$$
$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

Remember, the goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

Back-Propagation

Recap: Univariate Chain Rule

- We can diagram out the computations using a computation graph.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.



Back-Propagation

Recap: Univariate Chain Rule

- A slightly more convenient notation:
  - Use $\overline{y}$ to denote the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}y$,sometimes called the error signal.
  - This emphasizes that the error signals are just values our program is computing (rather than a mathematical operation).
  - This is not a standard notation, but I couldn't find another one that I liked.

Computing the loss

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives

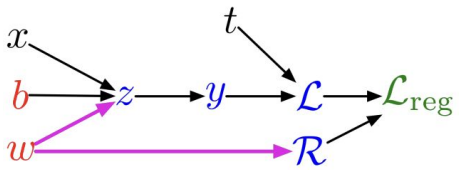
$$\overline{y} = y - t$$
$$\overline{z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\,x$$
$$\overline{b} = \overline{z}$$

Back-Propagation

Recap: Multivariate Chain Rule

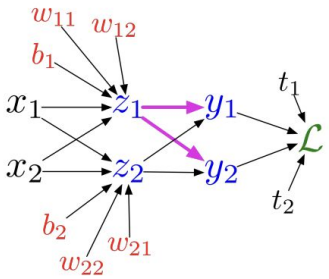
- Problem: what if the computation graph has fan-out > 1? This requires the multivariate Chain Rule!

L2-Regularized regression



$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Multiclass logistic regression



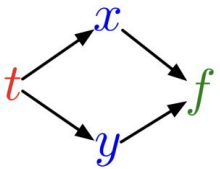
$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$
$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$
$$\mathcal{L} = -\sum_k t_k \log y_k$$

Back-Propagation

Recap: Multivariate Chain Rule

- Suppose we have a function f (x, y) and functions x(t) and y(t). (All the variables here are scalar-valued.) Then

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t), y(t)) = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$



- Example:

$$f(x, y) = y + e^{xy}$$
$$x(t) = \cos t$$
$$y(t) = t^2$$

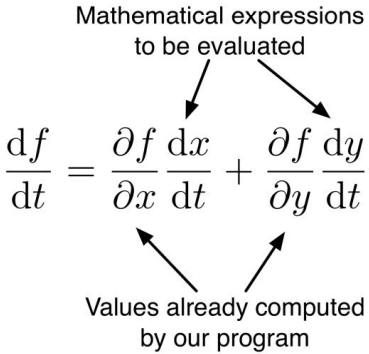
- Plug in to Chain Rule:

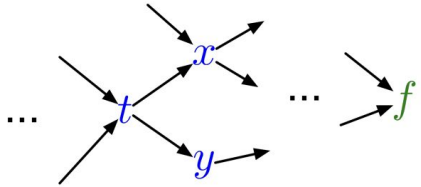
$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}t} &= \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t\end{aligned}$$

Back-Propagation

Recap: Multivariate Chain Rule

- In the context of backpropagation:





- In our notation:

$$\bar{t} = \bar{x}\frac{\mathrm{d}x}{\mathrm{d}t} + \bar{y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

Back-Propagation

Full backpropagation algorithm

- Let v1, . . . , vN be a topological ordering of the computation graph (i.e. parents come before children.) vN denotes the variable we're trying to compute derivatives of (e.g. loss).

Forward pass

For $i = 1, \ldots, N$

Compute v_i as a function of $\mathrm{Pa}(v_i)$

Backward pass

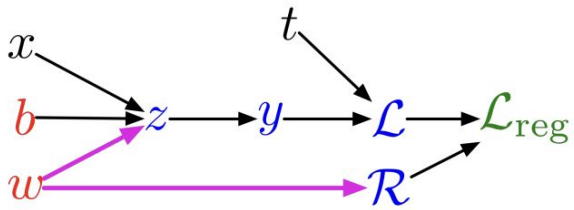
$\overline{v_N} = 1$

For $i = N - 1, \ldots, 1$

$\overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$

Back-Propagation

Example: univariate logistic least squares regression



Forward pass:

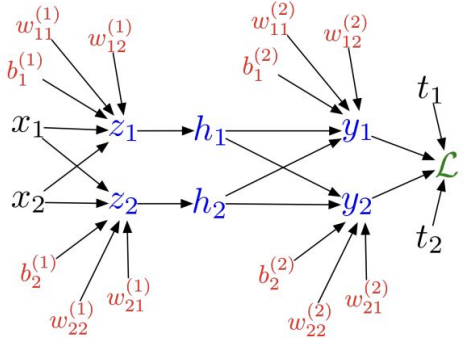
$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Backward pass:

$$\overline{\mathcal{L}_{\text{reg}}} = 1$$
$$\overline{\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\,\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\,\lambda$$
$$\overline{\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}\,\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}$$
$$\overline{y} = \overline{\mathcal{L}}\,\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = \overline{\mathcal{L}}\,(y - t)$$
$$\overline{z} = \overline{y}\,\frac{\mathrm{d}y}{\mathrm{d}z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\,\frac{\partial z}{\partial w} + \overline{\mathcal{R}}\,\frac{\mathrm{d}\mathcal{R}}{\mathrm{d}w} = \overline{z}\,x + \overline{\mathcal{R}}\,w$$
$$\overline{b} = \overline{z}\,\frac{\partial z}{\partial b} = \overline{z}$$

Back-Propagation

Multilayer Perceptron (multiple outputs):



Forward pass:

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}} (y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k} \, h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

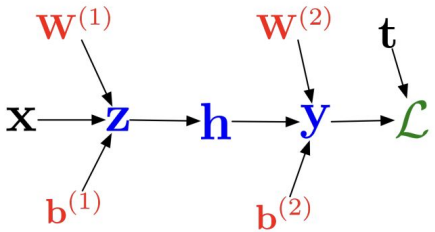
$$\overline{z_i} = \overline{h_i} \, \sigma'(z_i)$$

$$\overline{w_{ij}^{(1)}} = \overline{z_i} \, x_j$$

$$\overline{b_i^{(1)}} = \overline{z_i}$$

Back-Propagation

In vectorized form:



Forward pass:

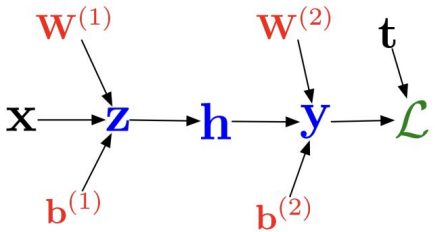
$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{h} = \sigma(\mathbf{z})$$
$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$
$$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$
$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$
$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$
$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$
$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$
$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$
$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Back-Propagation

In vectorized form: (Add an Example, simple vector to calculate, explain it)



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{h} = \sigma(\mathbf{z})$$
$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

Backward pass:

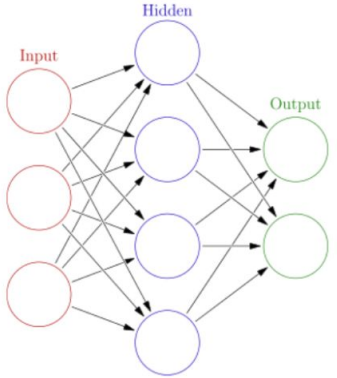
$$\overline{\mathcal{L}} = 1$$
$$\overline{\mathbf{y}} = \overline{\mathcal{L}}\,(\mathbf{y} - \mathbf{t})$$
$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$
$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$
$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$
$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$
$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$
$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

Convolutional Neural Network (CNN)

Introduction

- Multi-Layer Perceptron (MLP) for image recognition task
    - Use intensity of each pixel as 1D input vector
    - There several problems such as:
        - Required extremely large number of parameter
        - No invariance to shifting, scaling
        - Unable to deal with variable size input data

28x28 image



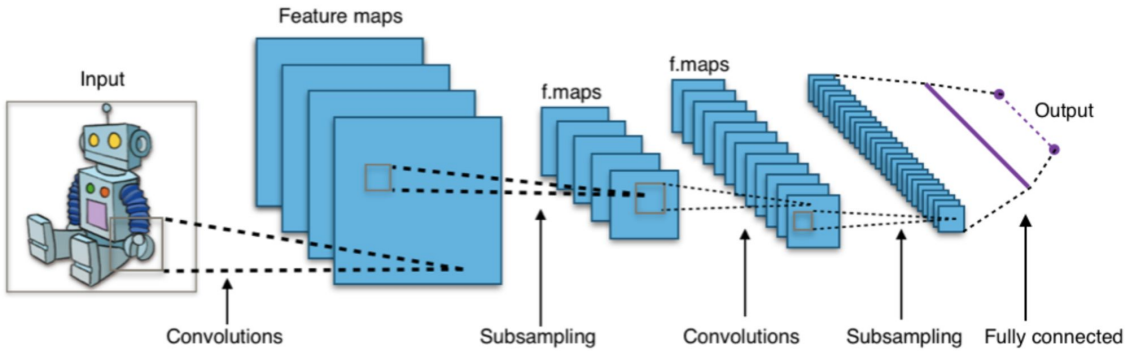
784 input neurons and large number of parameters

https://en.wikipedia.org/wiki/Neural_network_(machine_learning)

Convolutional Neural Network (CNN)

Introduction

- The image contains information about Convolutional Neural Networks (CNNs). Here are the key points:
  - **CNN consists of:**
    - Convolutional layer
    - Pooling layer
    - Fully connected layer
  - **CNN learns the convolutional layer's kernel.**
    - To make computation easier, CNN uses cross-correlation instead of convolution.



https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

Convolutional Neural Network (CNN)

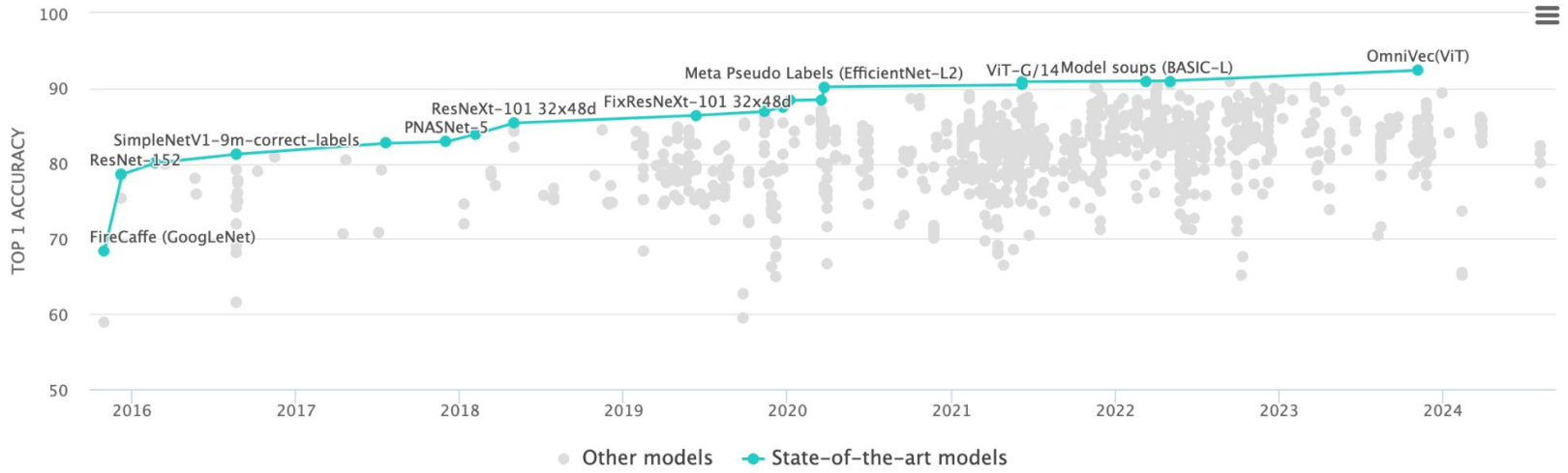
ImageNet

- ImageNet is a **large-scale** visual database designed for use in visual object recognition software research.
- It contains over **14 million images** that have been manually annotated to indicate the objects they depict.
- Organized based on the WordNet hierarchy, a lexical database of English.
- Synsets (synonym sets): Groupings of words with the same or similar meaning.
- Developed by Fei-Fei Li and her team at Stanford University in 2009.

Convolutional Neural Network (CNN)

ImageNet

- **Image Classification on ImageNet**

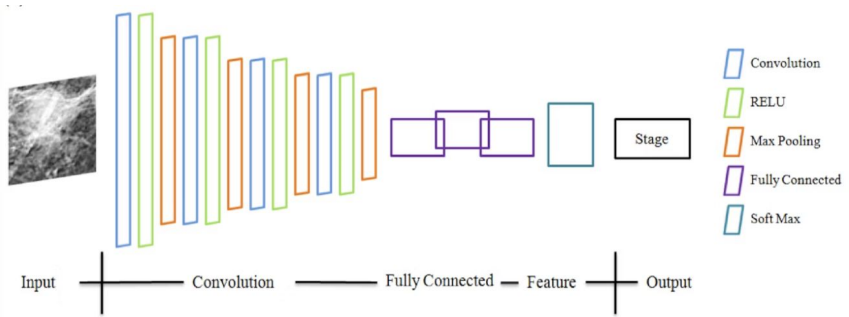


https://paperswithcode.com/sota/image-classification-on-imagenet

Convolutional Neural Network (CNN)

CNN Architecture Overview

- **Image: Diagram of CNN layers**
  - CNNs consist of several **layer types**:
    - **Convolutional Layer**: Extracts features using filters/kernels.
    - **Pooling Layer**: Reduces the spatial dimensions of the feature maps.
    - **Fully Connected Layer**: Makes final predictions by connecting all neurons.
    - **Activation Functions**: Typically use **ReLU** to introduce non-linearity.



https://www.auntminnie.com/imaging-informatics/artificial-intelligence/article/15668616/deeplearning-model-determines-breast-tumor-staging

Convolutional Neural Network (CNN)

Convolutional Layer

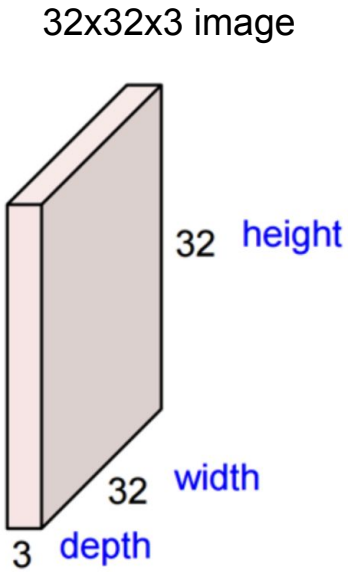
- **Image: Convolution process, showcasing filter moving across the image**
    - CNNs consist of several layer types:
        - **Convolution Operation:** Applies filters (small matrices) over the input image to detect features like edges, textures, and shapes.
        - Filters scan the image in sections (strides) and output **feature maps**.
        - Helps detect **low-level feature**s (e.g., edges) in early layers and **high-level features** (e.g., objects) in later layers.

https://www.auntminnie.com/imaging-informatics/artificial-intelligence/article/15668616/deeplearning-model-determines-breast-tumor-staging

Convolutional Neural Network (CNN)

Convolution Layer

- **Example**

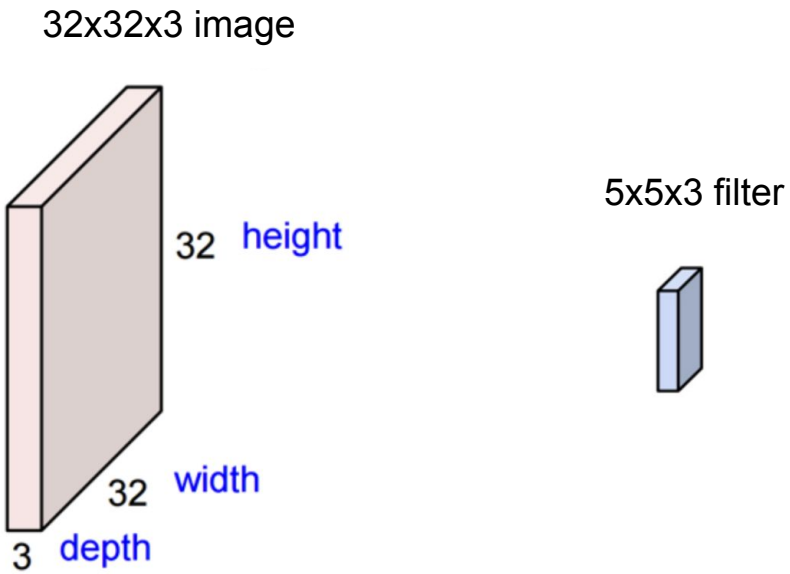


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Example**



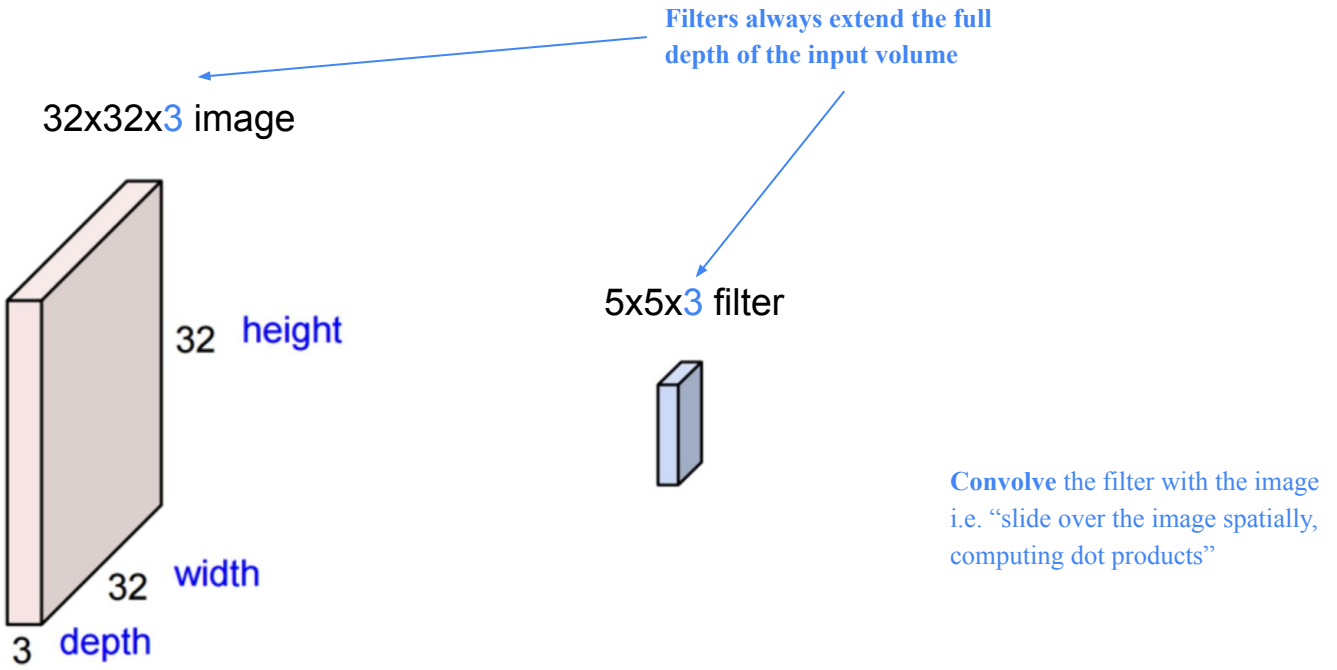
Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Example**

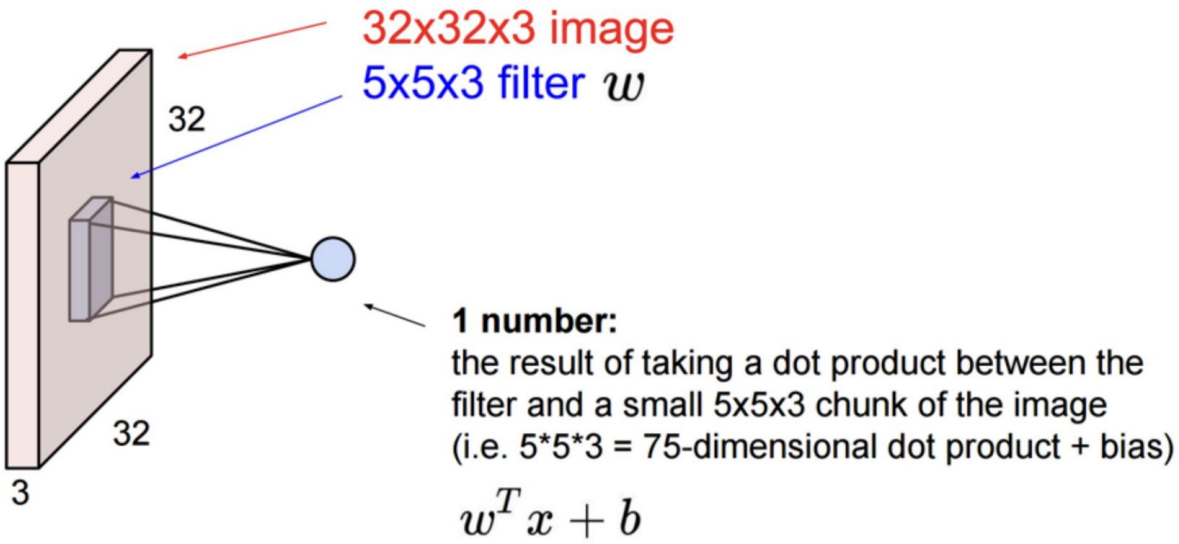


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Example**

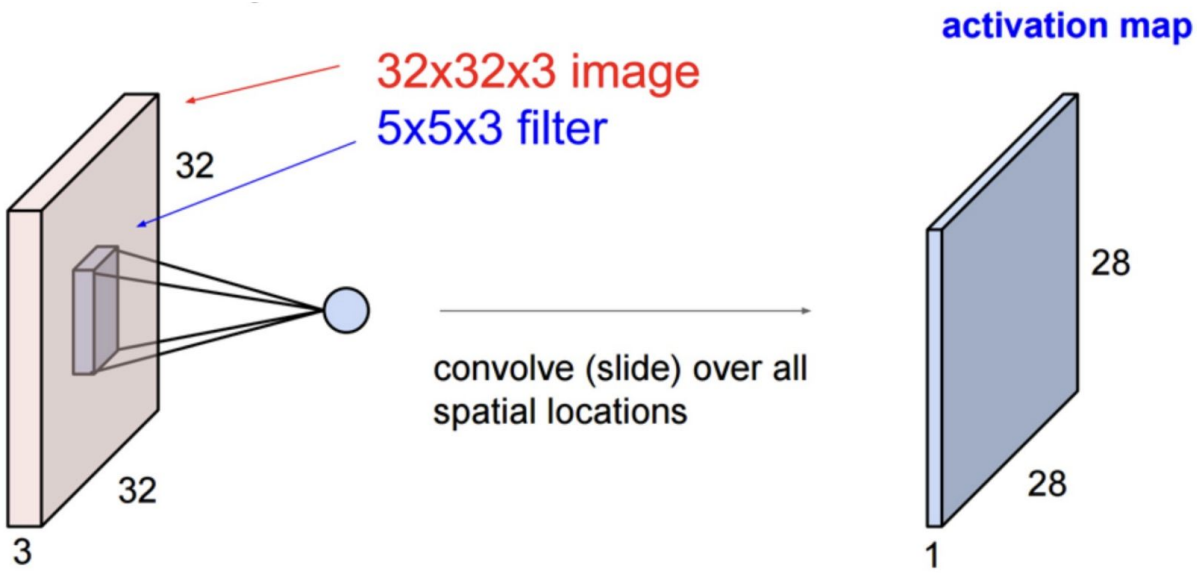


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Example**



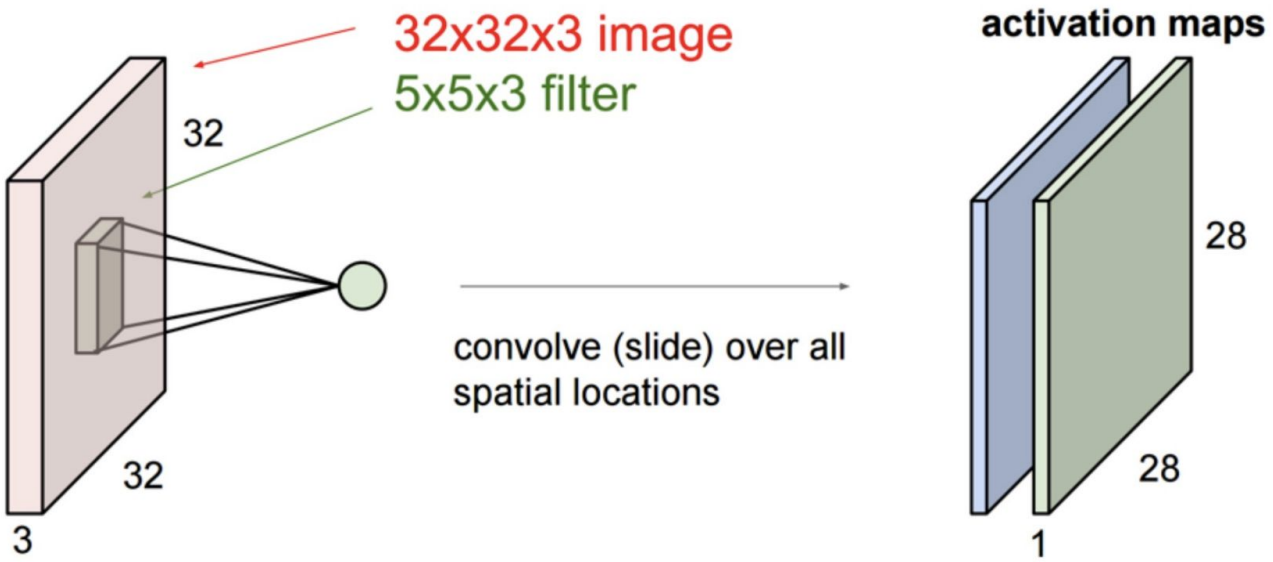
*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Example**

Consider a second, green filter



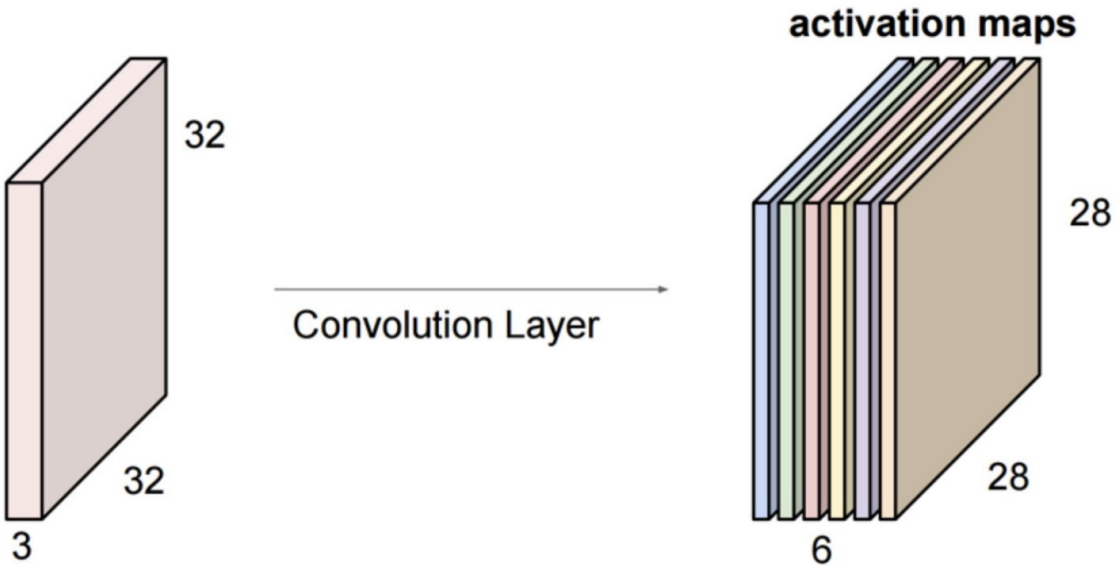
*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Example**

If we had 6 5x5 filters, we will get 6 separate activation maps



We stack these up to get a "new image" of size 6x28x28

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Demos**

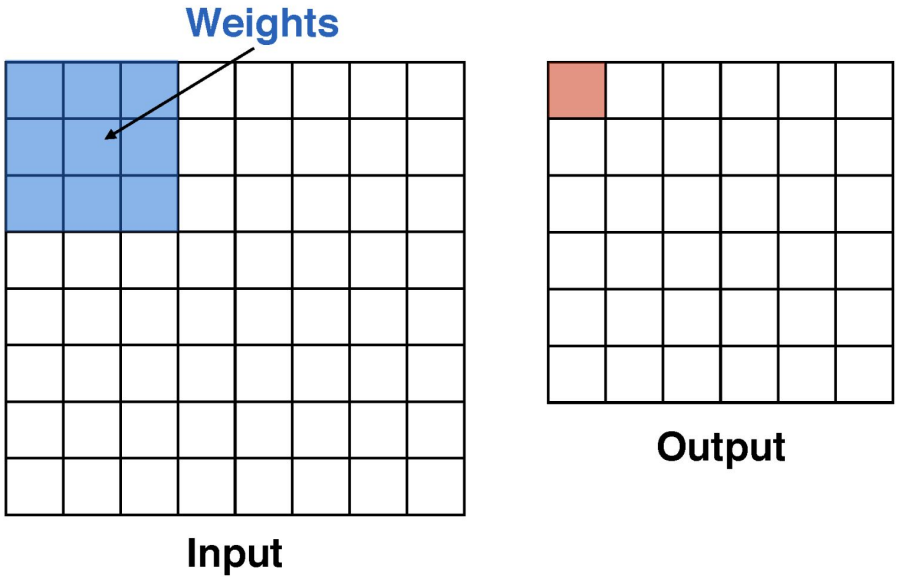
  http://cs231n.stanford.edu/

  http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**



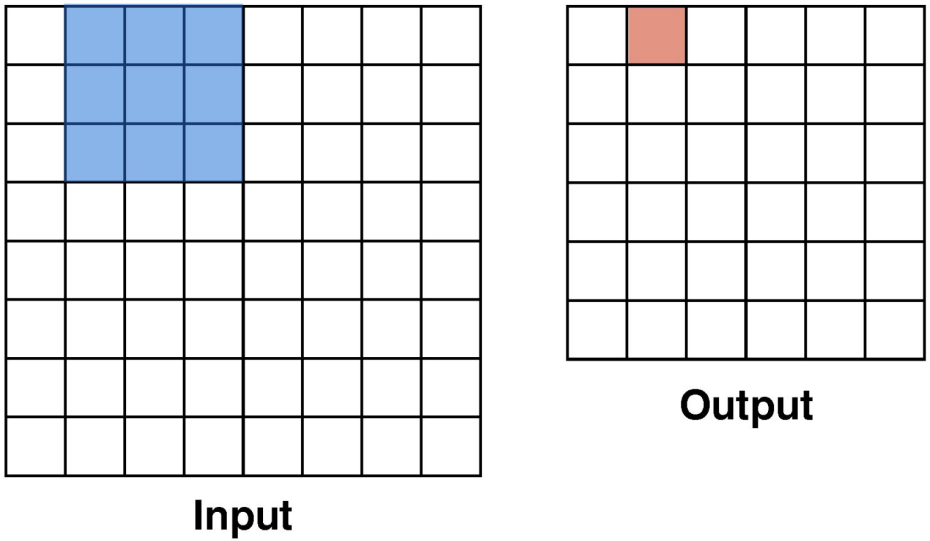
During convolution, the weights "slide" along the input to generate each output

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**



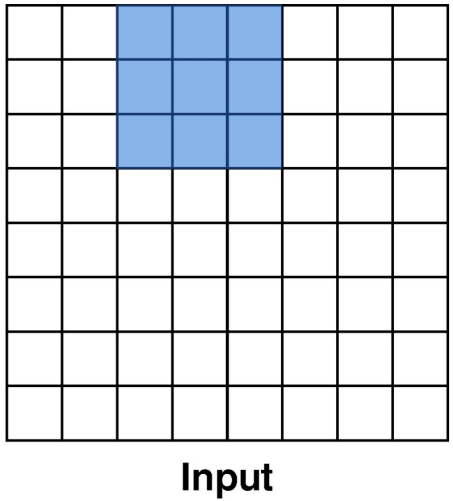
During convolution, the weights "slide" along the input to generate each output

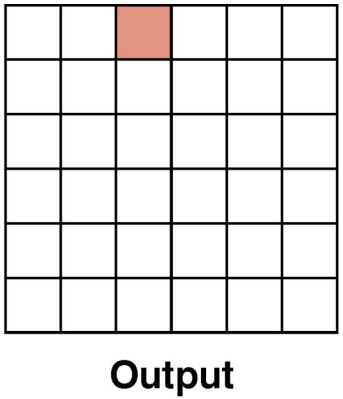
*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**





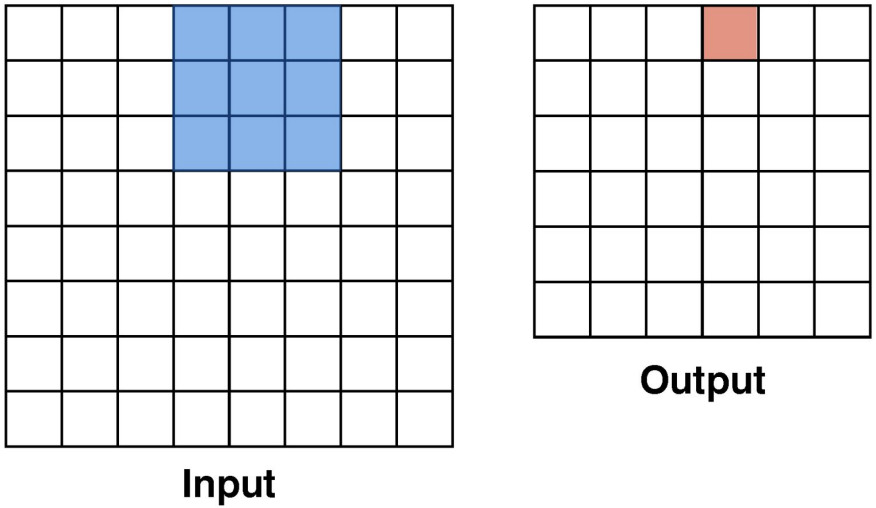
During convolution, the weights "slide" along the input to generate each output

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**



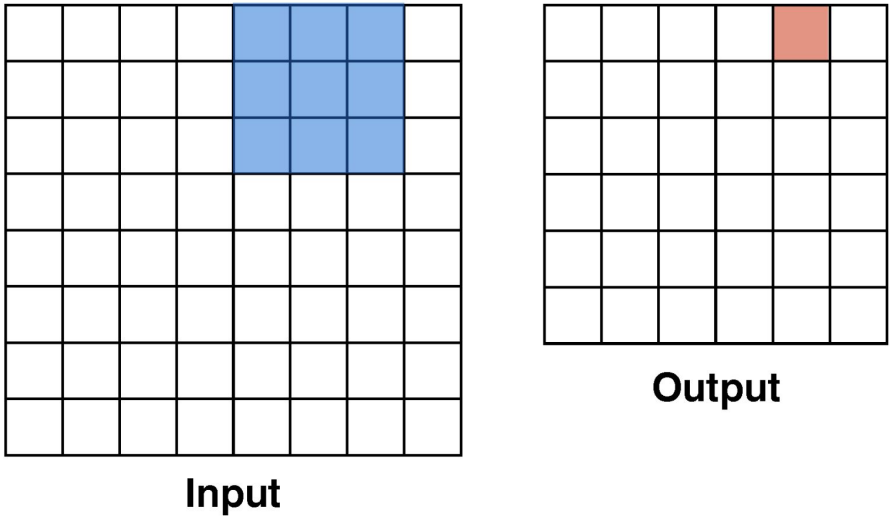
During convolution, the weights "slide" along the input to generate each output

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**



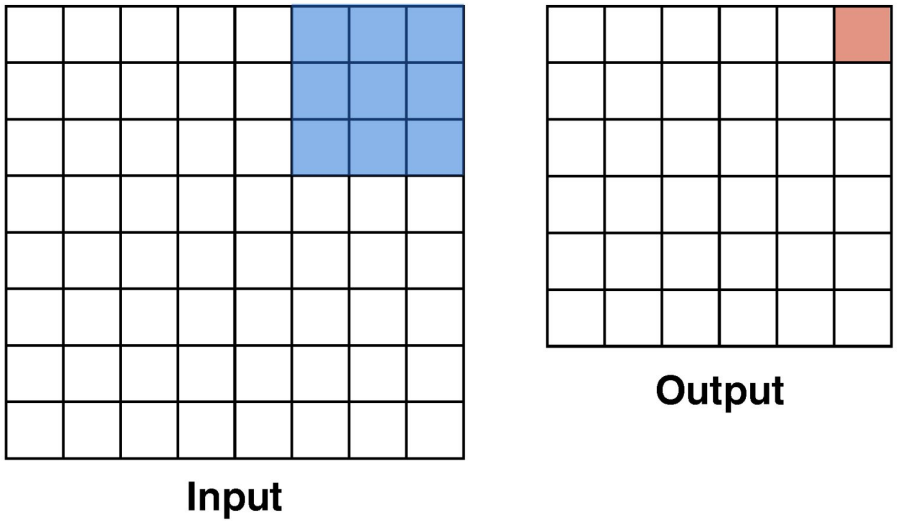
During convolution, the weights "slide" along the input to generate each output

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**



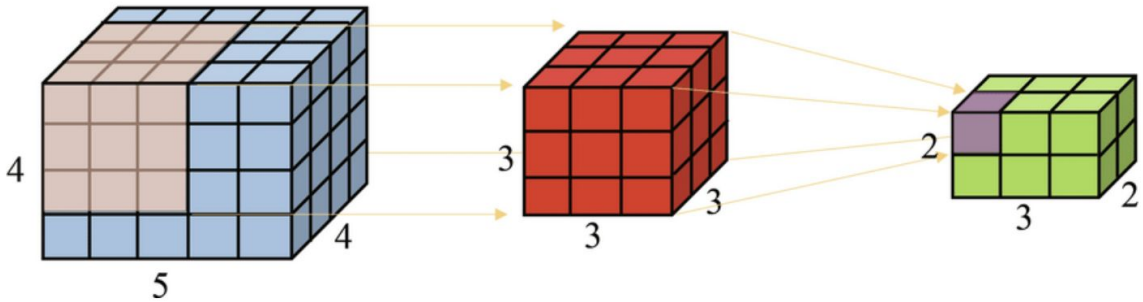
During convolution, the weights "slide" along the input to generate each output

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**
  - **Example of a 3D convolution operation with a stride of 1**



Note: each position we are doing a **3D** sum: channel, row, column

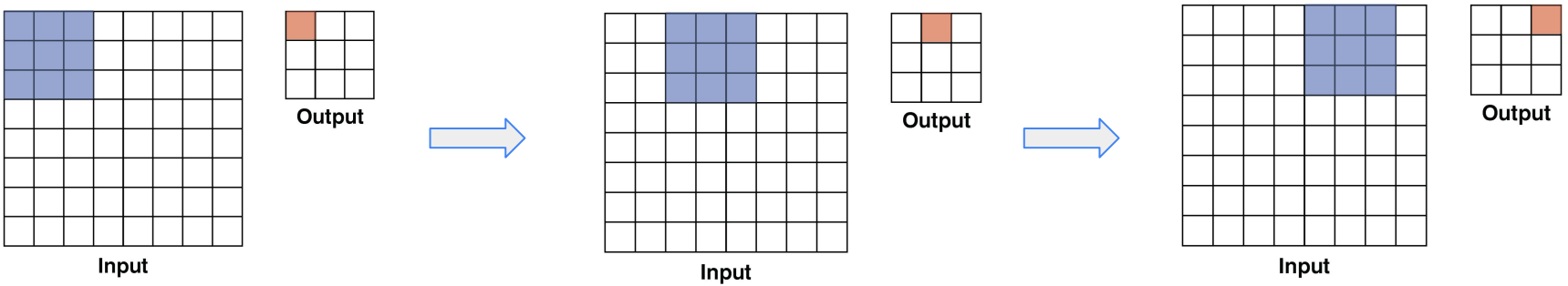
$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

Accelerated topology optimization design of 3D structures based on deep learning - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Example-of-a-3D-convolution-operation-with-a-stride-of-1_fig1_358908601 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**
  - **Example: stride = 2**

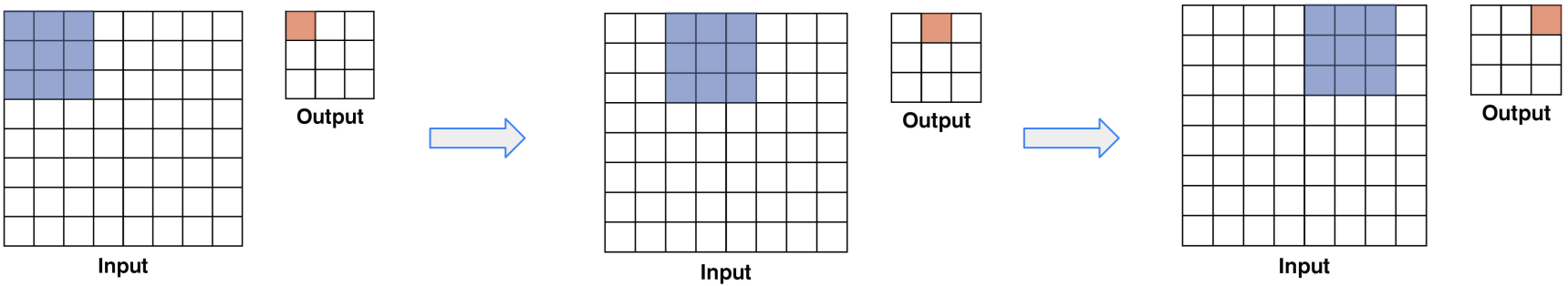


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Stride**
  - **Example: stride = 2**



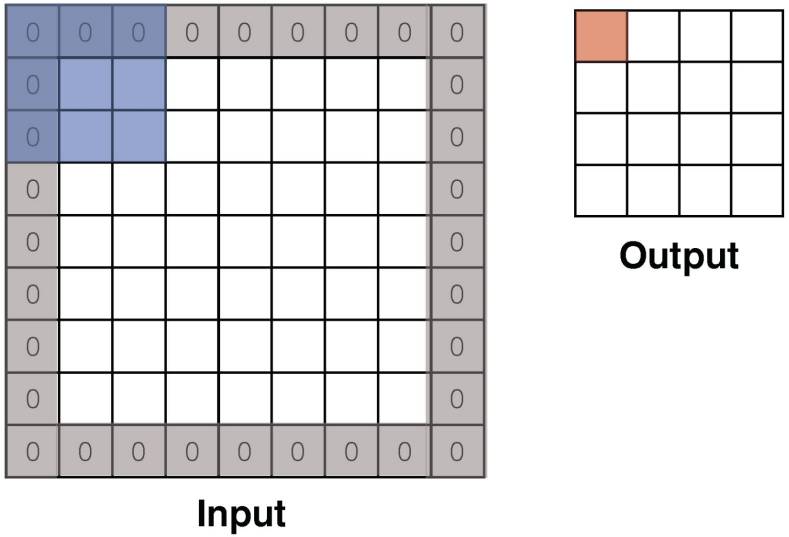
Notice that with certain strides, we may not be able to all the input.
The output is also half the size of the input.

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Padding**
  - **We can also pad the input with zeros.**
  - **Example, pad = 1, stride = 2.**

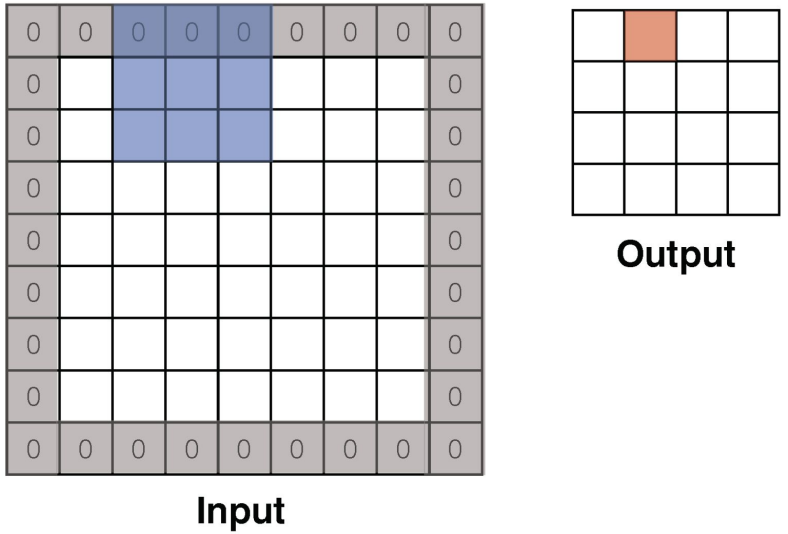


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Padding**
  - **We can also pad the input with zeros.**
  - **Example, pad = 1, stride = 2.**

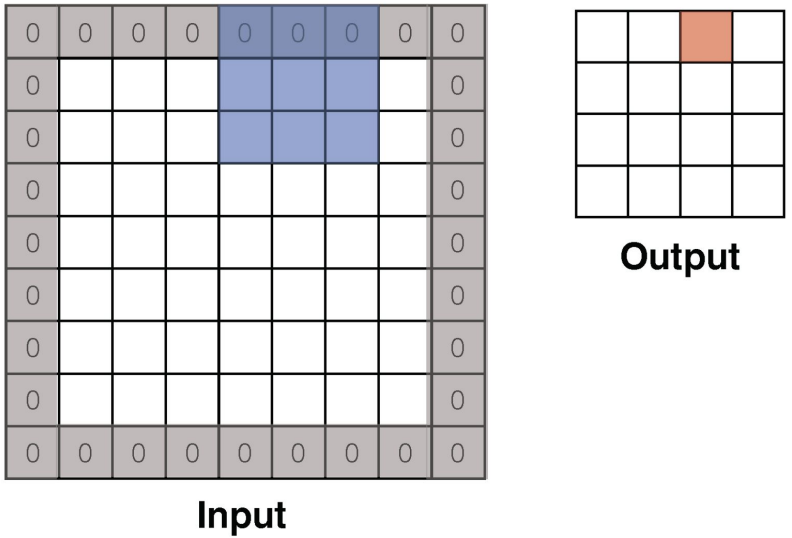


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Padding**
  - **We can also pad the input with zeros.**
  - **Example, pad = 1, stride = 2.**

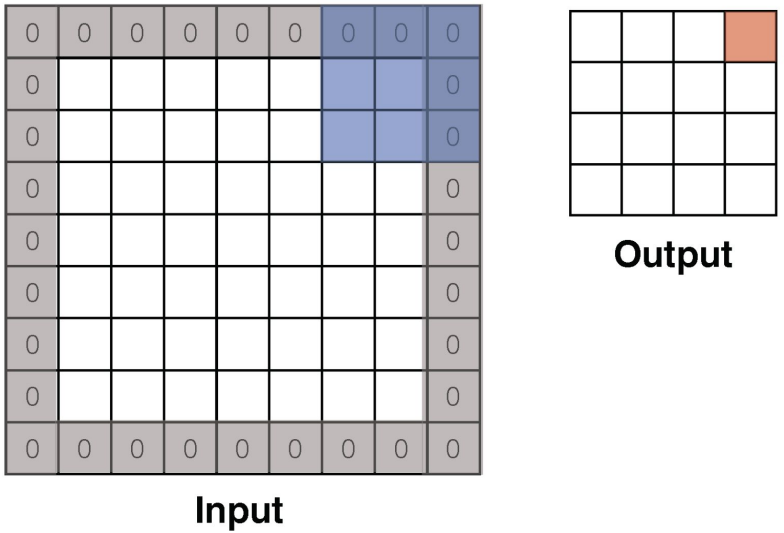


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Padding**
  - **We can also pad the input with zeros.**
  - **Example, pad = 1, stride = 2.**

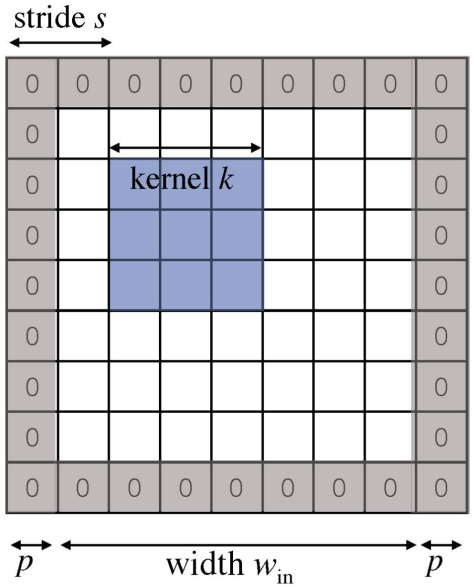


*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Q: How big is the output?**



In general, the output has size:

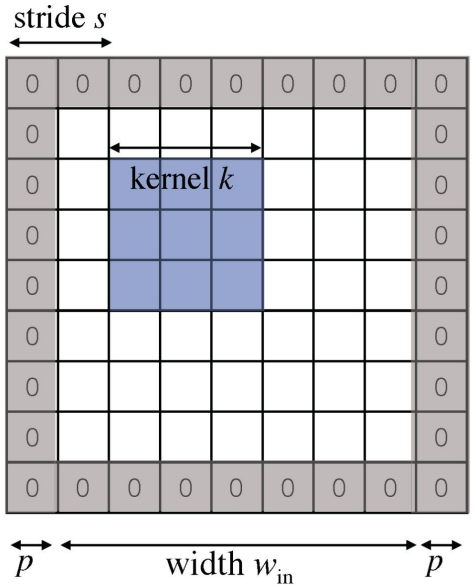
$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Convolution Layer

- **Q: How big is the output?**



Example: k = 3, s= 1, p = 1

$$w_{out} = \left\lfloor \frac{w_{in} + 2p - k}{s} \right\rfloor + 1$$
$$= \left\lfloor \frac{w_{in} + 2 - 3}{1} \right\rfloor + 1$$
$$= w_{in}$$

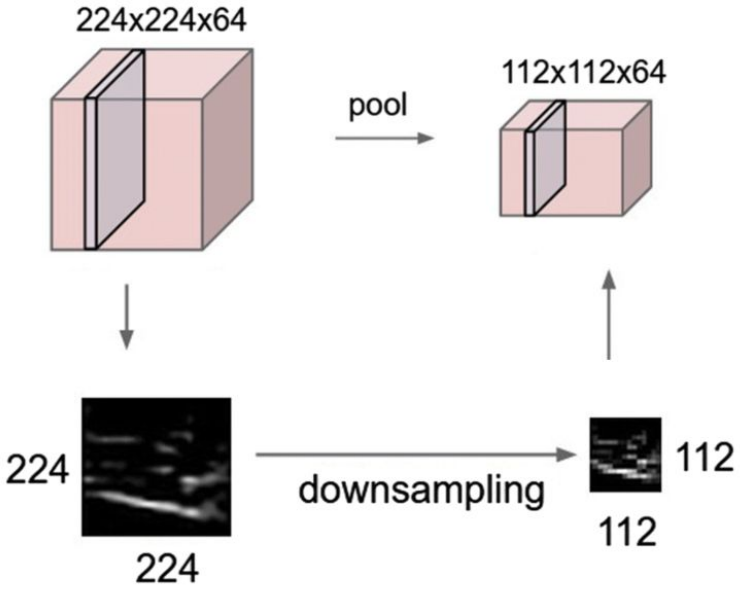
VGGNet [Simonyan 2014] used filters of this shape

*Carnegie Mellon University. (2022). *Convolutional neural networks (CNNs)* [Lecture slides]. http://16385.courses.cs.cmu.edu/fall2022content/lectures/16_cnn/16_cnn_slides.pdf*

Convolutional Neural Network (CNN)

Pooling Layer

- For most ConvNets, convolution is often followed by **pooling**
  - Creates a smaller representation while retaining the most important information
  - The **“Max”** operation is the most common
    - Max-pooling is a **downsampling** operation often used in between convolution layers to reduce the spatial dimensions of the feature maps.
    - It helps in simplifying the network and extracting important features.



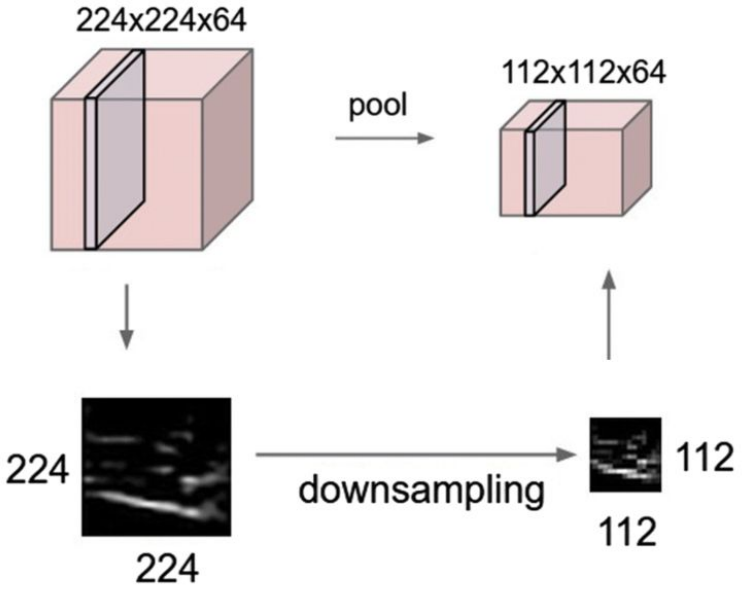
Q: Why might “Avg” be a poor choice?

Interpretation of intelligence in CNN-pooling processes: a methodological survey - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Down-sampling-using-pooling-5_fig1_334208543 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

Pooling Layer

- **Q: Why might “Avg” be a poor choice?**
  - Makes the representations smaller and more manageable
  - Operates over each activation map independently

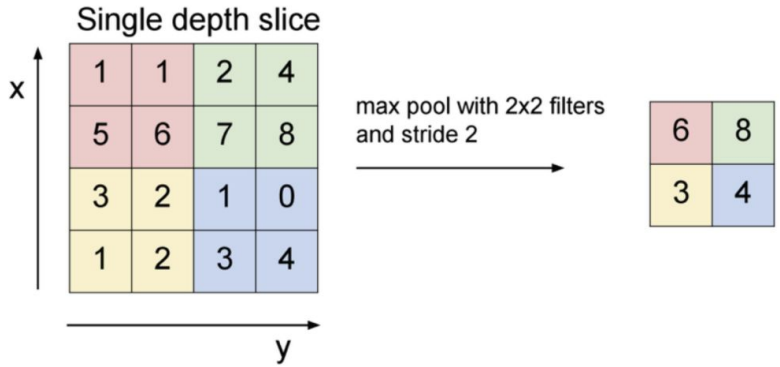


Interpretation of intelligence in CNN-pooling processes: a methodological survey - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Down-sampling-using-pooling-5_fig1_334208543 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

Pooling Layer

- Max Pooling
  - What's the backprop rule for max pooling?
    - In the forward pass, store the index that took the max
    - The backprop gradient is the input gradient at the index

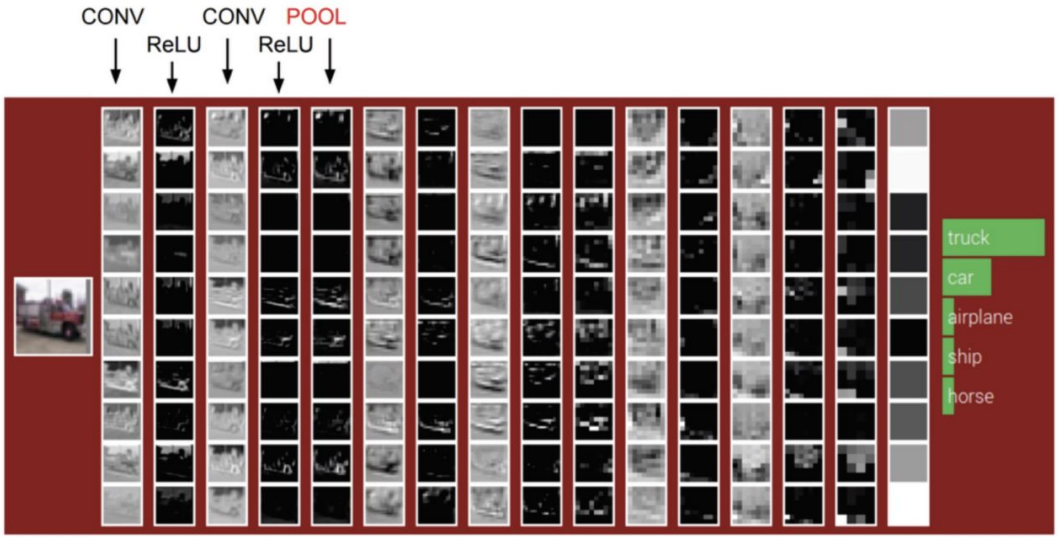


Interpretation of intelligence in CNN-pooling processes: a methodological survey - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Down-sampling-using-pooling-5_fig1_334208543 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

Pooling Layer

- Example ConvNet

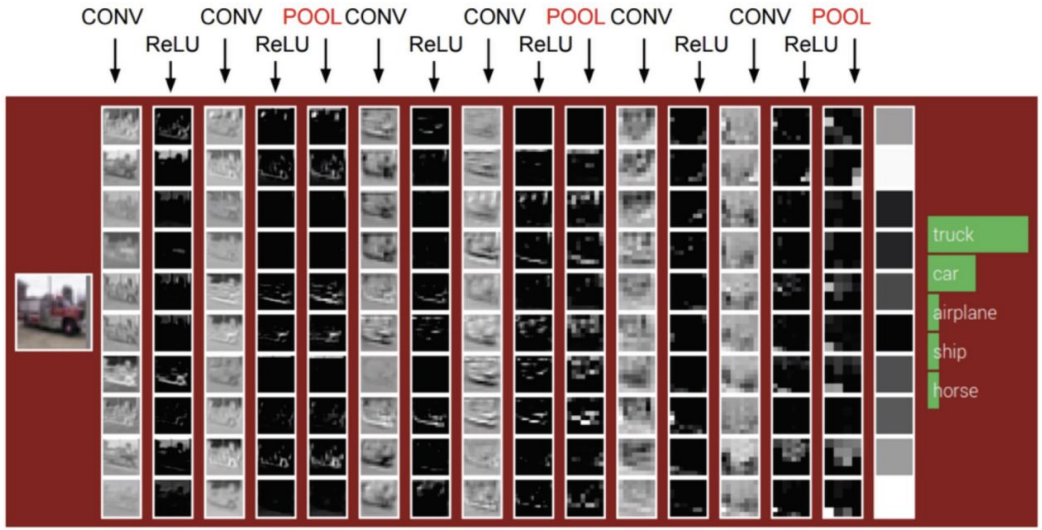


Interpretation of intelligence in CNN-pooling processes: a methodological survey - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Down-sampling-using-pooling-5_fig1_334208543 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

Pooling Layer

- Example ConvNet

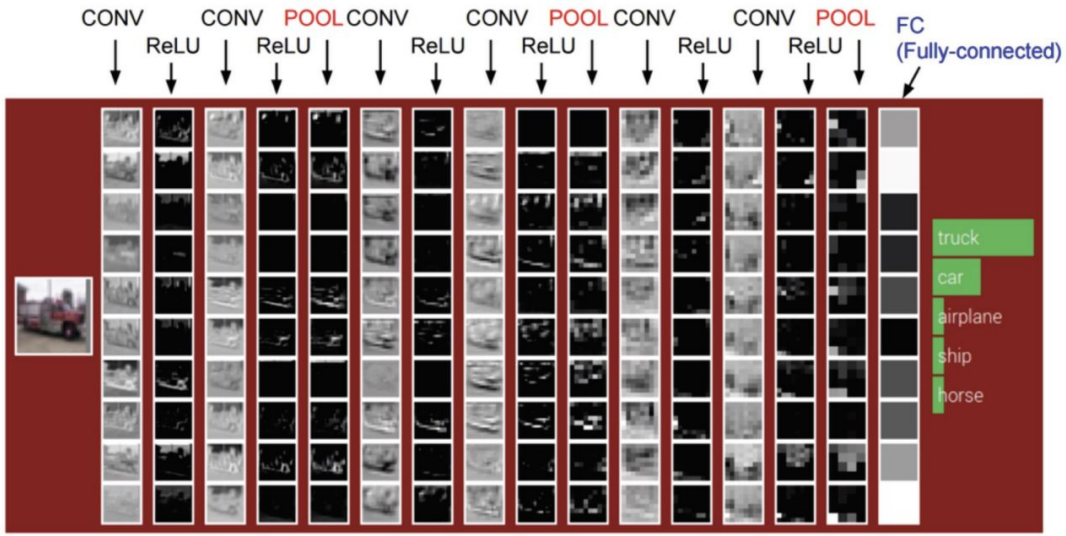


Interpretation of intelligence in CNN-pooling processes: a methodological survey - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Down-sampling-using-pooling-5_fig1_334208543 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

Pooling Layer

- Example ConvNet

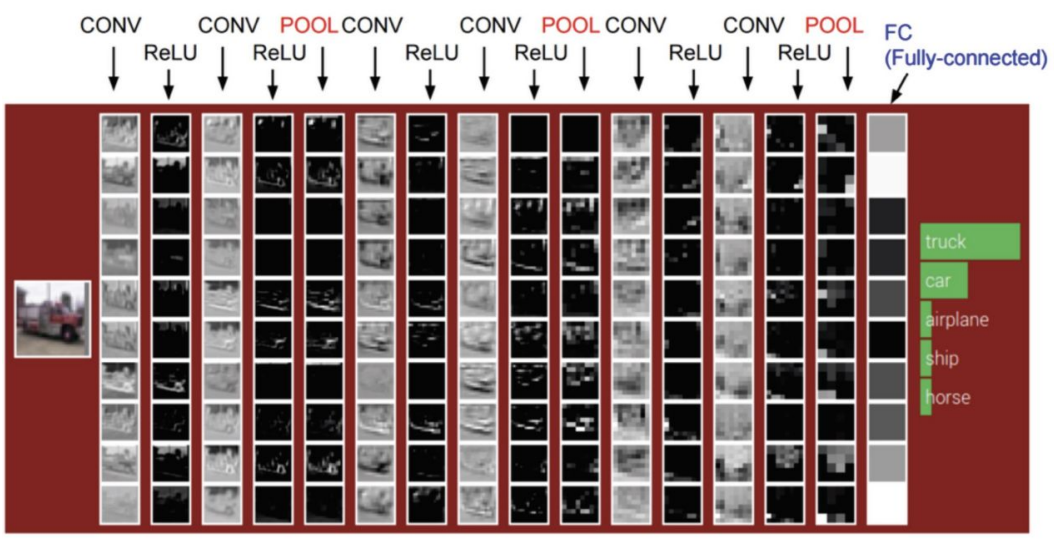


Interpretation of intelligence in CNN-pooling processes: a methodological survey - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Down-sampling-using-pooling-5_fig1_334208543 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

Pooling Layer

- Example ConvNet

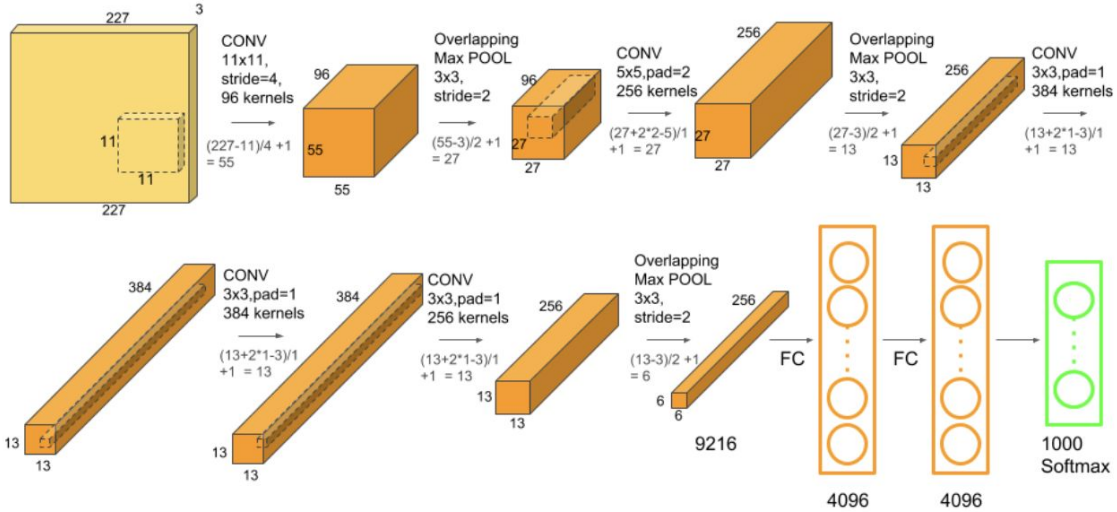


10x3x3 conv filters,
stride=1, pad=1
2x2 pool filters,
stride=2

Interpretation of intelligence in CNN-pooling processes: a methodological survey - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Down-sampling-using-pooling-5_fig1_334208543 [accessed 16 Sept 2024]

Convolutional Neural Network (CNN)

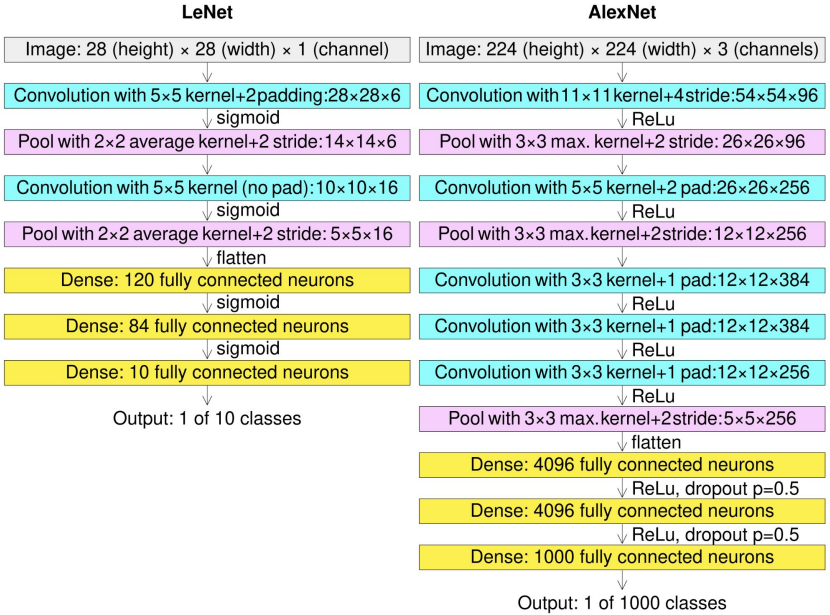
Example: AlexNet



AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers

Convolutional Neural Network (CNN)

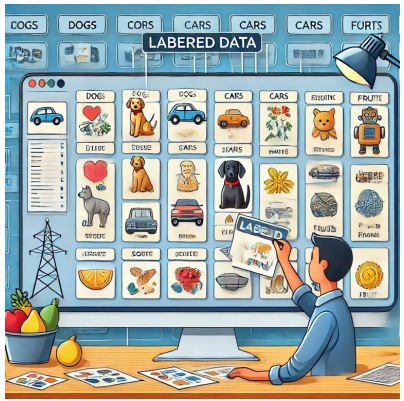
Example: Comparison of the LeNet and AlexNet convolution, pooling, and dense layers

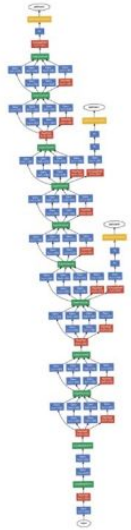


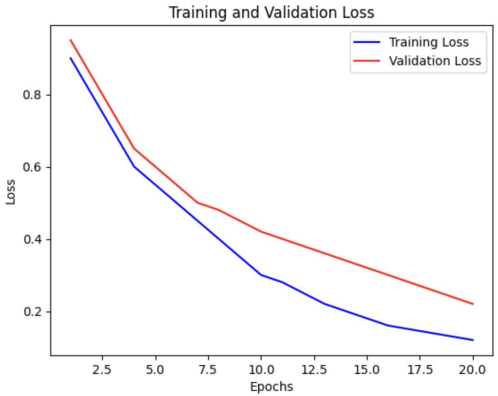
https://en.wikipedia.org/wiki/AlexNet#/media/File:Comparison_image_neural_networks.svg

Training ConvNets

Q: How do you actually train these things?







Gather labeled data

Build a ConvNet architecture

Minimize the loss

Training ConvNets

- **Training a convolutional neural network**
  - **Loading the Data Set**
  - **Normalization, Reshape and Label Encoding**
  - **Train Test Split**
  - **Convolutional Neural Network**
    - Convolution Operation
    - Same Padding
    - Max Pooling
    - Flattening
    - Full Connection

Training ConvNets

- **Training a convolutional neural network**
  - **Loading the Data Set**

```python
# read train
train = pd.read_csv("/Users/xiaotingzhou/Documents/GitHub/AI_Civil/Dataset/Digit Recognizer/train.csv")
print(train.shape)
train.head()
```

Python

```
(42000, 785)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel782	pixel783
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

Training ConvNets

- **Training a convolutional neural network**
  - **Normalization, Reshape and Label Encoding**

```
# Normalize the data
X_train = X_train / 255.0
test = test / 255.0
print("x_train shape: ",X_train.shape)
print("test shape: ",test.shape)
```

```
x_train shape:  (42000, 28, 28, 1)
test shape:  (28000, 28, 28, 1)
```

```
# Reshape
X_train = X_train.reshape(-1,28,28,1)
test = test.reshape(-1,28,28,1)
print("x_train shape: ",X_train.shape)
print("test shape: ",test.shape)
```

```
x_train shape:  (42000, 28, 28, 1)
test shape:  (28000, 28, 28, 1)
```

```
# Label Encoding
# from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
from tensorflow.keras.utils import to_categorical

Y_train = to_categorical(Y_train, num_classes = 10)
```

Training ConvNets

- **Training a convolutional neural network**
  - **Train Test Split**

We split the data into train and test sets.

- test size is 10%.
- train size is 90%.

```
# Split the train and the validation set for the fitting
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size = 0.1, random_state=2)
print("x_train shape",X_train.shape)
print("x_test shape",X_val.shape)
print("y_train shape",Y_train.shape)
print("y_test shape",Y_val.shape)
```

```
x_train shape (37800, 28, 28, 1)
x_test shape (4200, 28, 28, 1)
y_train shape (37800, 10, 10)
y_test shape (4200, 10, 10)
```

Training ConvNets

- **Training a convolutional neural network**
  - **Convolutional Neural Network**
    - Convolution Operation
    - Same Padding
    - Max Pooling
    - Flattening
    - Full Connection

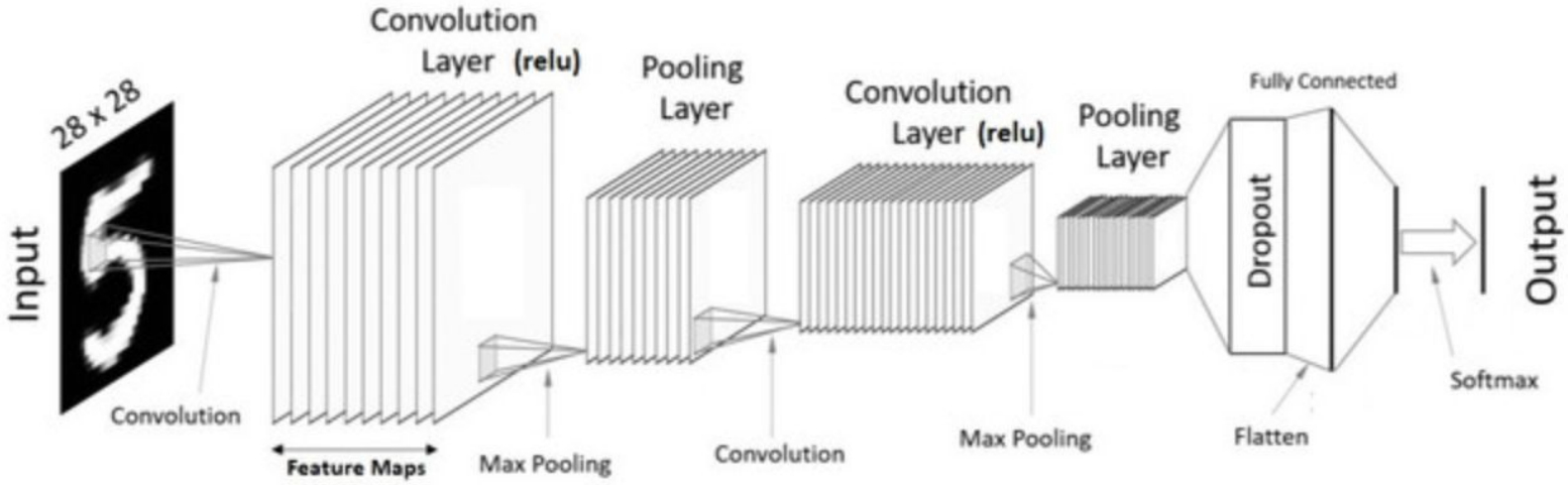
Create Model

- conv => max pool => dropout =>
  conv => max pool => dropout =>
  fully connected (2 layer)
- **Dropout**: Dropout is a technique where randomly selected neurons are ignored during training

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D, Reshape
from tensorflow.keras.optimizers import RMSprop, Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras import backend as K

# Define custom loss function if necessary (e.g., categorical crossentropy for multi-class segmentation)
def custom_loss_function(y_true, y_pred):
    return K.categorical_crossentropy(y_true, y_pred)

# Model definition
model = Sequential()

model.add(Conv2D(filters=8, kernel_size=(5,5), padding='Same', activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=16, kernel_size=(3,3), padding='Same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))

# Fully connected
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))

# Change to Dense to output a (10, 10) matrix, then reshape to (10, 10, 1)
model.add(Dense(10*10, activation='softmax'))  # Flattening into 100 units
model.add(Reshape((10, 10, 1)))  # Reshape to (10, 10, 1)

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Print model summary to verify architecture
model.summary()
```

Python: CNN in Python

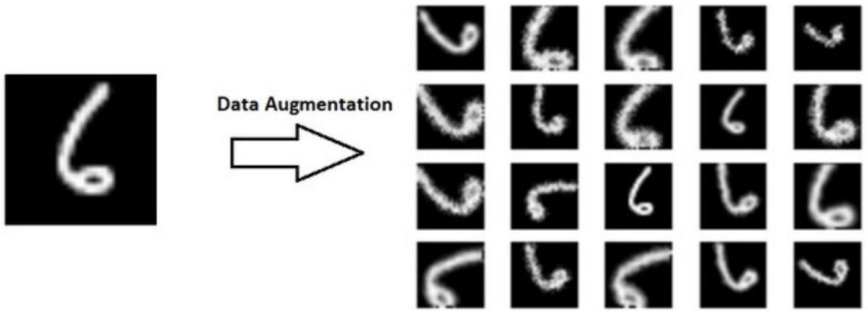
Example: Digit Recognizer



Python: CNN in Python

Example: Digit Recognizer

- **Implementing with Keras**
  - Create Model
  - Define Optimizer
  - Compile Model
  - Epochs and Batch Size
  - Data Augmentation
  - Fit the Model
  - Evaluate the Model



Python: CNN in Python

Example: Digit Recognizer

Create Model

- conv => max pool => dropout => conv => max pool => dropout => **fully connected (2 layer)**
- **Dropout**: Dropout is a technique where randomly selected neurons are ignored during training

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 28, 28, 8)	208
max_pooling2d_6 (MaxPooling2D)	(None, 14, 14, 8)	0
dropout_9 (Dropout)	(None, 14, 14, 8)	0
conv2d_7 (Conv2D)	(None, 14, 14, 16)	1,168
max_pooling2d_7 (MaxPooling2D)	(None, 7, 7, 16)	0
dropout_10 (Dropout)	(None, 7, 7, 16)	0
flatten_3 (Flatten)	(None, 784)	0
dense_6 (Dense)	(None, 256)	200,960
dropout_11 (Dropout)	(None, 256)	0
dense_7 (Dense)	(None, 100)	25,700
reshape (Reshape)	(None, 10, 10, 1)	0

Python: CNN in Python

Example: Digit Recognizer