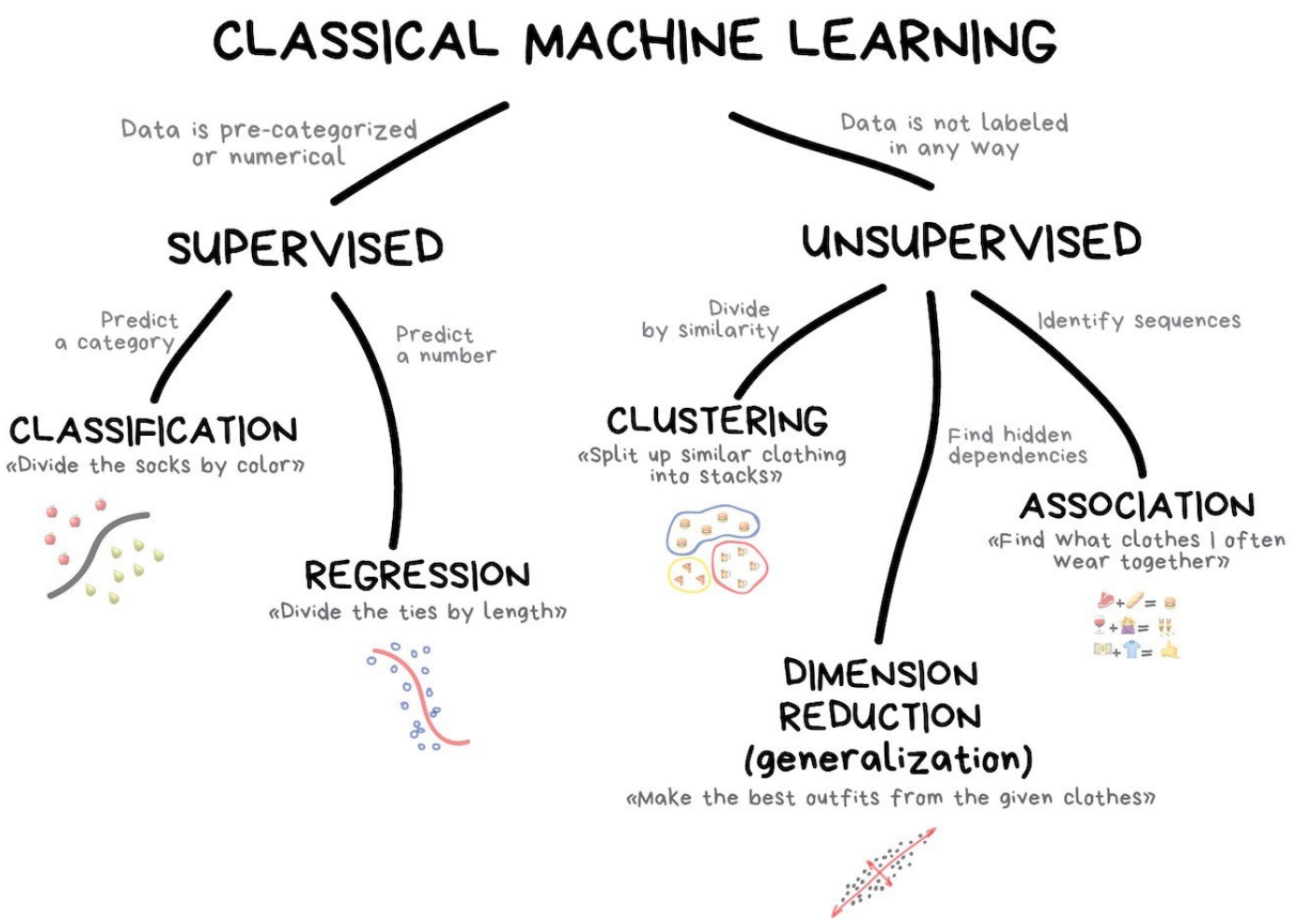
Capturing Spatial Dependencies with Deep Neural Networks I

Yao-Yi Chiang
Computer Science and Engineering
University of Minnesota
yaoyi@umn.edu

What have we covered so far?

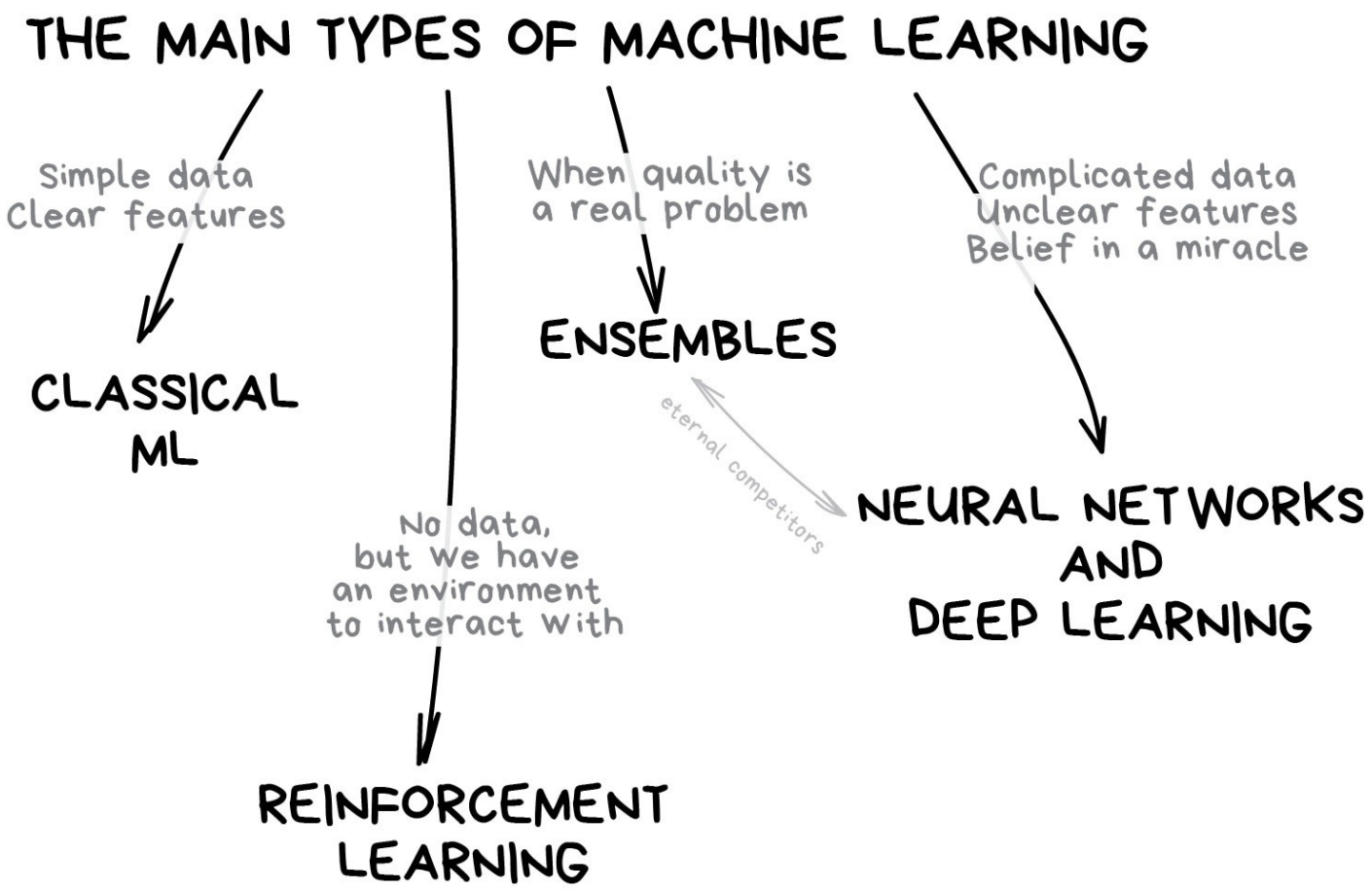
- Clustering
- Dimension Reduction
- Random Forest
  - Classification
  - Regression



https://vas3k.com/blog/machine_learning/

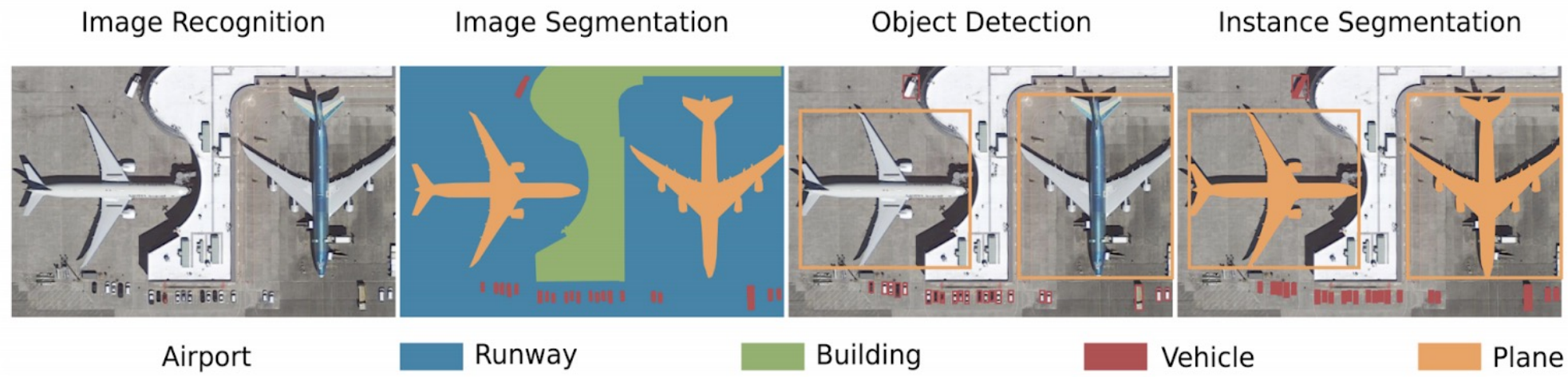
What have we covered so far?

- Classical ML
- Ensembles



https://vas3k.com/blog/machine_learning/

Overhead Imagery Understanding Tasks

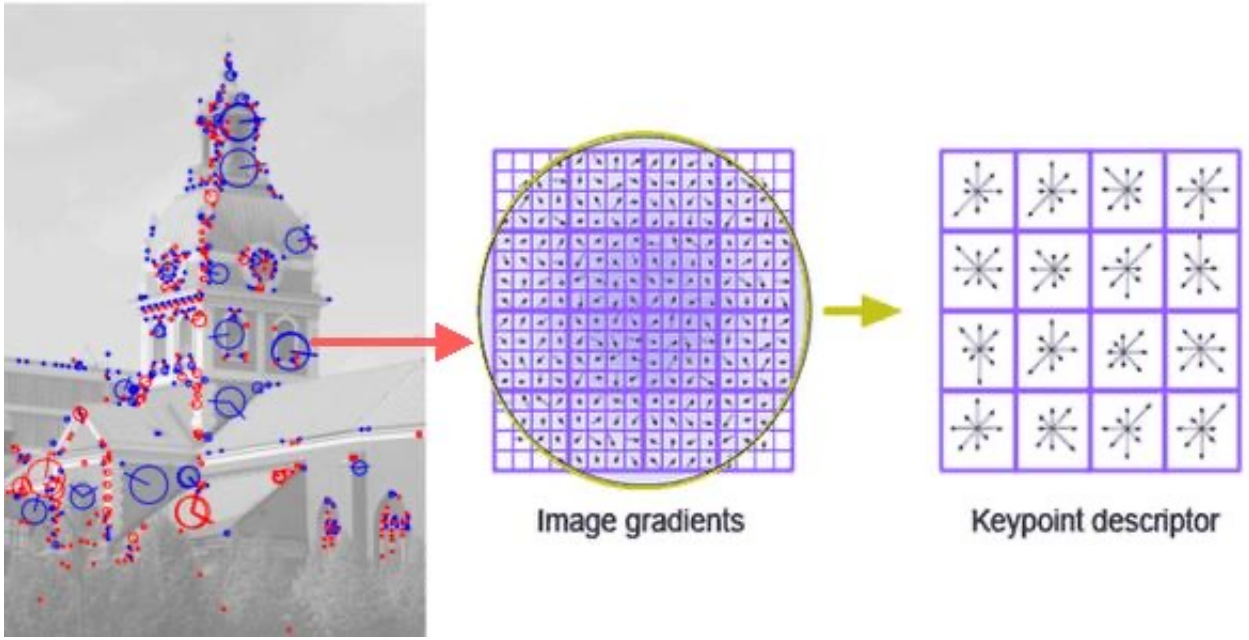


Thorsten Hoeser and Claudia Kuenzer. 2020. Object detection and image segmentation with deep learning on earth observation data: A review-part i: Evolution and recent trends. *Remote Sensing*. Retrieved from https://www.mdpi.com/723500

Object Detection & Recognition in 1999

SIFT (Scale Invariant Feature Transform)

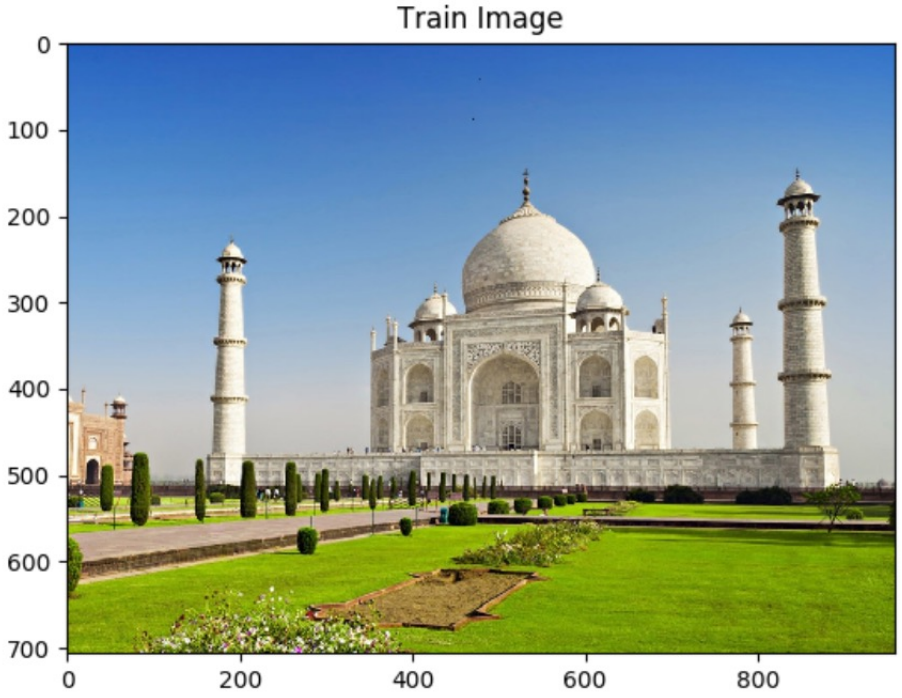
- SIFT is one of most popular image feature extraction and description algorithms
- SIFT extracts feature points and describe them with a scale, illumination, and rotational invariant descriptor

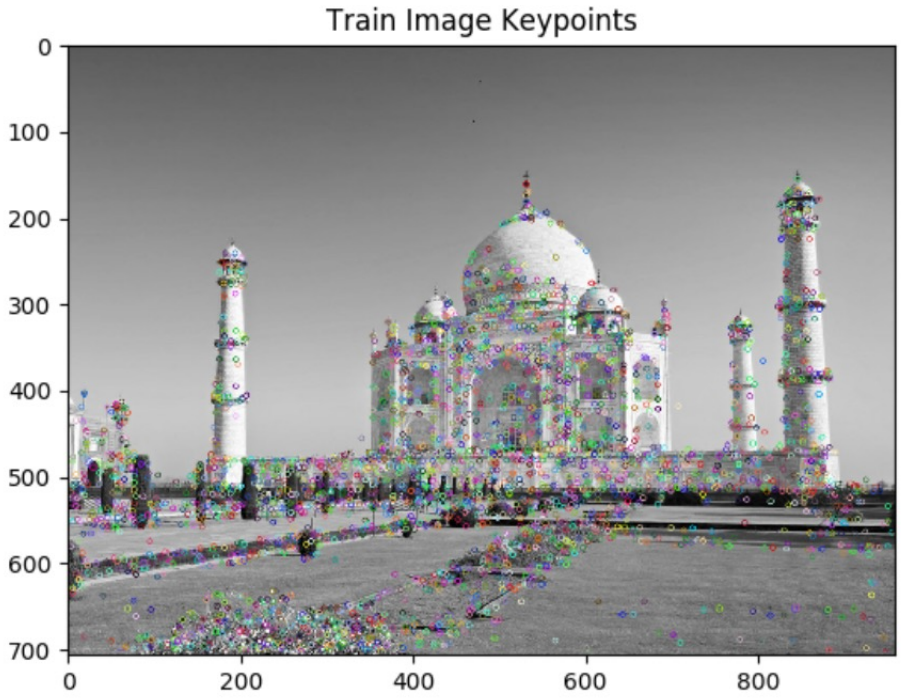


https://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT-Features-with-O

Object Detection & Recognition in 1999

SIFT (Scale Invariant Feature Transform)

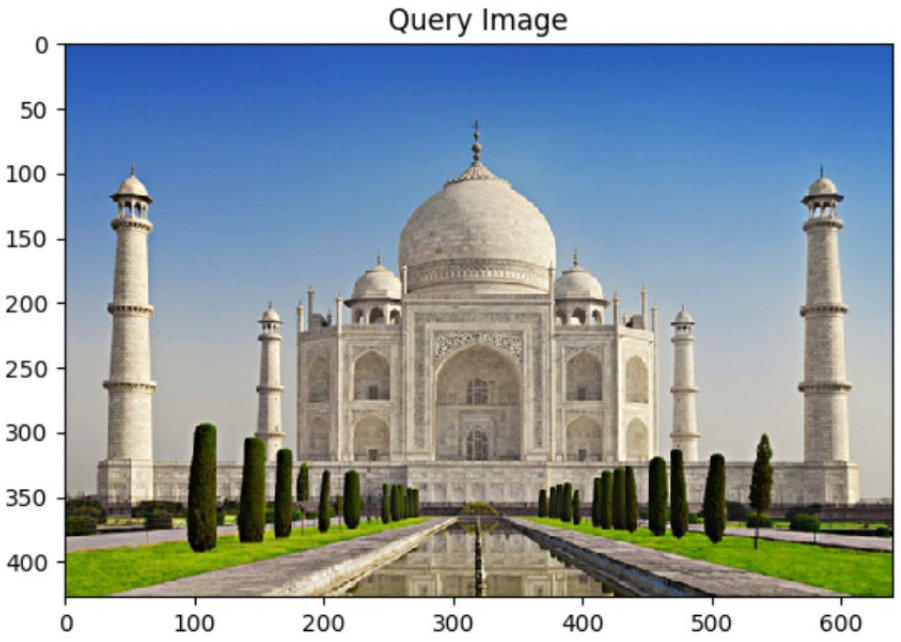


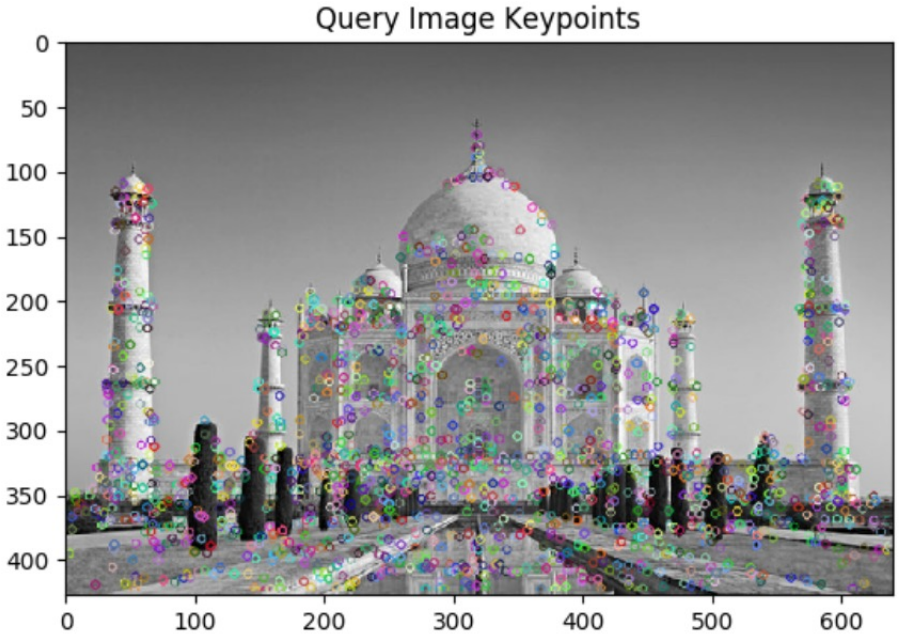


https://iq.opengenus.org/scale-invariant-feature-transform/

Object Detection & Recognition in 1999

SIFT (Scale Invariant Feature Transform)

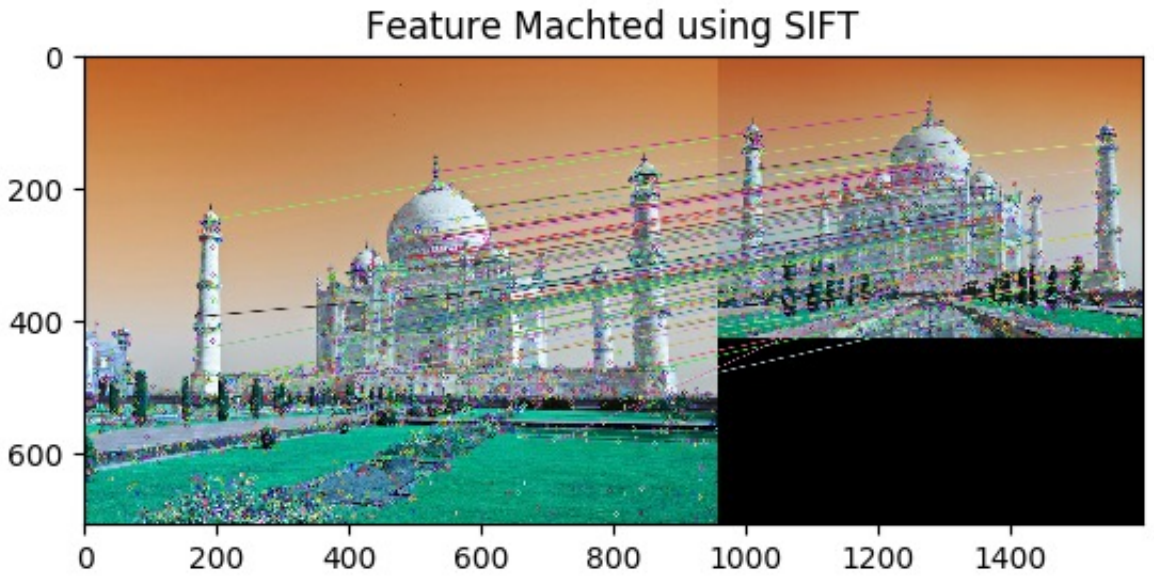




https://iq.opengenus.org/scale-invariant-feature-transform/

Object Detection & Recognition in 1999

SIFT (Scale Invariant Feature Transform)



https://iq.opengenus.org/scale-invariant-feature-transform/

Object Detection & Recognition in 1999

SIFT (Scale Invariant Feature Transform)

https://iq.opengenus.org/scale-invariant-feature-transform/

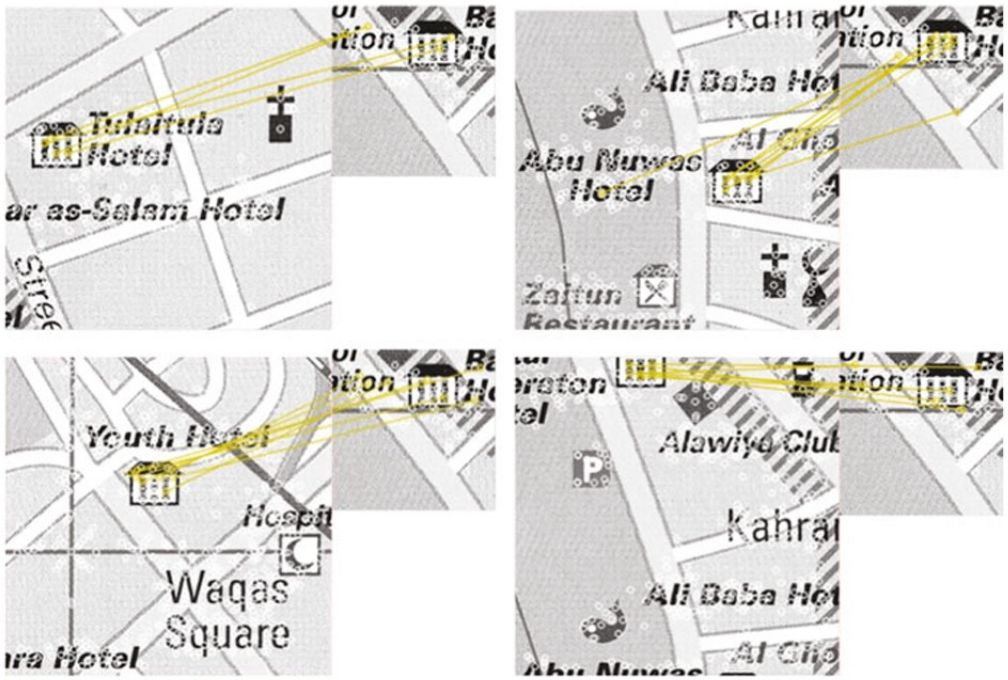
Object Detection & Recognition in 1999

SIFT (Scale Invariant Feature Transform)

https://iq.opengenus.org/scale-invariant-feature-transform/

Map Processing

- The small image on the right: hotel sample
- The small white circles on the maps: SURF descriptors
- The yellow lines connect matches between the SURF descriptors of the map area and the sample.



(a) Examples of correct matches

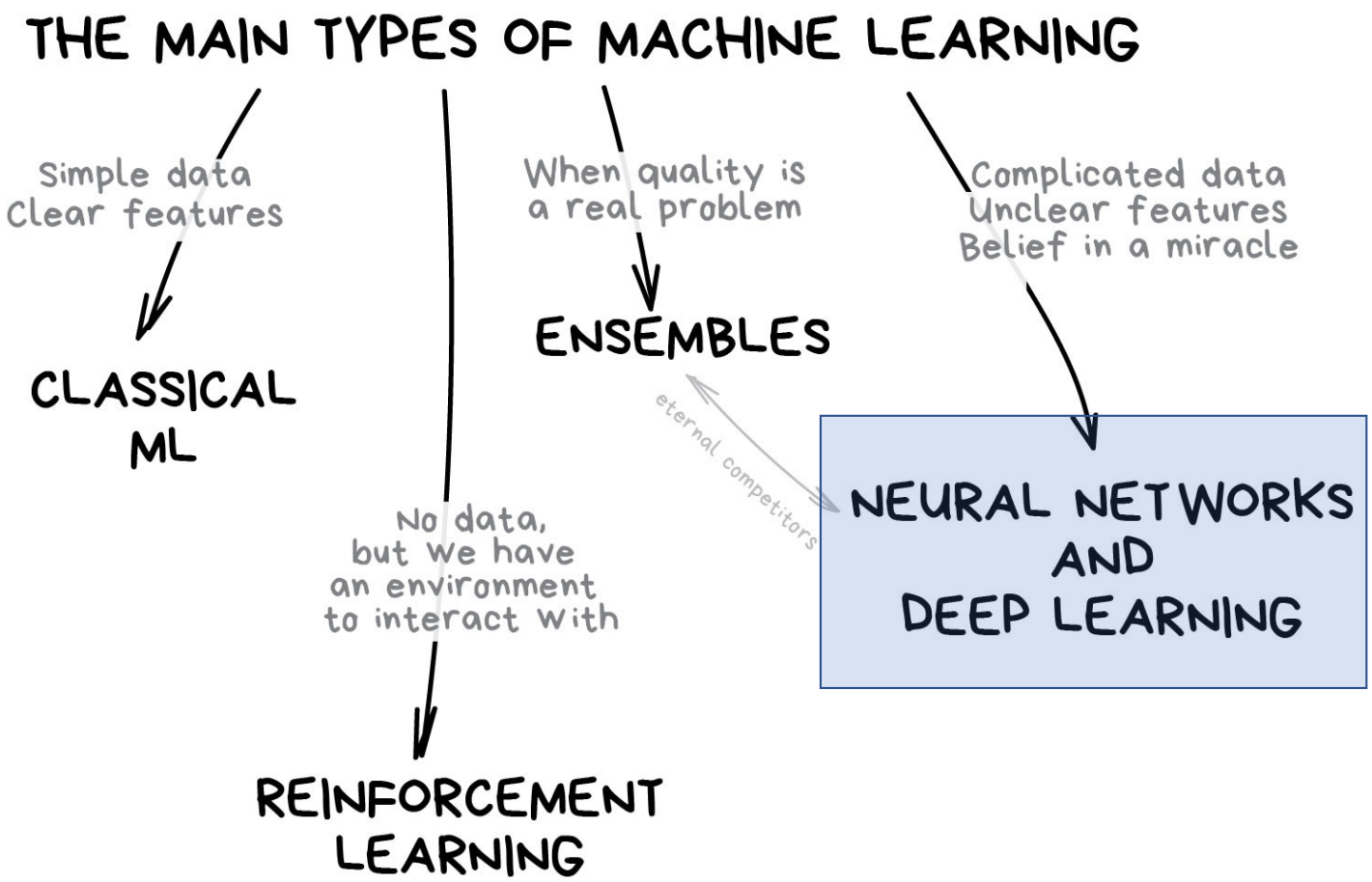
Using Historical Maps in Scientific Studies: Applications, Challenges, and Best Practices. Chiang, Y.; Duan, W.; Leyk, S.; Uhl, J. H; and Knoblock, C. A of SpringerBriefs in GeographySpringer International Publishing, 2020.

Handcrafted Image Features

- Pros
  - Simple, easy to implement
  - Do not require significant computational power
  - Somehow explainable results
- Cons
  - Not very robust (e.g., objects need to be very similar for a match)
  - Limited computer vision applications

What have we covered so far?

- Classical ML
- Ensembles



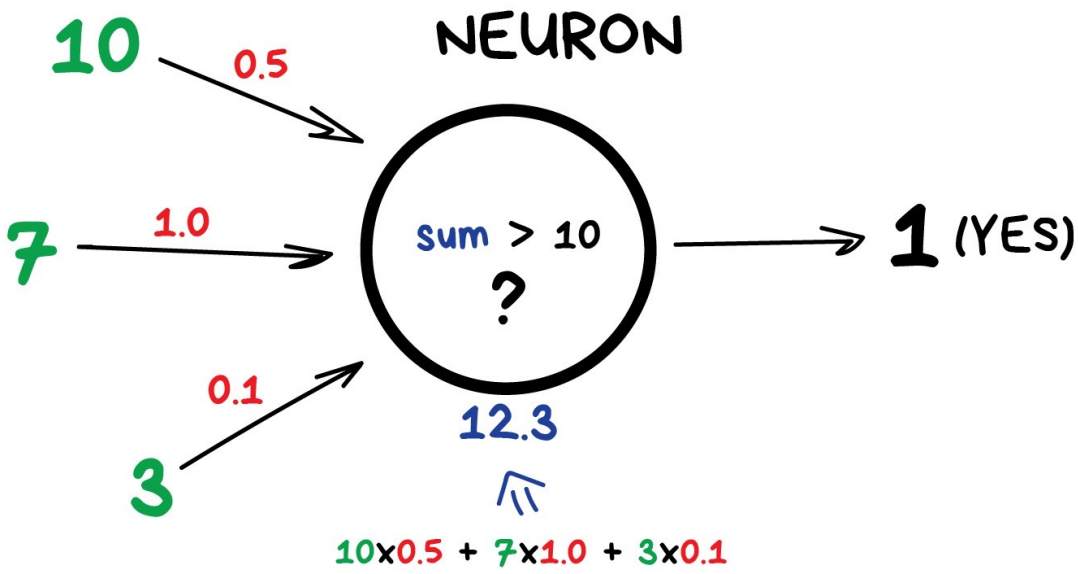
https://vas3k.com/blog/machine_learning/

Neural Networks and Deep Learning

A short introduction

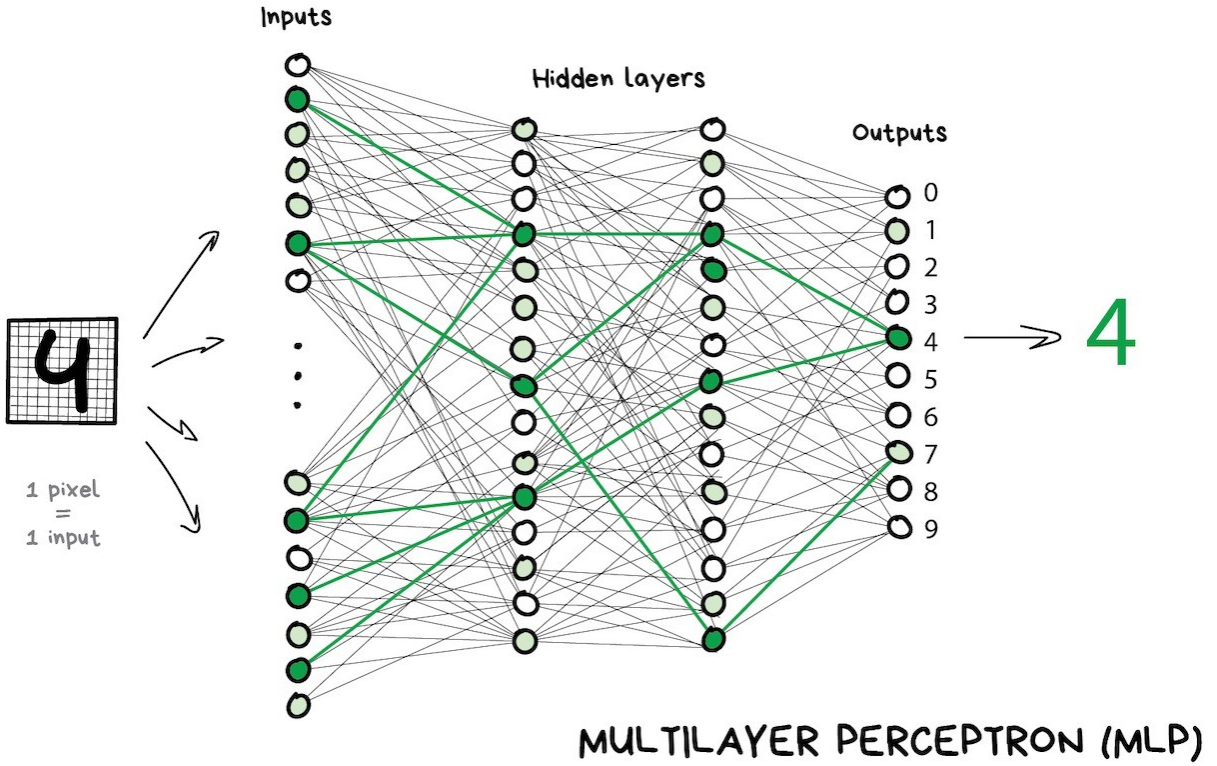
Deep Neural Networks

- Neural networks with lots of “neurons” and layers



Deep Neural Networks

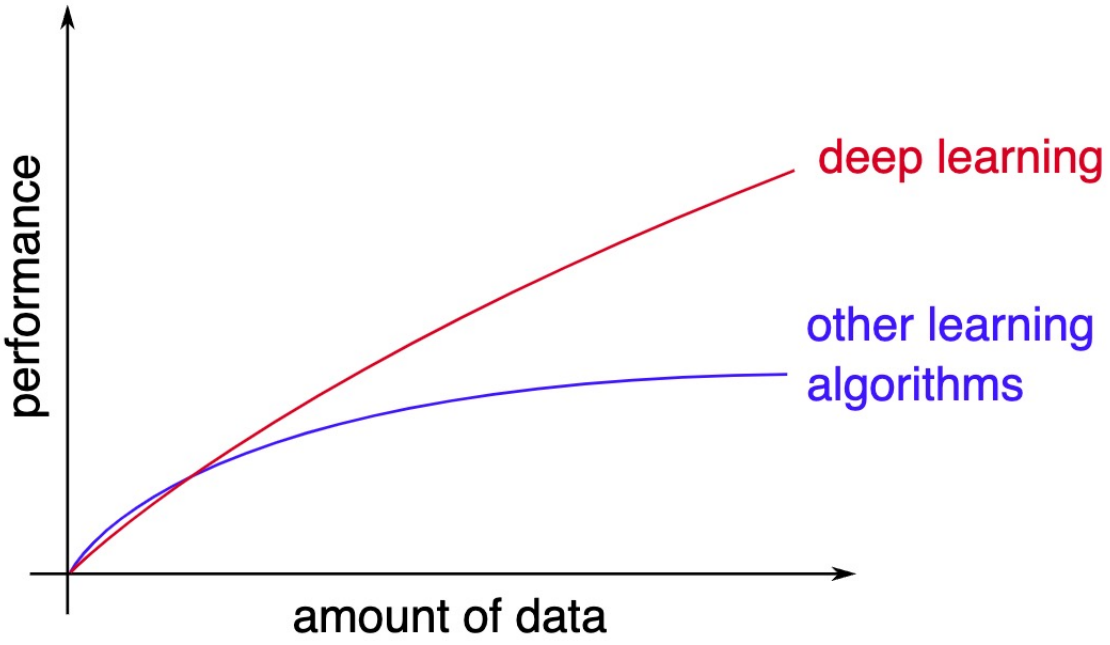
- Neural networks with lots of “neurons” and layers
- Flexible network modules
- Can capture non-linear relationships in data



https://vas3k.com/blog/machine_learning/

Deep Neural Networks

- Neural networks with lots of “neurons” and layers
- Flexible network modules
- Can capture non-linear relationships in data
- Can take advantage of large amounts of training data



Open Source Tools and Models

PYTORCH

Microsoft CNTK

Caffe2

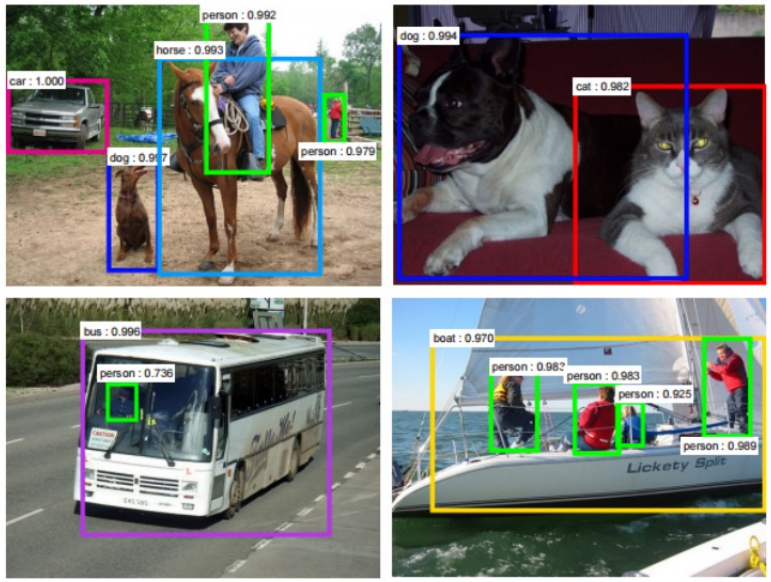
dmlc mxnet

gensim

spaCy

theano

DL for Computer Vision

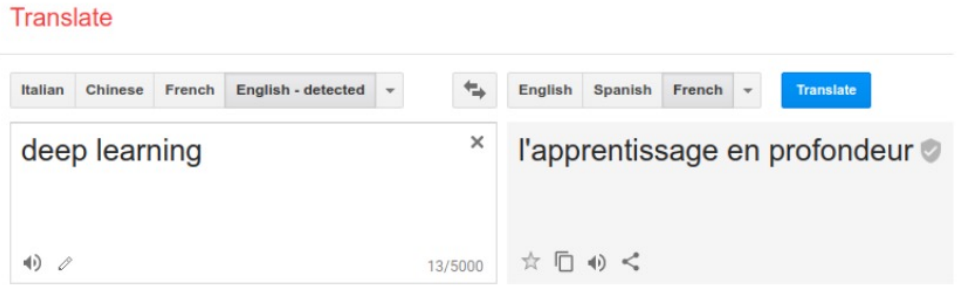


[Faster R-CNN - Ren 2015]

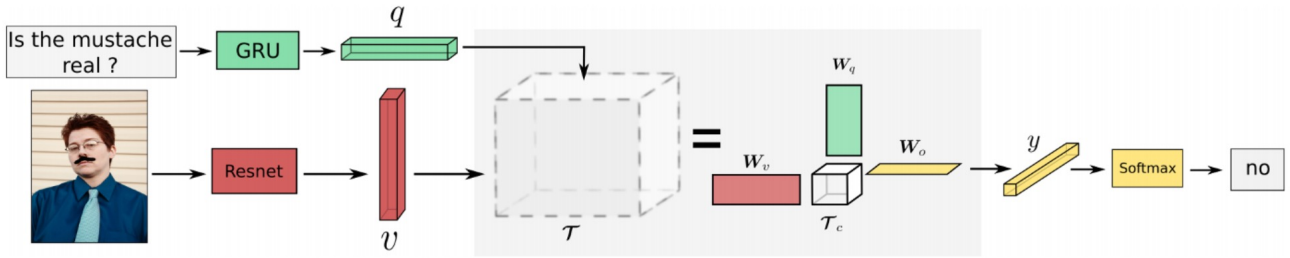
[NVIDIA dev blog]

DL for Natural Language Processing

[Google Inbox Smart Reply]



DL for CV + NLP



[VQA - Mutan 2017]

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

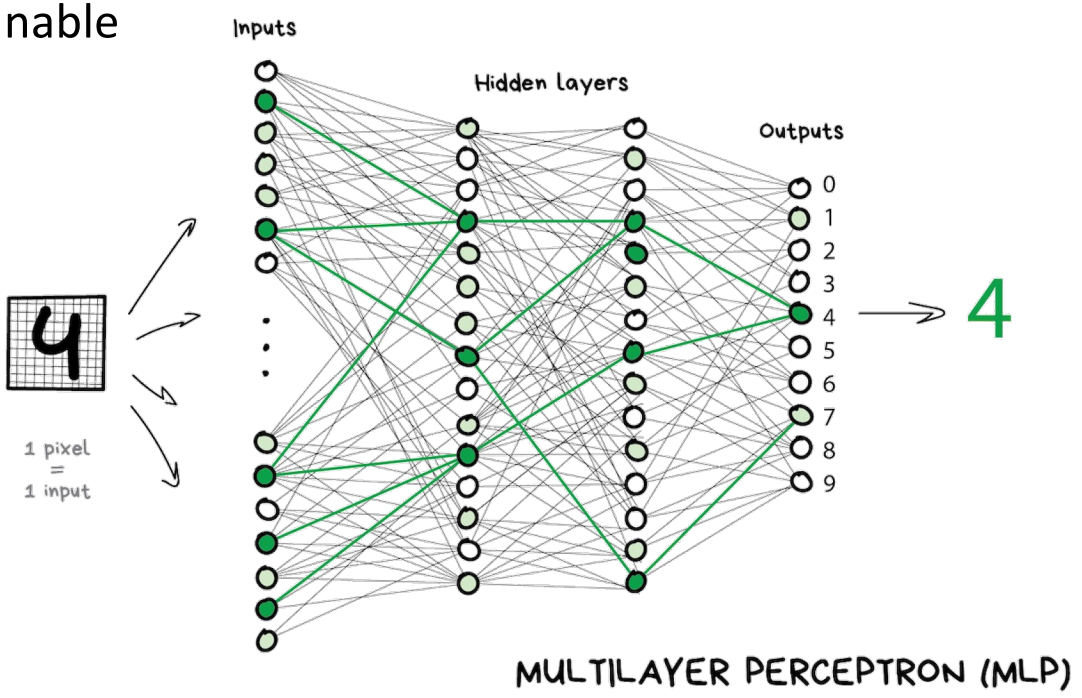
"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

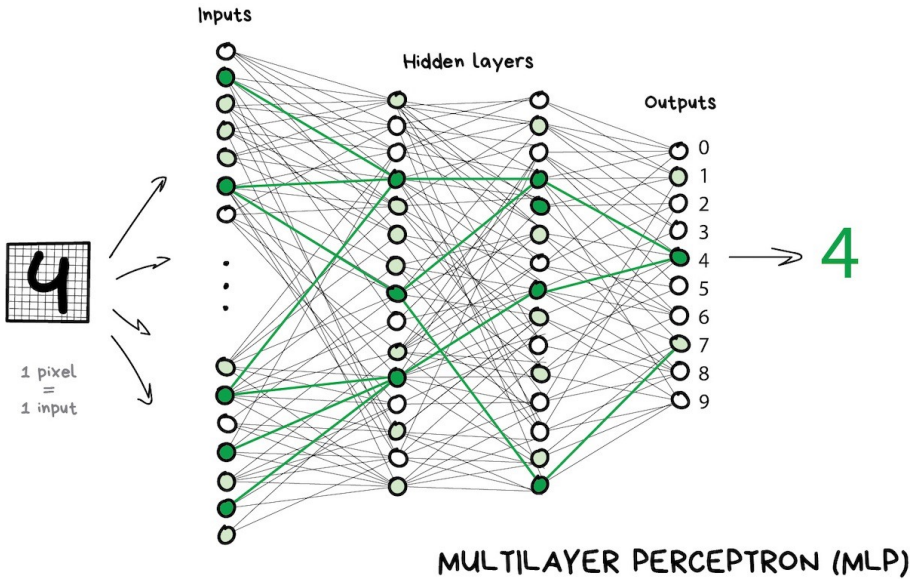
[Karpathy 2015]

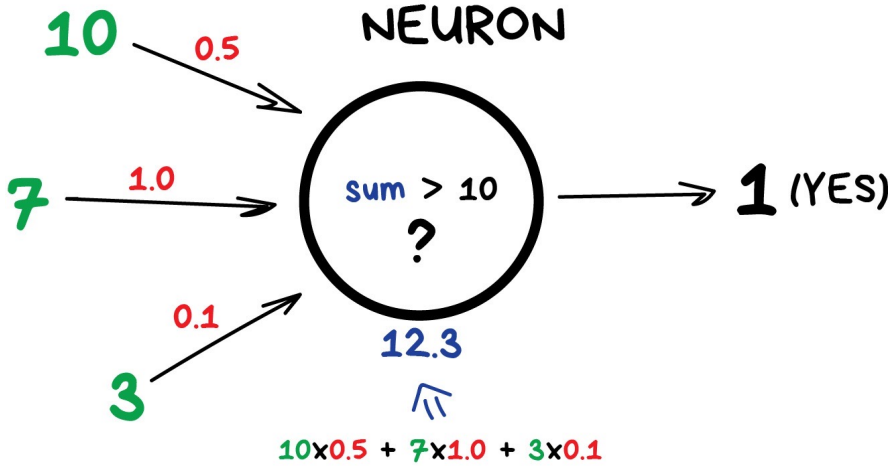
Neural Networks for Classification

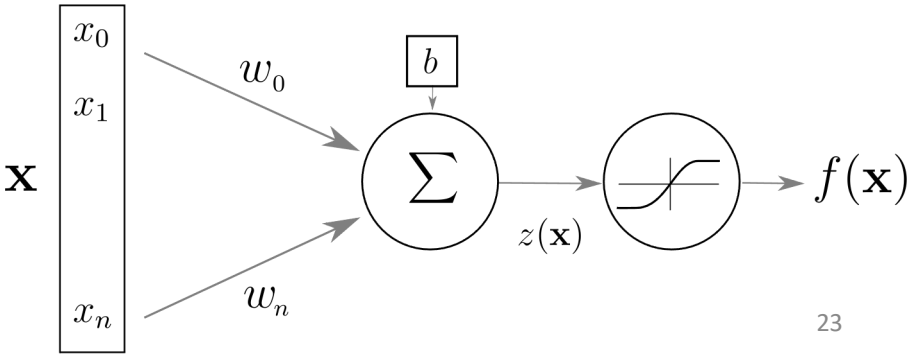
- Neural networks are functions with tunable parameters
  - Lots of parameters
- K-Class Classification
  - $\mathbf{f}(\cdot;\theta)\colon \mathbb{R}^N \to (0,1)^K$
- Sample s in dataset S:
  - $\mathbf{x}^s \in \mathbb{R}^N$
- Expected output:
  - $y^s \in [0, K-1]$
- Output is a conditional probability distribution:
  - $\mathbf{f}(\mathbf{x}^s;\theta)_c = P(Y = c|X = \mathbf{x}^s)$



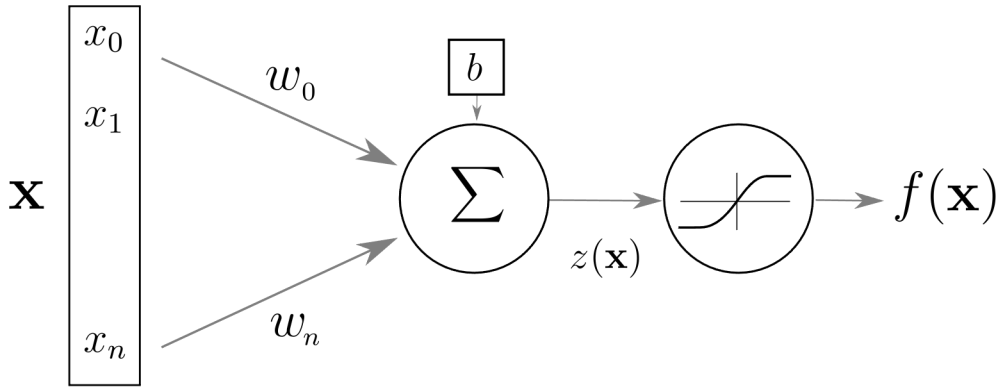
Artificial Neuron







Artificial Neuron

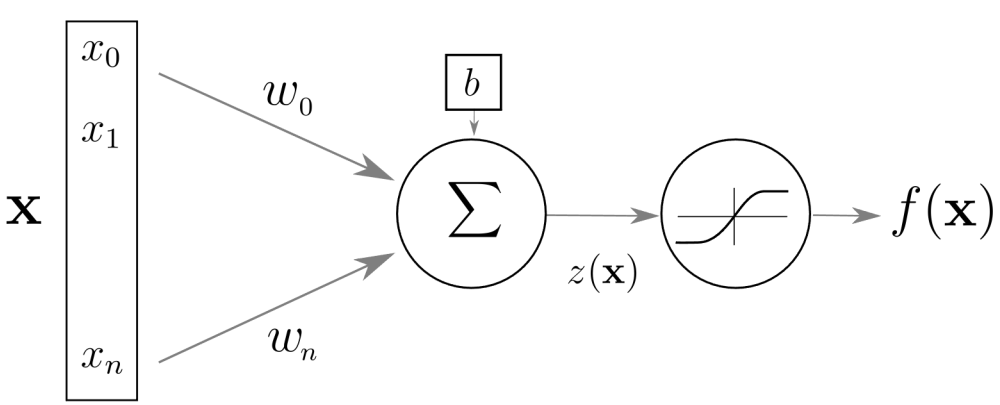


$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$
$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

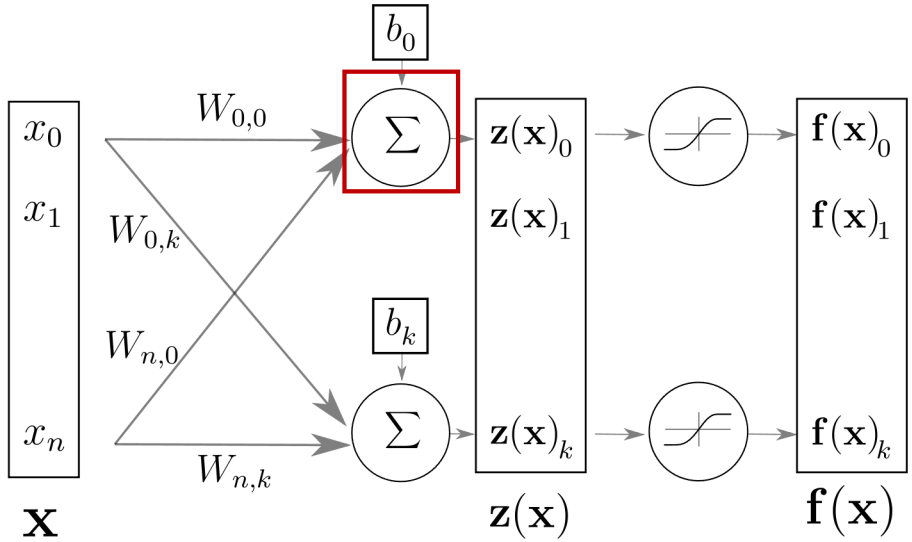
e.g., $[0.01,0.03,0.04]^T x[1,2,3]+9$

$\mathbf{x}, f(\mathbf{x})$ input and output
$z(\mathbf{x})$ pre-activation (intermediate results)
$\mathbf{w}, b$ weights and bias
g activation function

Layer of Neurons



$$z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$
$$f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$$

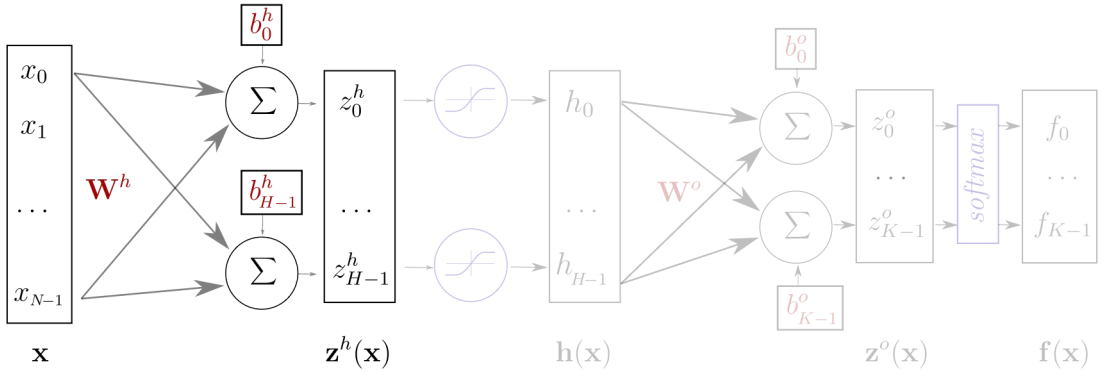


$\mathbf{x}, f(\mathbf{x})$ input and output
$z(\mathbf{x})$ pre-activation (intermediate results)
$\mathbf{w}, b$ weights (vector) and bias (scalar)
g activation function

$$\mathbf{f}(\mathbf{x}) = g(\mathrm{z(x)}) = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

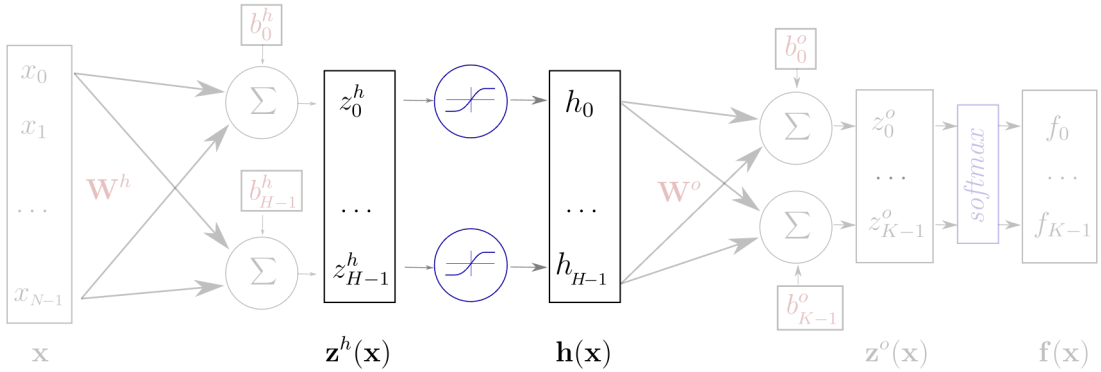
$\mathbf{W}, \mathbf{b}$ now matrix and vector

One Hidden Layer Network



$$\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$$

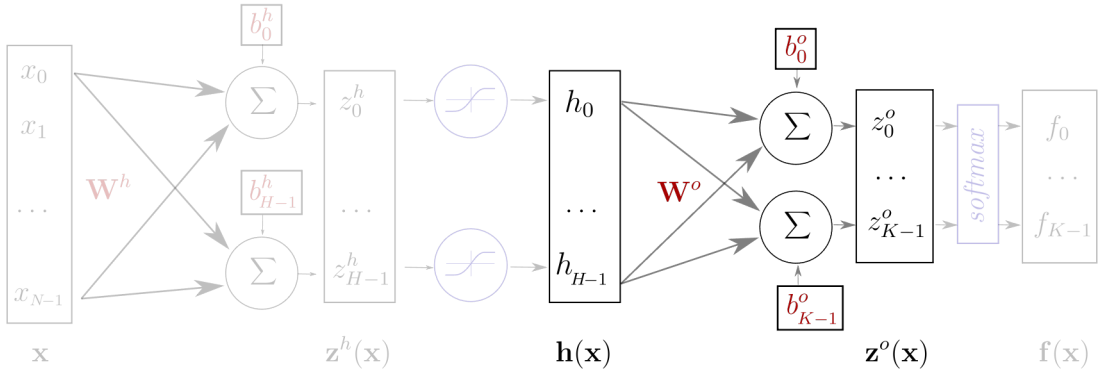
One Hidden Layer Network



$$\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$$

$$\mathbf{h}(\mathbf{x}) = g\left(\mathbf{z}^h(\mathbf{x})\right) = g\left(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h\right)$$

One Hidden Layer Network

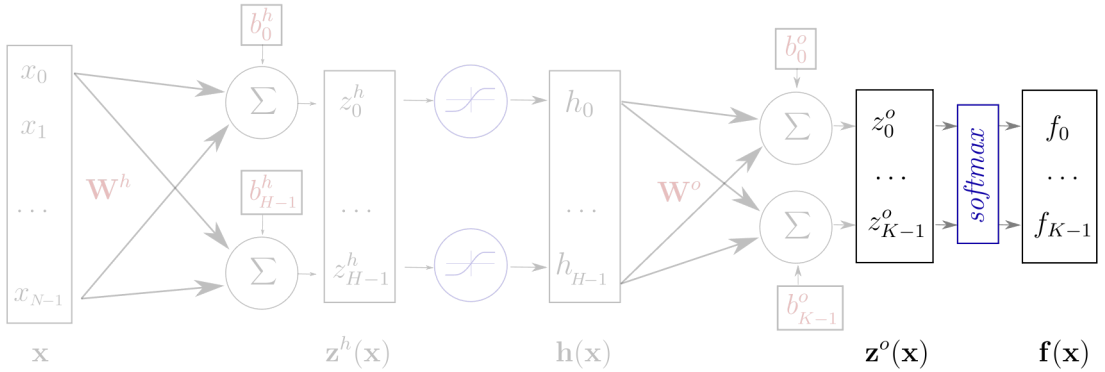


$$\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$$

$$\mathbf{h}(\mathbf{x}) = g\left(\mathbf{z}^h(\mathbf{x})\right) = g\left(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h\right)$$

$$\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$$

One Hidden Layer Network



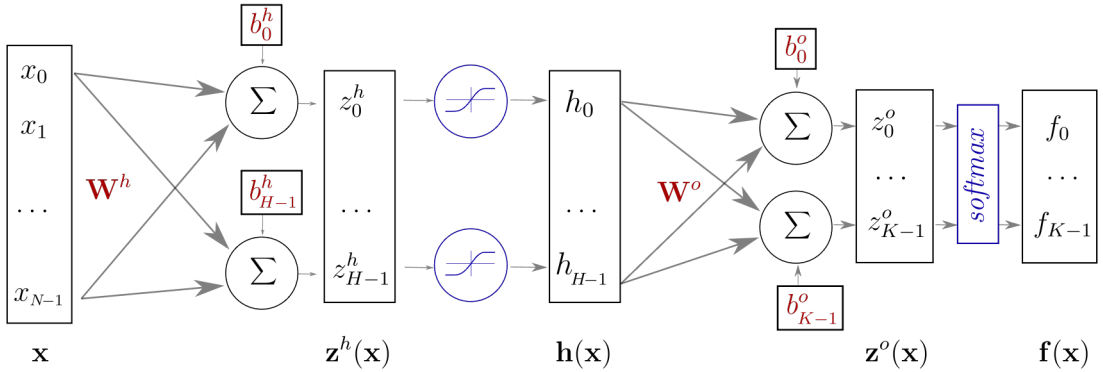
$$\mathbf{z}^h(\mathbf{x}) = \mathbf{W}^h\mathbf{x} + \mathbf{b}^h$$

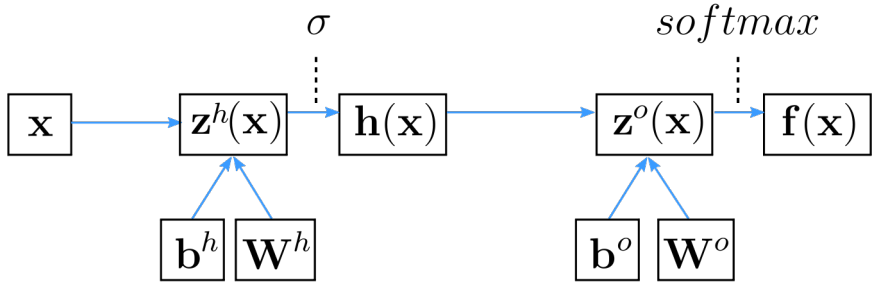
$$\mathbf{h}(\mathbf{x}) = g\left(\mathbf{z}^h(\mathbf{x})\right) = g\left(\mathbf{W}^h\mathbf{x} + \mathbf{b}^h\right)$$

$$\mathbf{z}^o(\mathbf{x}) = \mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o$$

$$\mathbf{f}(\mathbf{x}) = softmax(\mathbf{z}^o) = softmax(\mathbf{W}^o\mathbf{h}(\mathbf{x}) + \mathbf{b}^o)$$

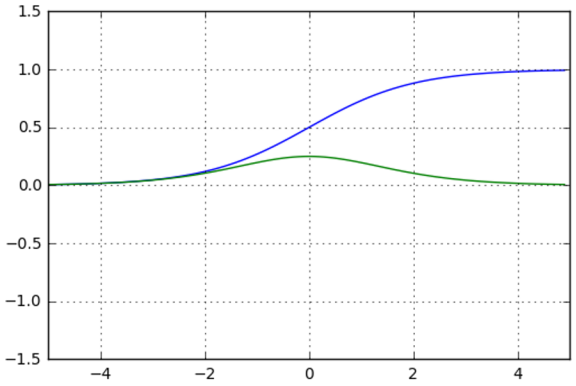
One Hidden Layer Network





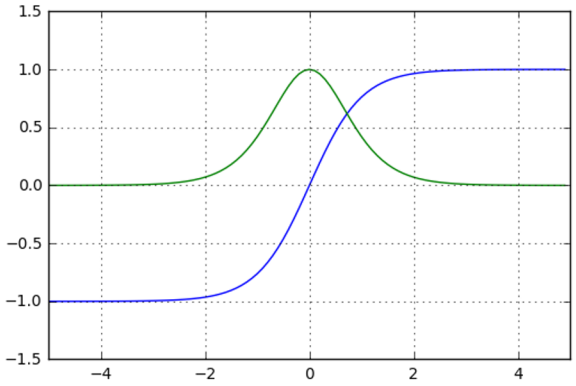
Computational Graph

Element-wise activation functions



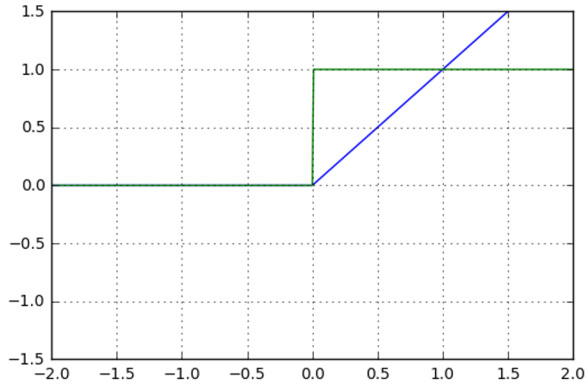
$$\text{sigm}(x) = \frac{1}{1+e^{-x}}$$

$$\text{sigm}'(x) = \text{sigm}(x)(1-\text{sigm}(x))$$



$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

$$\tanh'(x) = 1-\tanh(x)^2$$

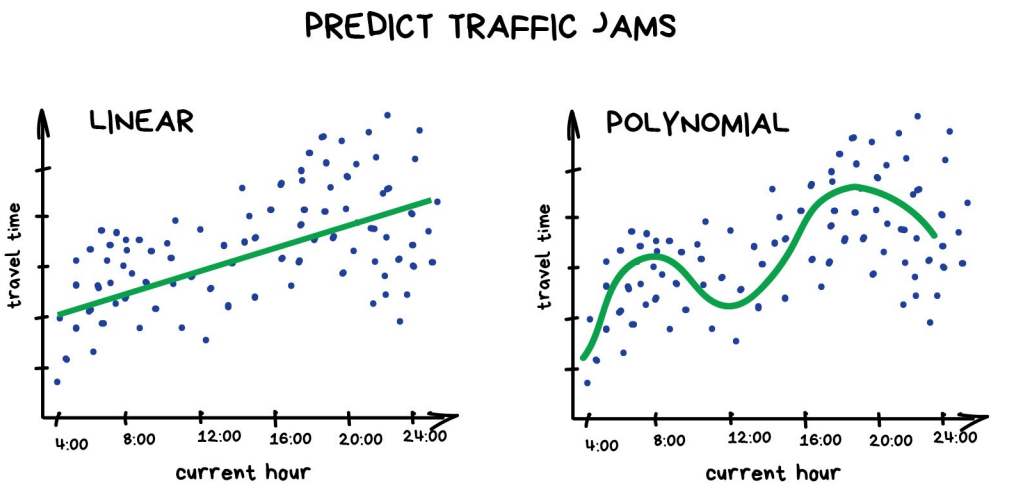


$$\text{relu}(x) = \max(0,x)$$

$$\text{relu}'(x) = 1_{x>0}$$

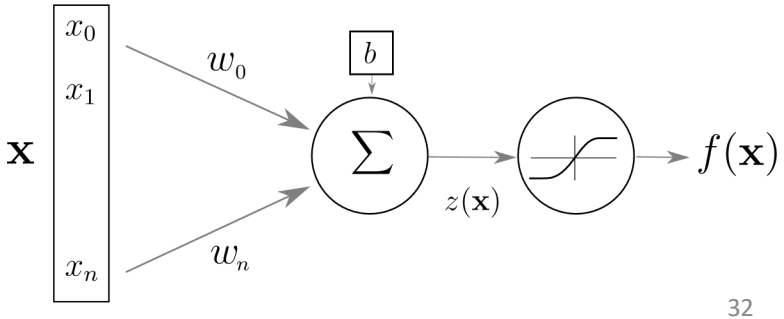
blue: activation function
green: derivative

Activation Functions



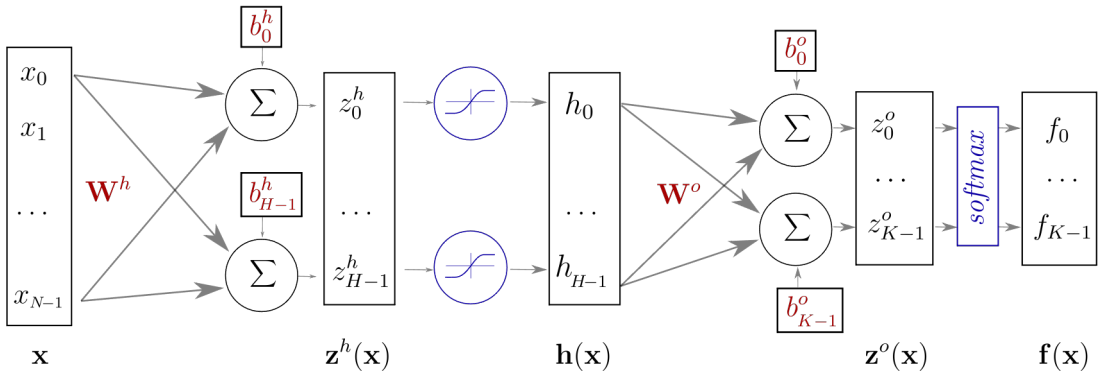
What if polynomial regression still could not find a good fit?

- Activation functions introduce non-linearities
- Linear
  - $z(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$
- Non-linear
  - $f(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + b)$



Softmax function

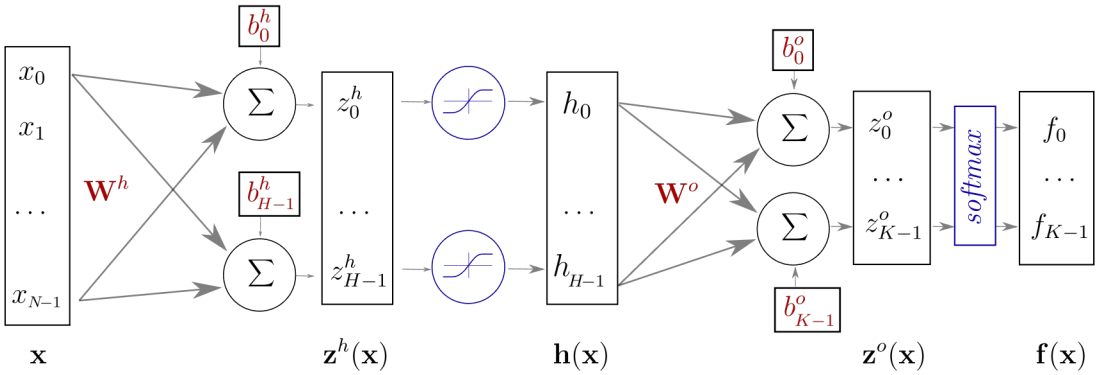
- *"Any time we wish to represent a probability distribution over a discrete variable with **n** possible values, we may use the softmax function. This can be seen as a generalization of the sigmoid function which was used to represent a probability distribution over a binary variable." Deep Learning*



$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

Softmax Example

- Class 0: [1, 0, 0]
- Class 1: [0, 1, 0]
- Class 2: [0, 0, 1]
- True *y* for an input *x*
  - [0, 1, 0]



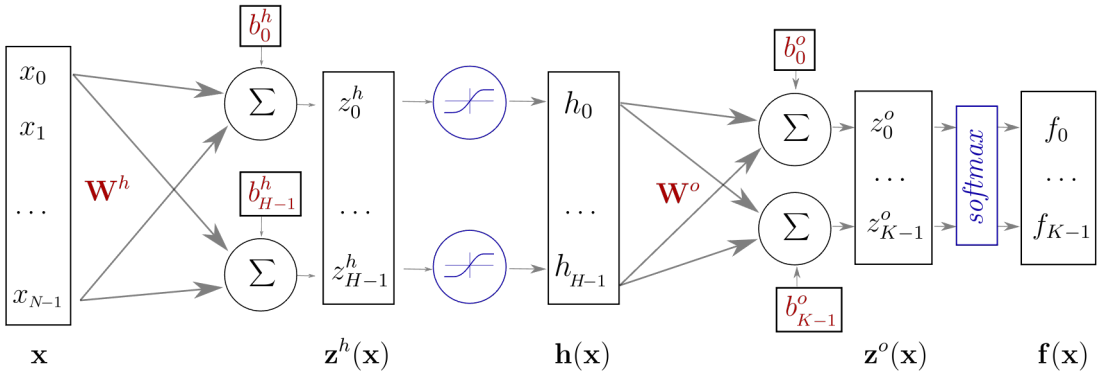
- Softmax results scale to probabilities and sum up to 1.0
  - [0.09003057 0.66524096 0.24472847]
  - $p(Y = c|X = \mathbf{x}) = \text{softmax}(\mathbf{z}(\mathbf{x}))_c$
  - Might use argmax() to return [0, 1, 0] -> Class 1

$$softmax(\mathbf{x}) = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \cdot \begin{bmatrix} e^{x_1} \\ e^{x_2} \\ \vdots \\ e^{x_n} \end{bmatrix}$$

https://machinelearningmastery.com/softmax-activation-function-with-python/#:~:text=The%20softmax%20function%20is%20used%20as%20the%20activation%20function%20in,more%20than%20two%20class%20labels.

Training the network

- From x = [x_0…x_{n-1}], the network generates [0.09003057 0.66524096 0.24472847]
  - What if the true label is [1, 0, 0]?
- Adjust $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ so that hopefully the next prediction will be closer to the true label

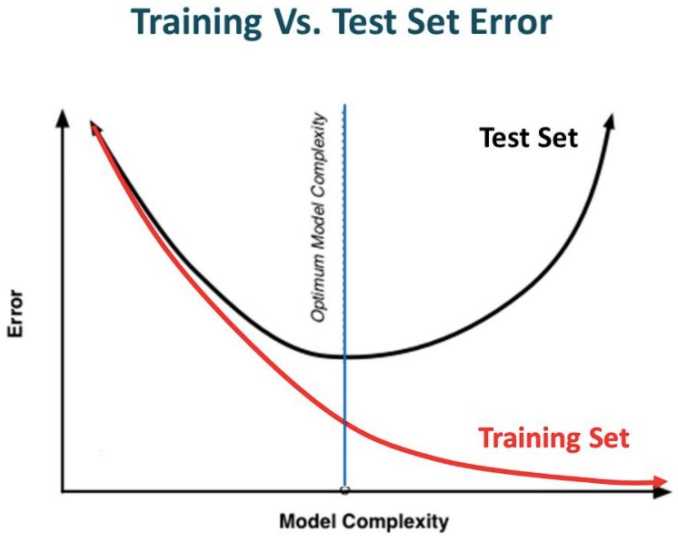


Training the network

- Using the training data, find $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ that minimize some loss function w.r.t the quality of the prediction
  - $l(\mathbf{f}(\mathbf{x}^s; \theta), y^s)$
- We need some way to quantitatively compare the two probability distributions: [0.09003057 0.66524096 0.24472847] and [1, 0, 0]
  - e.g., using cross entropy to measure the difference between the predicted distribution to the ground truth label
- The comparison results will give the network some directions to generate a prediction closer to [1, 0, 0]
  - by updating $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$

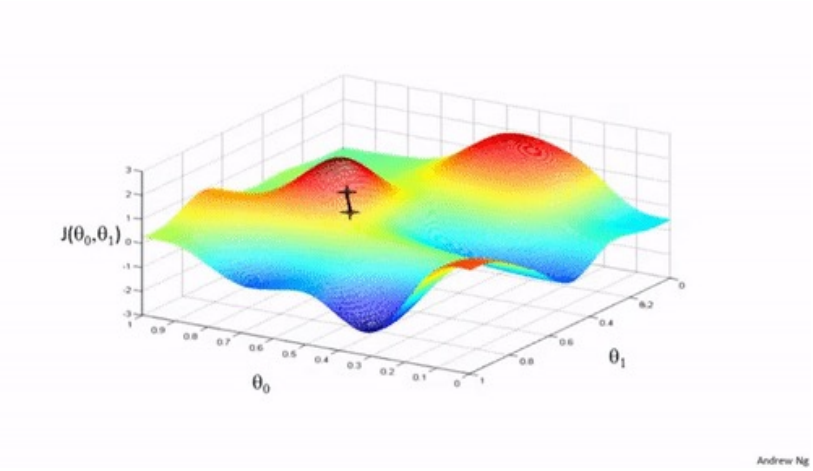
Avoid Overfitting

- Typically, minimizing $l(\mathbf{f}(\mathbf{x}^s;\theta), y^s)$ is not enough
  - Especially with complex models (lots of parameters)
- Regularization
  - We might want to also add a regularization term
    - e.g., L2: $\lambda\Omega(\theta) = \lambda(||W^h||^2 + ||W^o||^2)$
    - Recall: $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$
- Dropout
- Early stopping



https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/

Gradient Descent

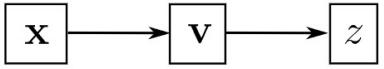


- Initialize $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ randomly
- For E epochs perform:
- For all samples ($s \subset S$)
  - Compute gradients from all samples: $\Delta = \nabla_\theta L_s(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)
- Stop when reaching criterion:
  - e.g., error stops decreasing when computed on validation set

Computing Gradients

- Output Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^o_{i,j}}$
- Hidden Weights: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial W^h_{i,j}}$
- The network is a composition of differentiable modules
- Instead of computing the gradients for each layer individually, we can apply the "chain rule"
- Easy to calculate with complex network and loss functions

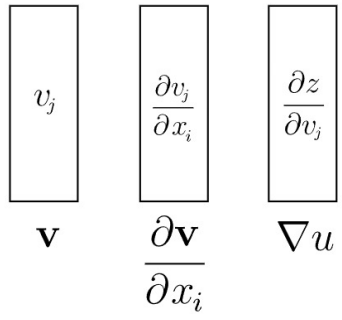
- Output bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^o_i}$
- Hidden bias: $\frac{\partial l(\mathbf{f}(\mathbf{x}),y)}{\partial b^h_i}$



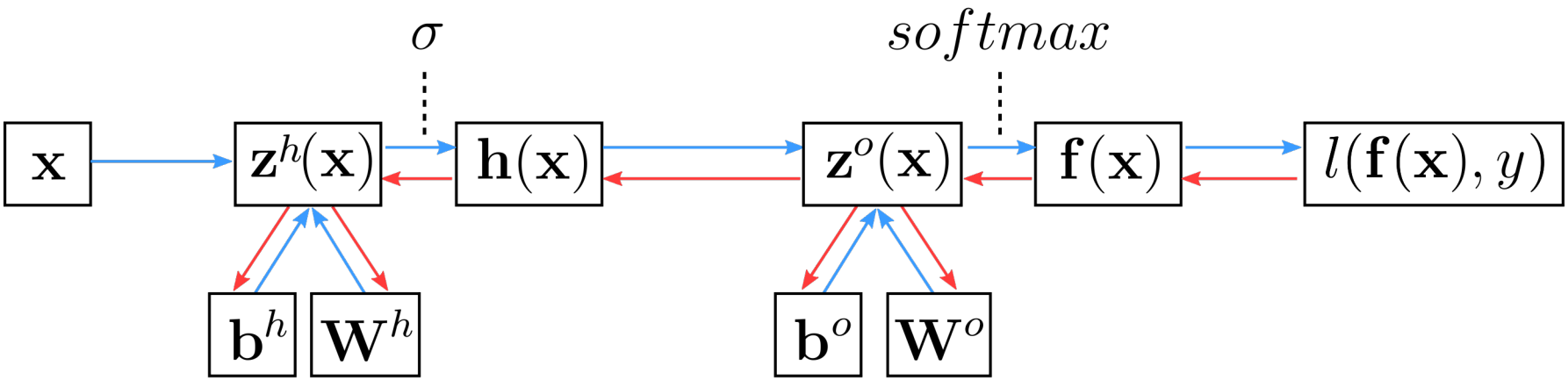
$$z = u(\mathbf{v}(\mathbf{x}))$$

chain-rule

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial v_j}\frac{\partial v_j}{\partial x_i} = \nabla u . \frac{\partial \mathbf{v}}{\partial x_i}$$

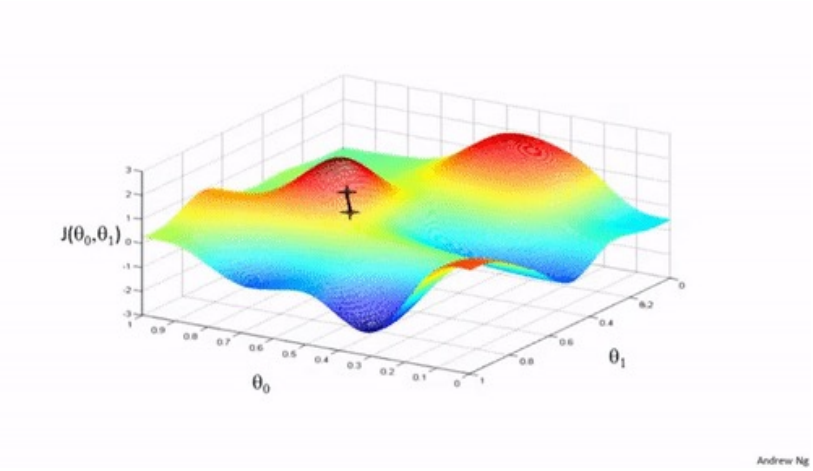


Backpropagation



https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd
https://machinelearningmastery.com/difference-between-backpropagation-and-stochastic-gradient-descent/

Stochastic Gradient Descent with Mini Batch

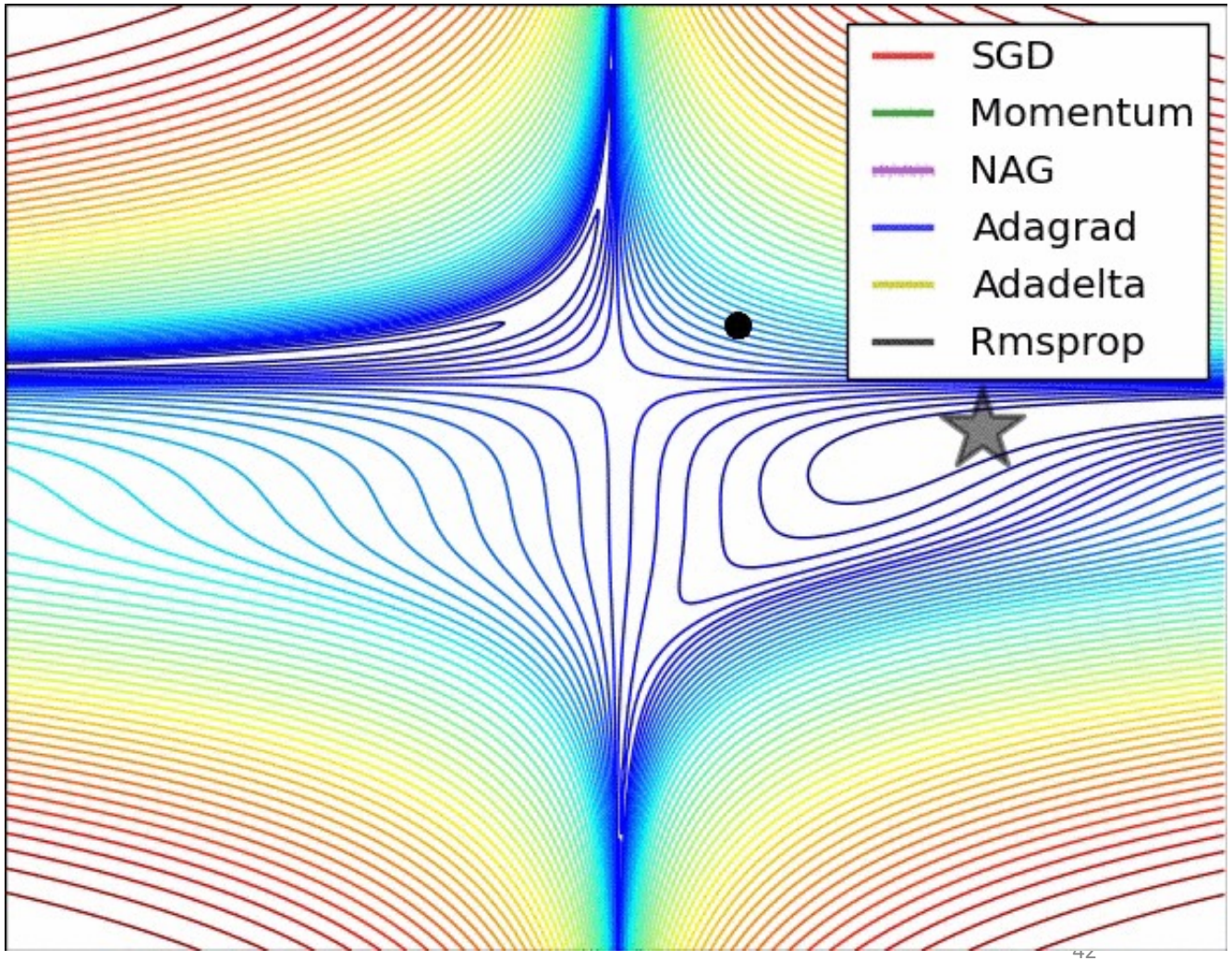


- Initialize $\theta = (\mathbf{W}^h; \mathbf{b}^h; \mathbf{W}^o; \mathbf{b}^o)$ randomly
- For E epochs perform:
- Randomly select a small batch (mini-batch) of samples ($B \subset S$)
  - Compute gradients: $\Delta = \nabla_\theta L_B(\theta)$
  - Update parameters: $\theta \leftarrow \theta - \eta\Delta$
  - $\eta > 0$ is called the learning rate
- Repeat until the epoch is completed (all of S is covered)
- Stop when reaching criterion:
  - e.g., error stops decreasing when computed on validation set

Watch me: https://www.youtube.com/watch?v=vMh0zPT0tLI

Initialization and Learning

- Normalize your input
- Initialize weights with random numbers (bias to 0)
- Adjust SGD learning rate
- Momentum
  - Accumulate gradients across successive updates
- Other optimizers

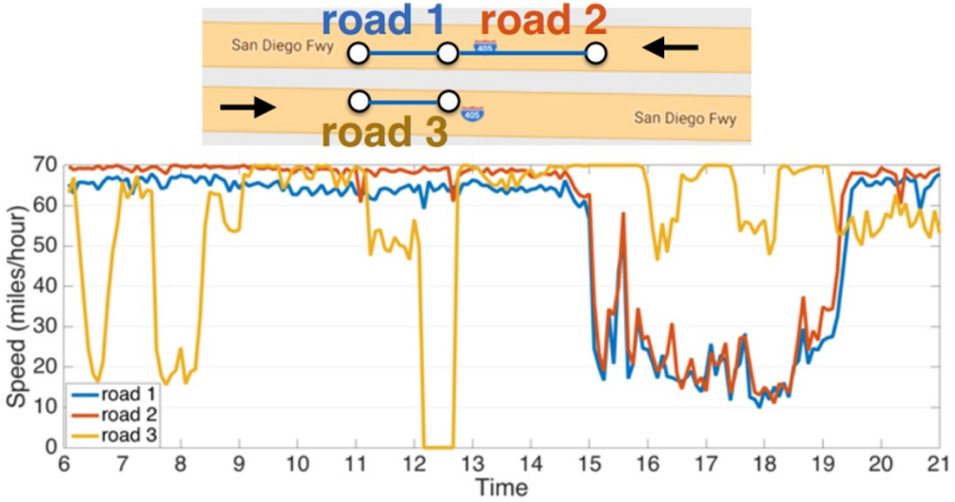


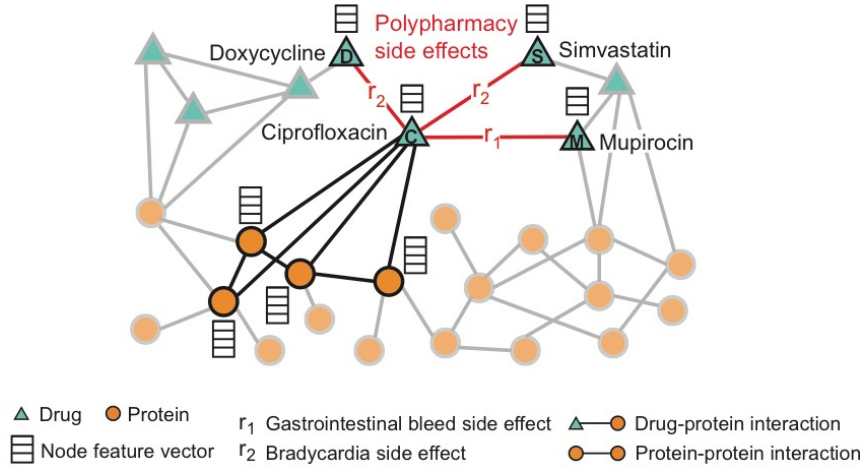
Credits: Alec Radford

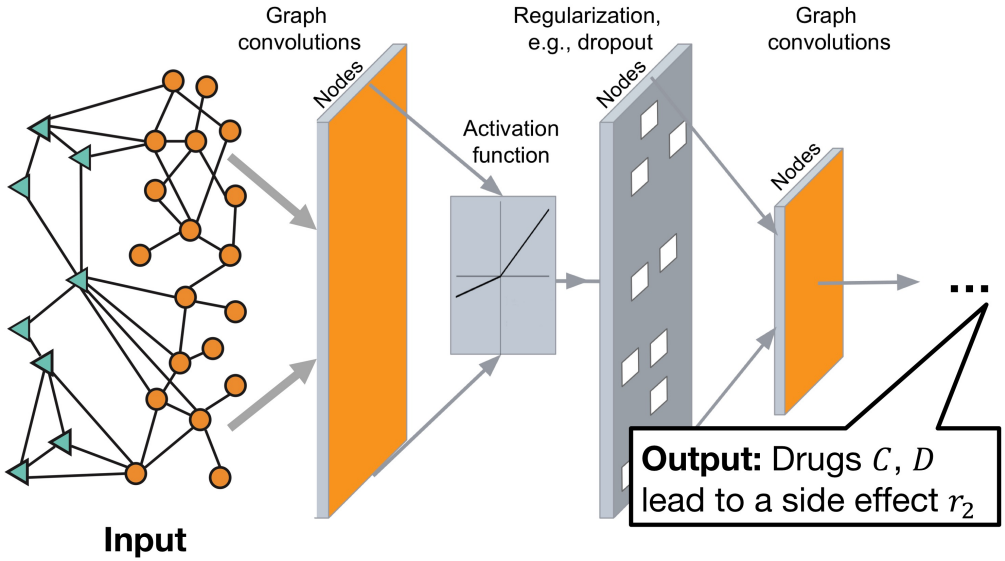
Many Types of Neural Network Architectures

- Graph Neural Networks
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transformer
- We can also mix them!

Graph Neural Networks

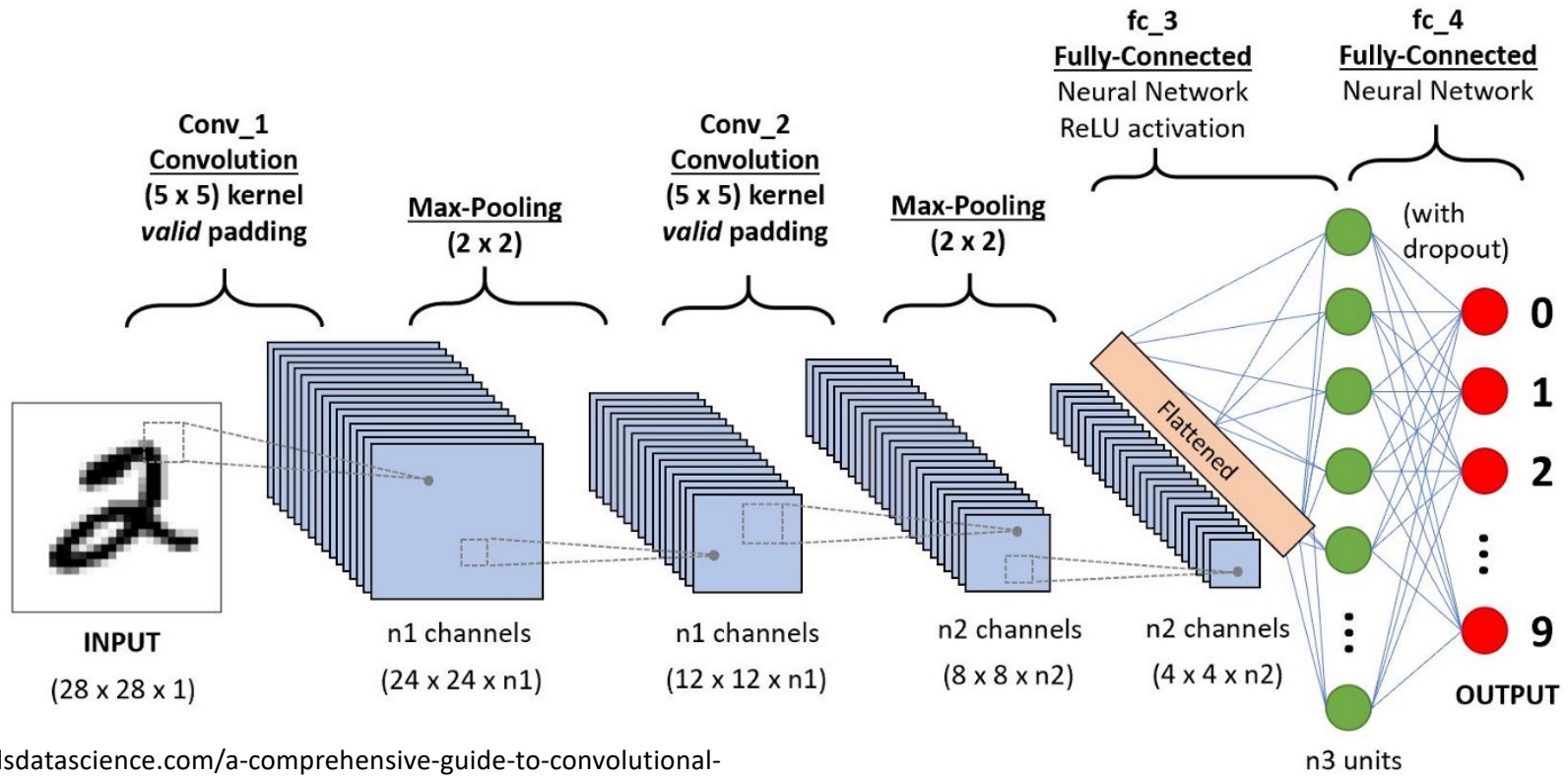






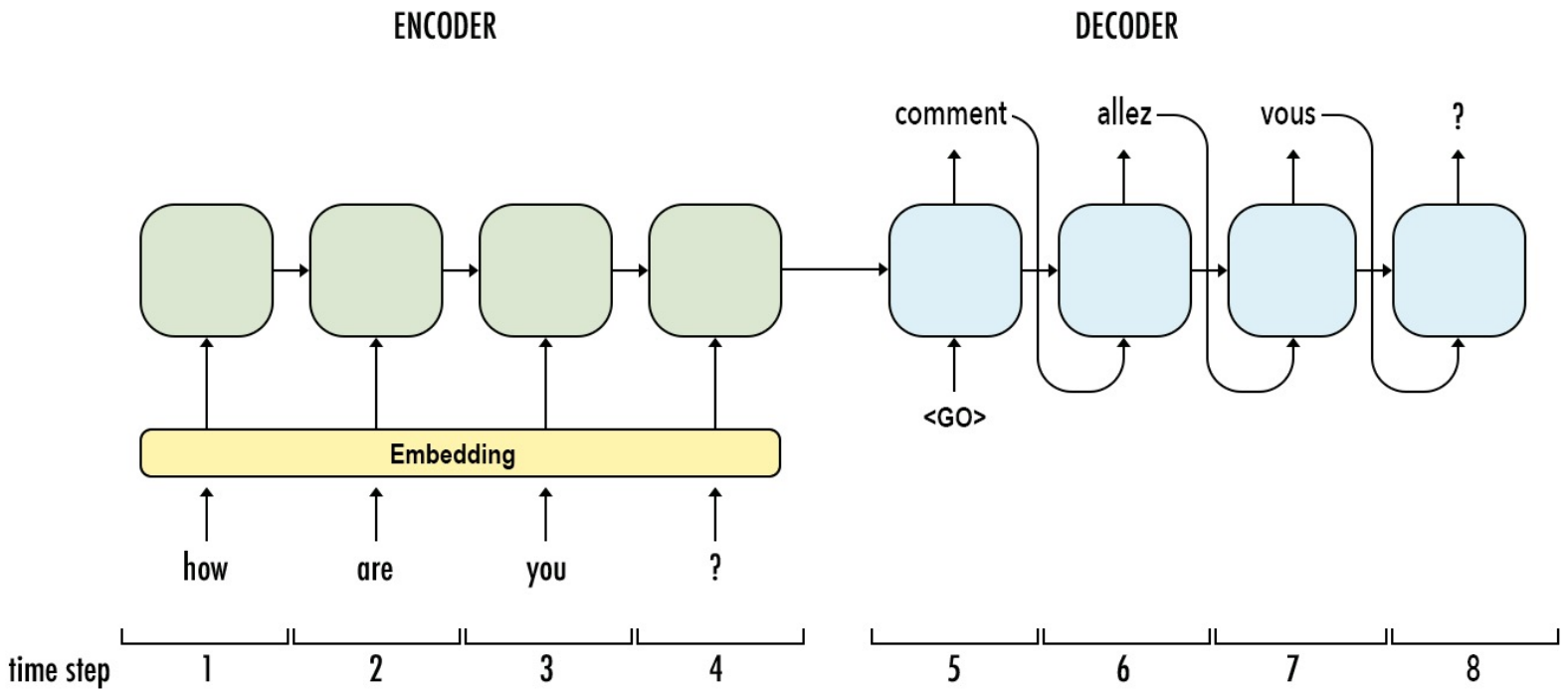
https://giters.com/liyaguang/DCRNN
http://snap.stanford.edu/decagon/

Convolutional Neural Networks



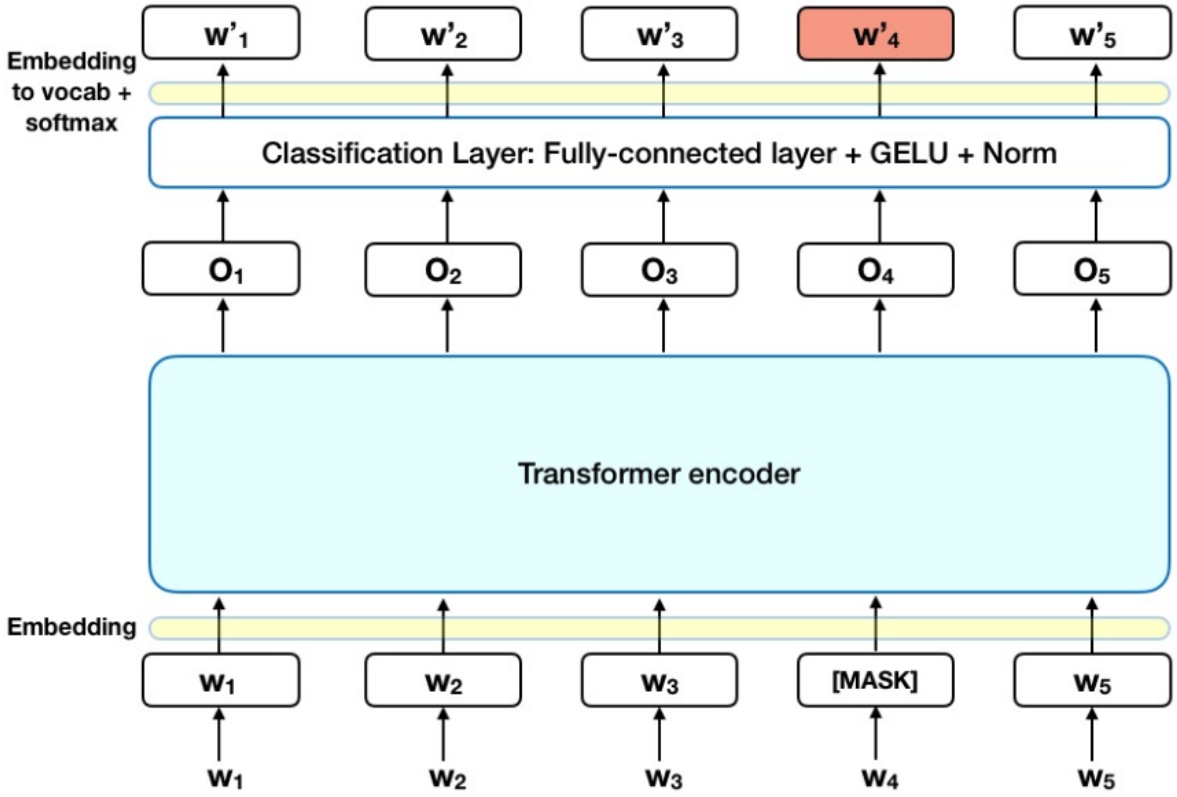
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Recurrent Neural Networks



https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571

Transformer



https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270

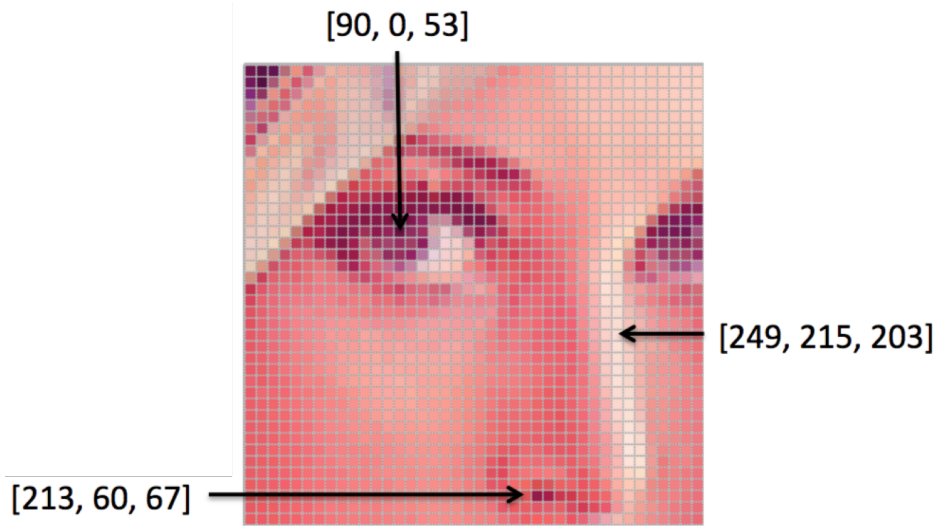
Neural Networks & DL Summary

- DL is very versatile and can capture non-linear relationships
- DL can take advantage of large amounts of training data
- Design a DL network include the decisions of network architectures, loss functions, and regularizer
- Training a DL network concerns initialization (including data normalization), learning rate, learning algorithms, and batch sizes (SGD)
- Many open-source tools, books, and tutorials are available online

Convolutional Neural Networks

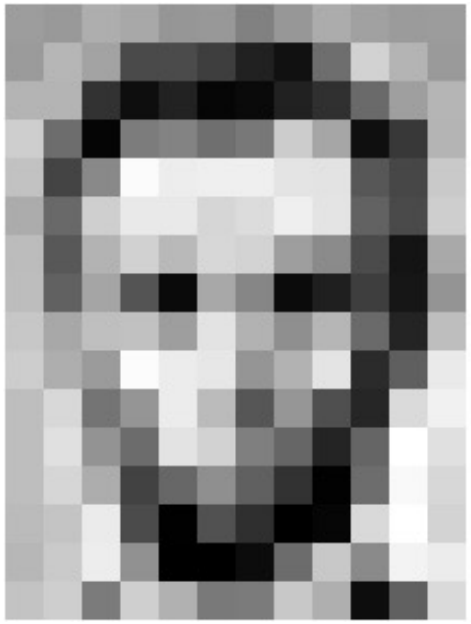
A short introduction

Image as Functions



https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Image as Functions



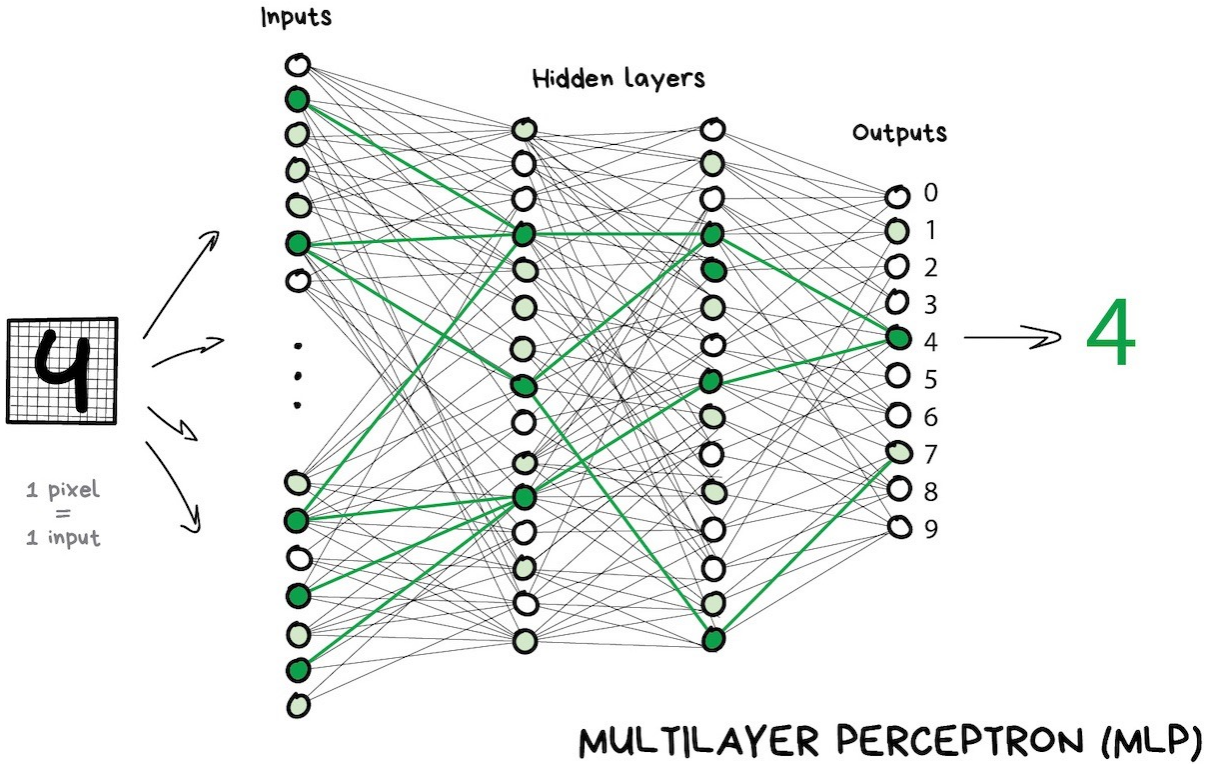
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Image Classification with Dense Layers

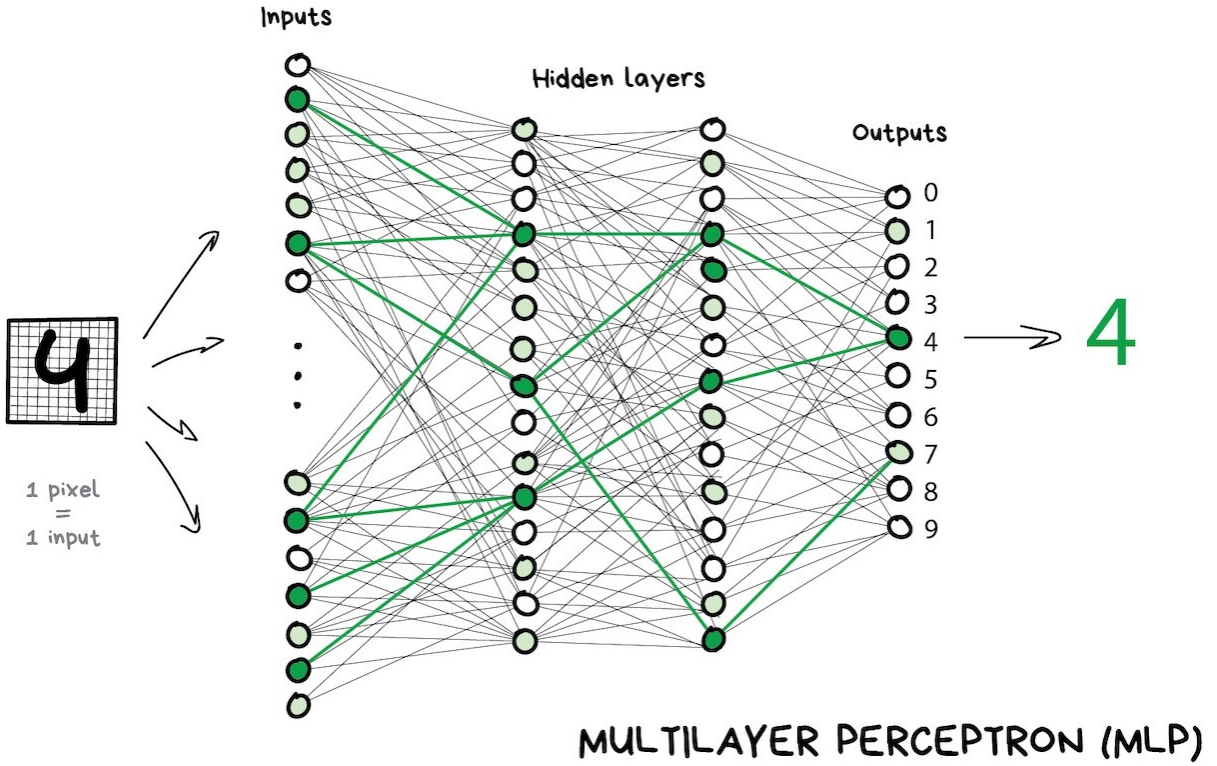
- How many parameters do we need?
  - Two hidden layers
  - Fully connected
  - Assuming a 28 x 28 image, 3 channels (RGB), 2 layers, 1,000 neuron each
    - 28 x 28 x 3 x 2 x 1000 + 2 x 1000 = 4.7M!



https://vas3k.com/blog/machine_learning/

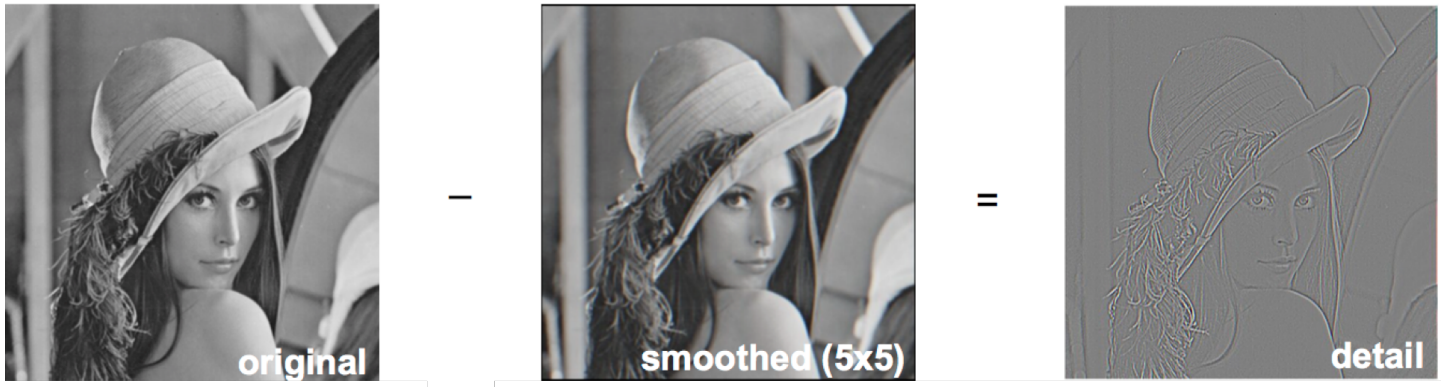
Image Classification with Dense Layers

- Spatial structure is not used
  - Spatial relations between pixel values
- The input image is “flattened”
  - From 2D pixels to 1D inputs

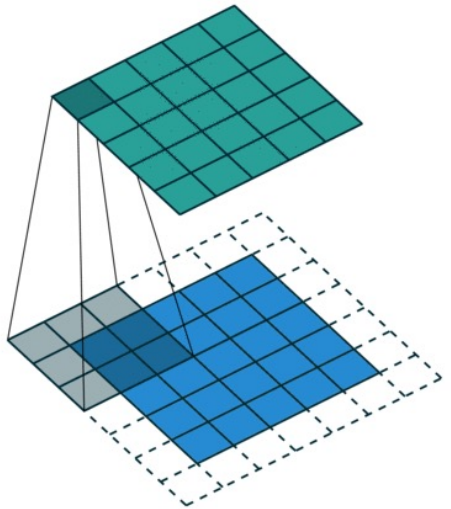


https://vas3k.com/blog/machine_learning/

Image Filters, Convolution, Kernel



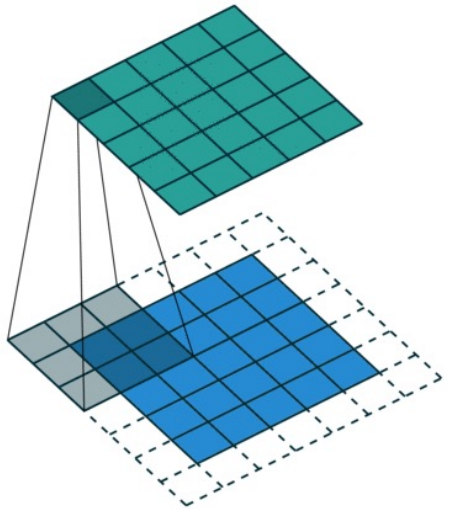
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

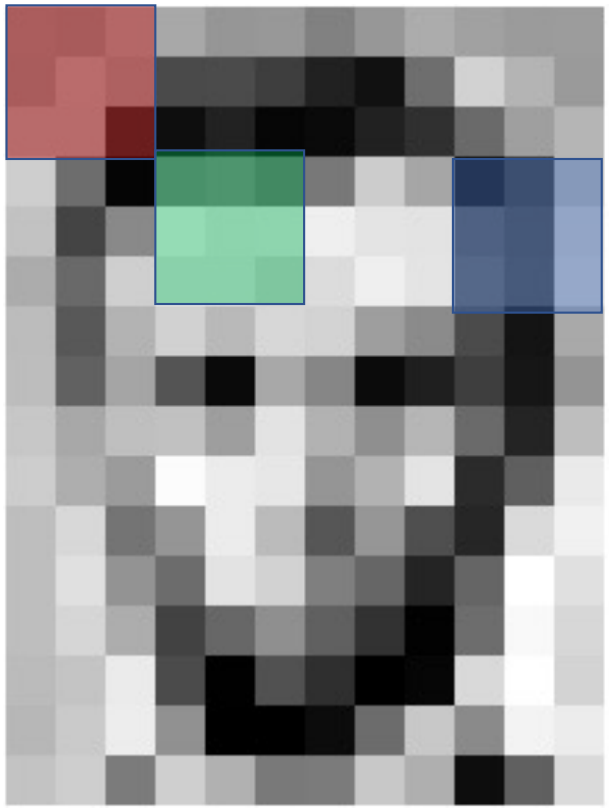
Image Filters, Convolution, Kernel

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



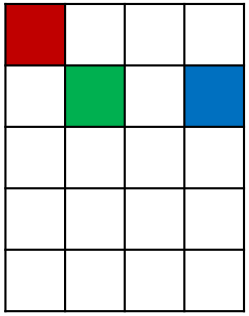
https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

Image Filters, Convolution, Kernel



16 x 12

- Apply a convolution kernel on the image to generate another image
- The output image can have a different size from the input image, depending on the kernel sizes/types, stride sizes, and padding methods



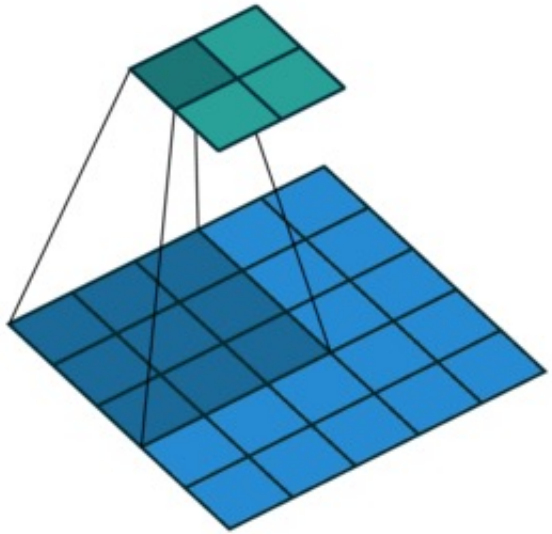
5 x 4

- Each cell in the output image summarizes a 3 x 3 area in the original image
- The same summarization operation for every cell, e.g., use the average intensity

https://github.com/vdumoulin/conv_arithmetic

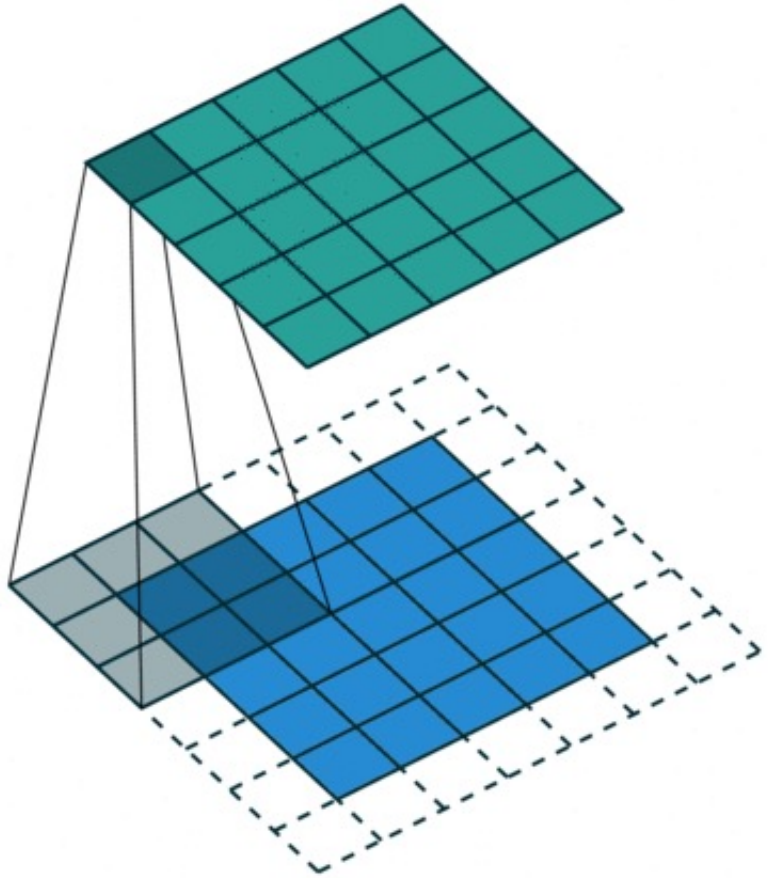
Strides

- Strides: increment step size for the convolution operator
- Reduces the size of the output map
- A stride of 2:

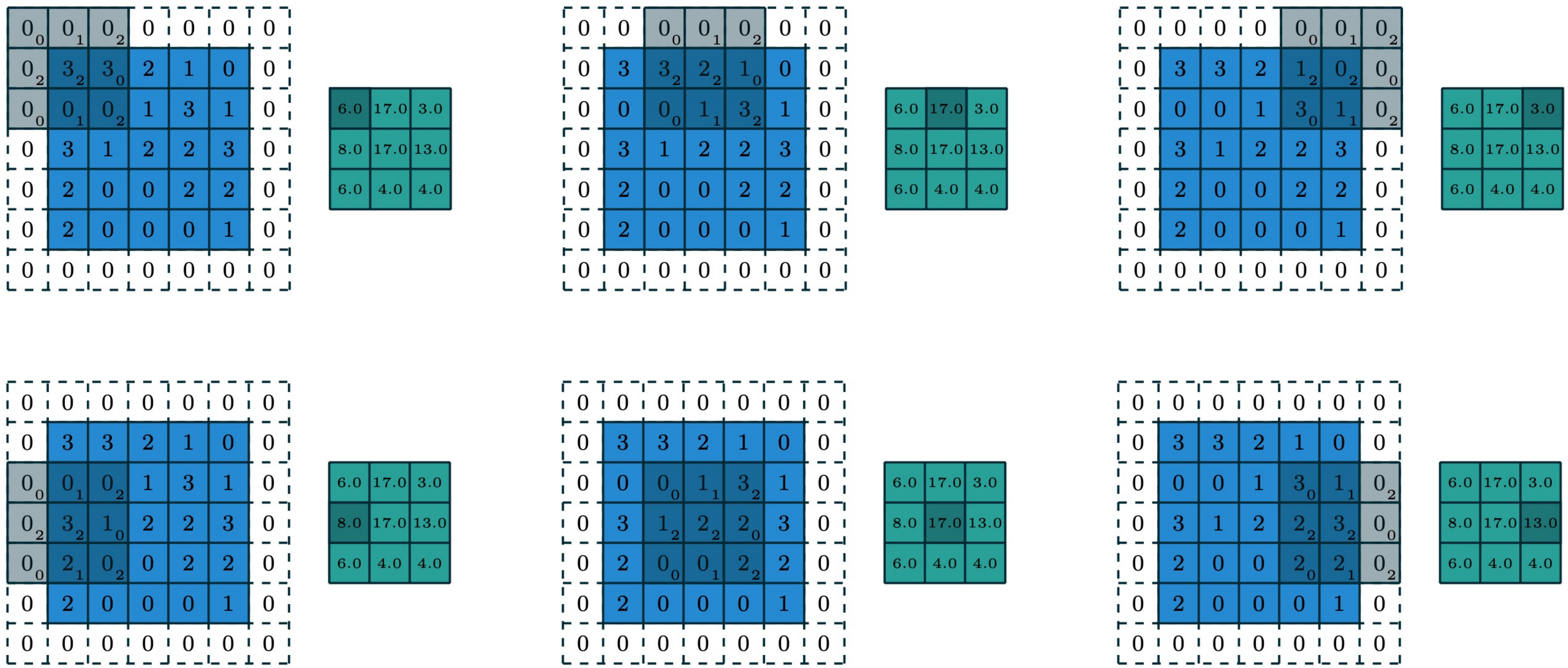


Padding

- Padding: artificially fill borders of image
- Useful to keep spatial dimension constant across filters
- Useful with strides and large receptive fields
- Usually: fill with 0s



Padding



Sizes and Dimensions

- Assuming square input, dimension i x i
- Padding size = p
- Stride size = 1
- Kernel size = k
- Output dimension = o x o

$$o = (i - k) + 2 \times p + 1$$

- Read https://arxiv.org/abs/1603.07285

Sizes and Dimensions

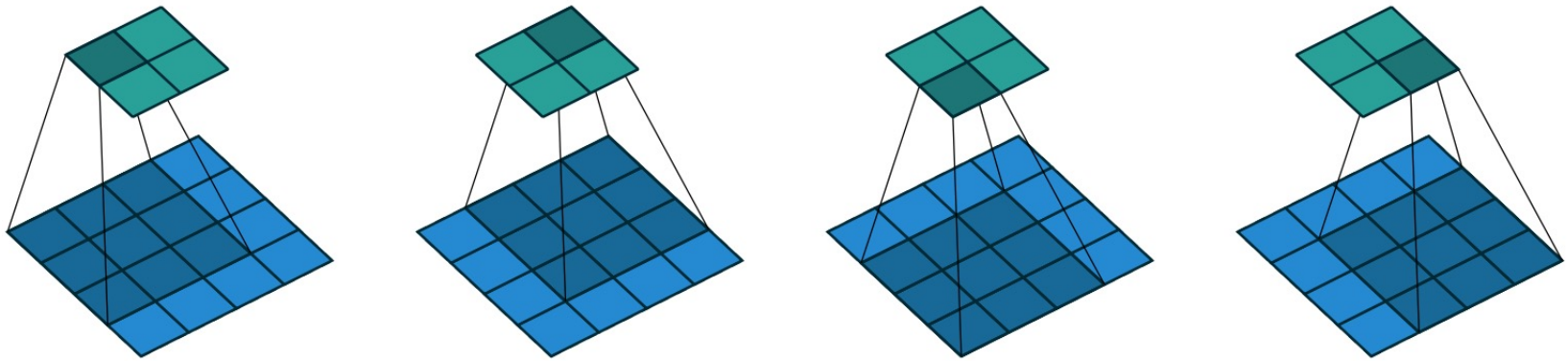


Figure 2.1: (No padding, unit strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

$$o = (i - k) + 2 \times p + 1$$

https://arxiv.org/abs/1603.07285

Sizes and Dimensions

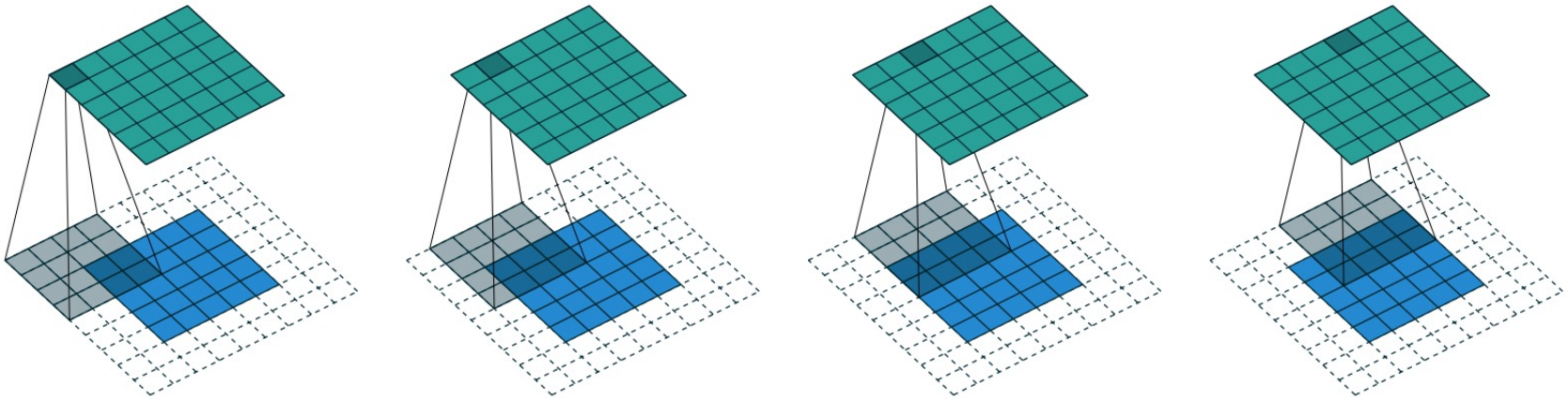


Figure 2.2: (Arbitrary padding, unit strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

$$o = (i - k) + 2 \times p + 1$$

https://arxiv.org/abs/1603.07285

Sizes and Dimensions

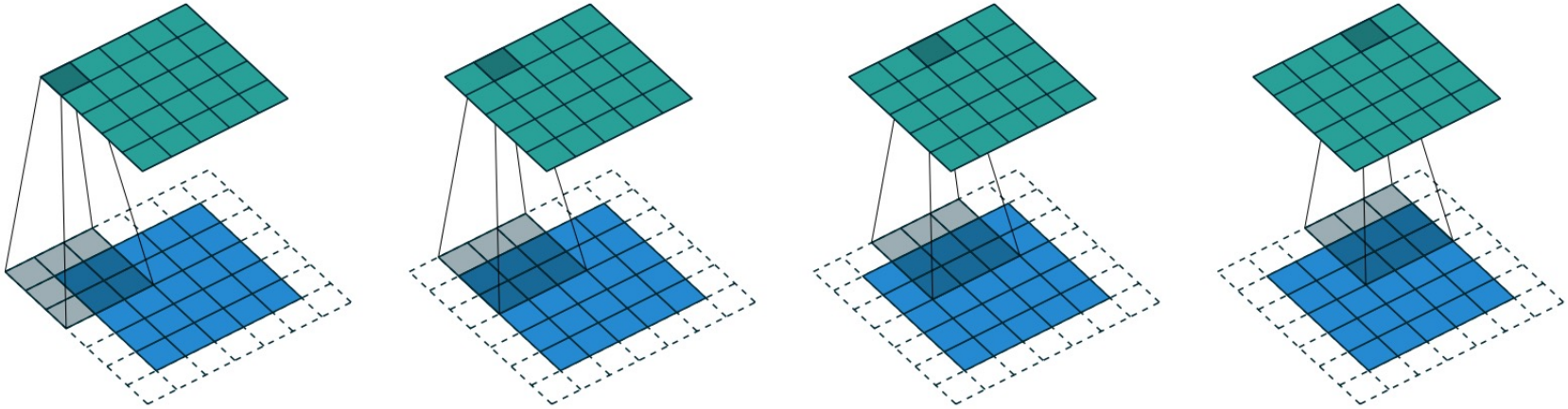


Figure 2.3: (Half padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

$$o = (i - k) + 2 \times p + 1; k = 2 \times p + 1$$

https://arxiv.org/abs/1603.07285

Sizes and Dimensions

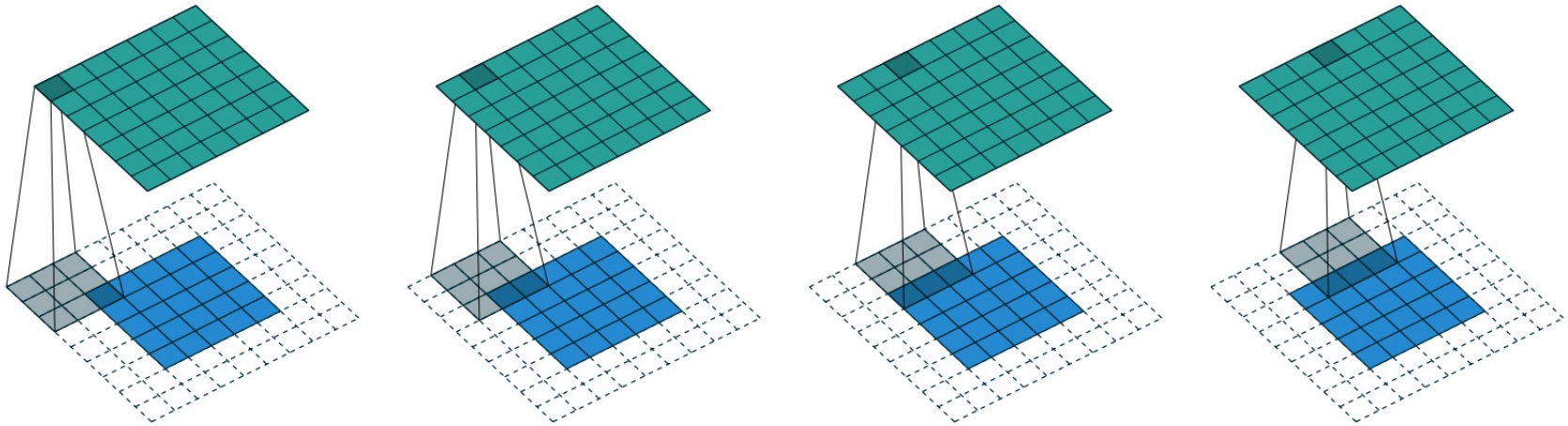


Figure 2.4: (Full padding, unit strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

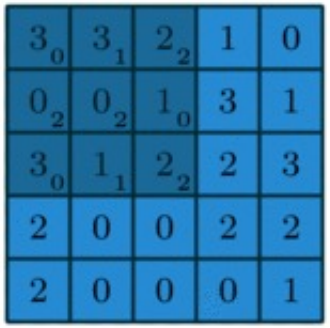
$$o = (i - k) + 2 \times p + 1; p = k - 1$$

https://arxiv.org/abs/1603.07285

Convolution in Neural Networks

- x is a 3×3 chunk (dark area) of the image (blue array)
- Kernel is the subscripts in the dark area
- A neuron is parameterized with the kernel
  - 3×3 weight matrix $\mathbf{w}$

Image: im, dimensions 5x5
Kernel: k, dimensions 3x3



$$(k \star im)(x,y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n,m).\, im(x+n-1, y+m-1)$$

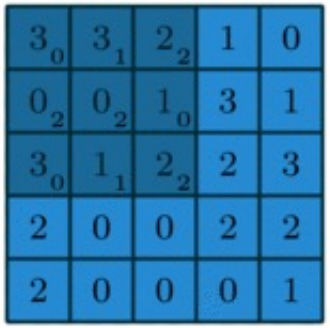
https://arxiv.org/abs/1603.07285

Convolution in Neural Networks

- x is a 3×3 chunk (dark area) of the image (blue array)
- Kernel is the subscripts in the dark area
- A neuron is parameterized with the kernel
  - 3×3 weight matrix $\mathbf{w}$
- The activation obtained by sliding the kernel window to compute:

$$z(x) = relu(\mathbf{w}^T x + b)$$

Image: im, dimensions 5x5
Kernel: k, dimensions 3x3



$\mathbf{w}^T x$

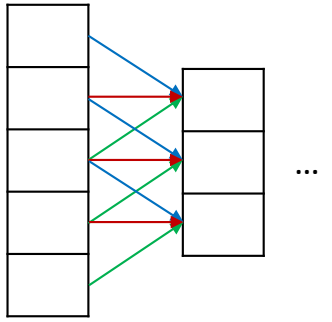
$$(k \star im)(x,y) = \sum_{n=0}^{2} \sum_{m=0}^{2} k(n,m).\, im(x+n-1, y+m-1)$$

https://arxiv.org/abs/1603.07285

Motivations

- Local connectivity
  - A neuron depends only on a few local input neurons
  - Translation invariance
- Comparison to Fully connected
  - Parameter sharing, reduce overfitting
  - Make use of spatial structure: **strong prior** for vision!

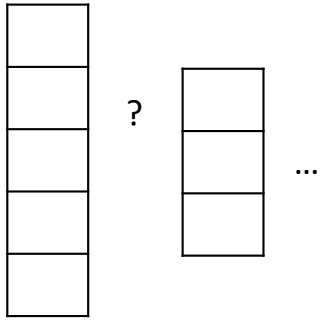
Kernel dimensions + 1



Convolutional layer

#current layer neurons x #previous layer neurons + #current layer neurons

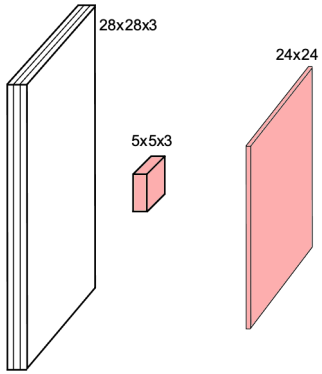
1D Convolution View: How many parameters do we need?

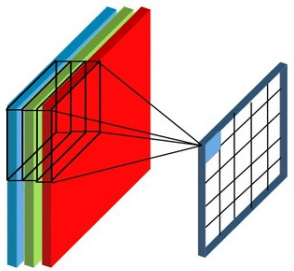


Fully connected (dense layer)

Multiple Channels

- Colored image = tensor of shape (height, width, channels)
- Convolutions are usually computed for each channel separately and summed:

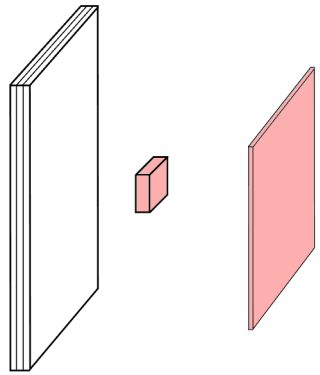




$$\left(k \star im^{color}\right) = \sum_{c=0}^{2} k^c \star im^c$$

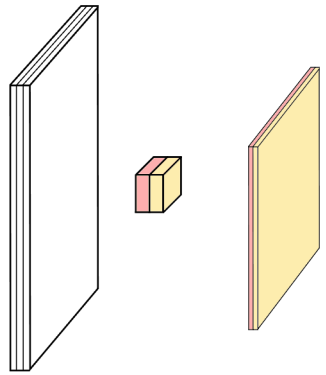
Multiple Convolutions

- Multiple kernel sizes aka receptive fields (usually 1, 3, 5, 7, 11)



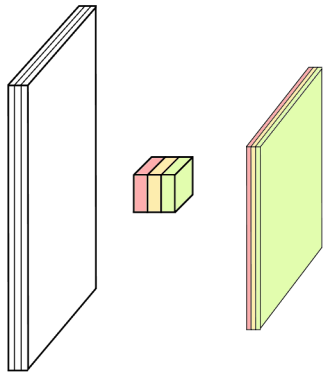
Multiple Convolutions

- Multiple kernel sizes aka receptive fields (usually 1, 3, 5, 7, 11)



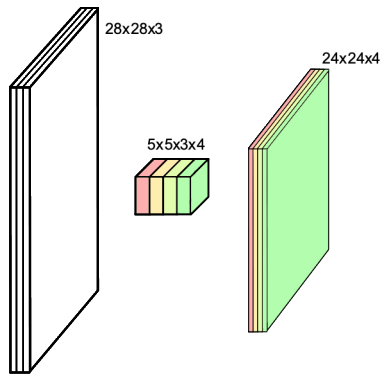
Multiple Convolutions

- Multiple kernel sizes aka receptive fields (usually 1, 3, 5, 7, 11)



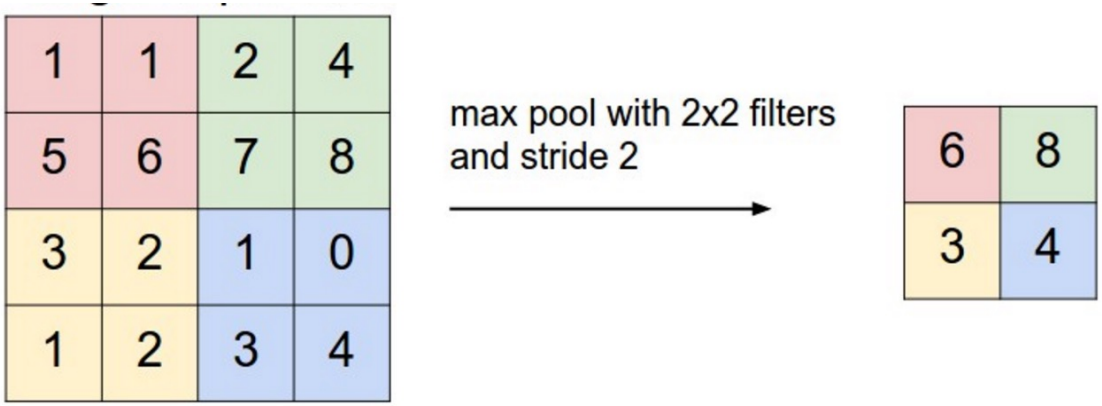
Multiple Convolutions

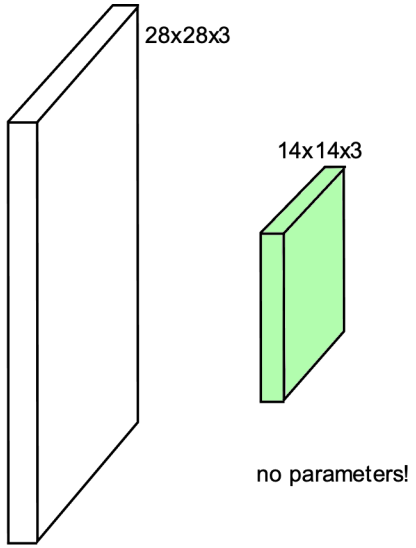
- Multiple kernel sizes aka receptive fields (usually 1, 3, 5, 7, 11)



Pooling

- Spatial dimension reduction
- Local invariance
- No parameters: max or average of 2x2 units





http://cs231n.github.io/convolutional-networks

Put it all together

Input

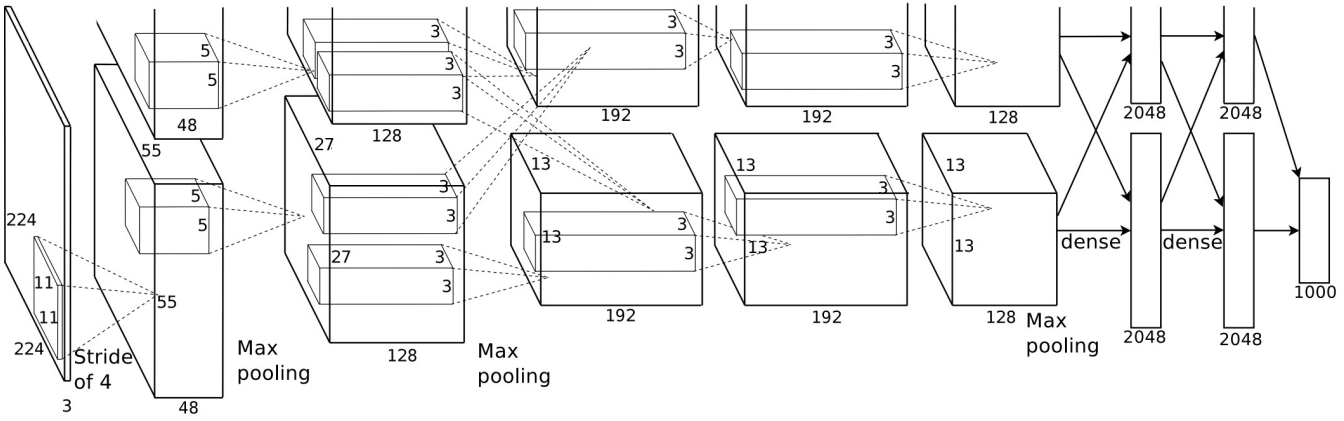
Conv blocks

- Convolution + activation (relu)
- Convolution + activation (relu)
- ...
- Maxpooling 2x2

Output

- Fully connected layers
- Softmax

AlexNet

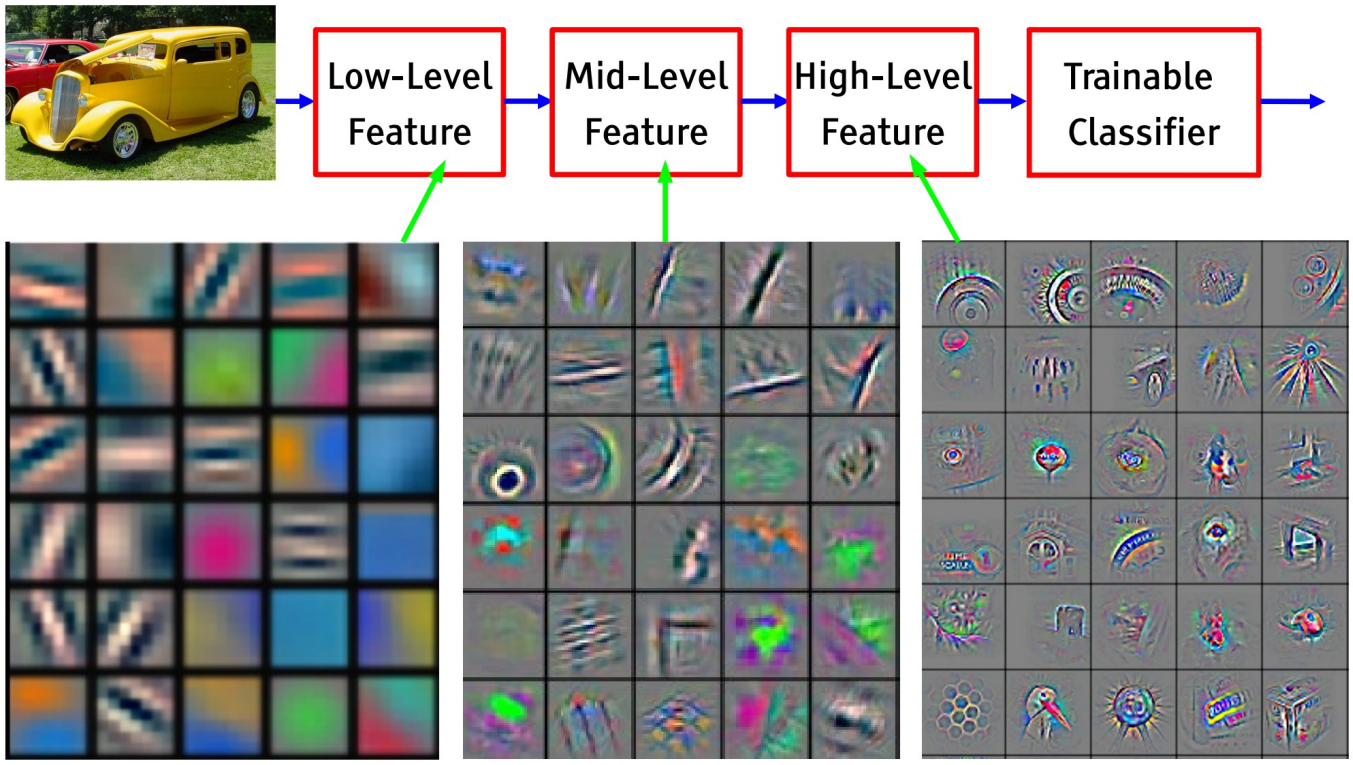


- First conv layer: kernel 11x11x3x96 stride 4
  - Kernel shape: (11,11,3,96)
  - Output shape: (55,55,96)
  - Number of parameters: 34,944
  - Equivalent MLP parameters: 43.7 x 1e9

```
INPUT:     [227x227x3]
CONV1:     [55x55x96]   96 11x11 filters at stride 4, pad 0
MAX POOL1: [27x27x96]      3x3   filters at stride 2
CONV2:     [27x27x256] 256 5x5   filters at stride 1, pad 2
MAX POOL2: [13x13x256]     3x3   filters at stride 2
CONV3:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV4:     [13x13x384] 384 3x3   filters at stride 1, pad 1
CONV5:     [13x13x256] 256 3x3   filters at stride 1, pad 1
MAX POOL3: [6x6x256]       3x3   filters at stride 2
FC6:       [4096]      4096 neurons
FC7:       [4096]      4096 neurons
FC8:       [1000]      1000 neurons (softmax logits)
```


Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

Hierarchical Representation



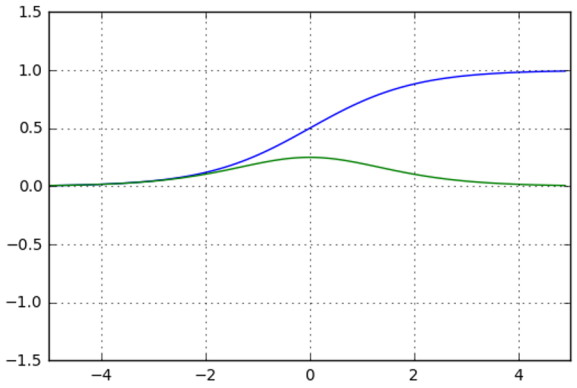
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

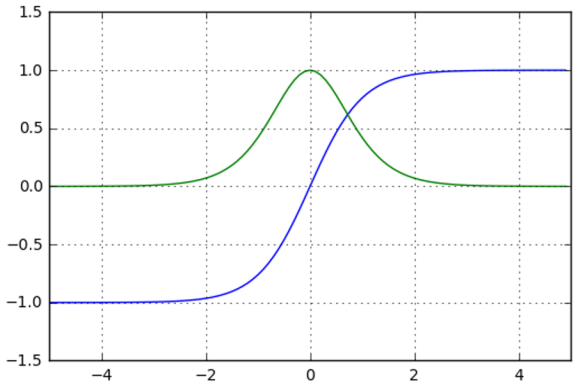
AlexNet

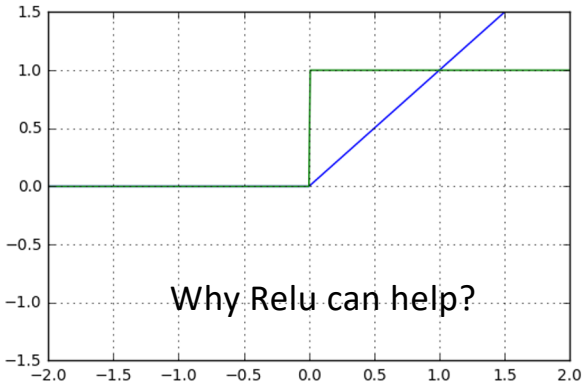
- One of the first deep convolutional networks to achieve considerable accuracy on the 2012 ImageNet LSVRC-2012 challenge
  - 84.7% vs 73.8% (second best)
- Data augmentation and dropout layers to reduce overfitting
- Use ReLu to avoid the vanishing gradient problem

Vanishing gradient problem

- Recall that the optimization process is based on backpropagation of gradient computed with the loss function
- When the gradients are already small, the chain rule makes the gradients of the earlier layers even smaller (e.g., decreases exponentially the number of layers)
- Earlier layers are difficult to train – weights do not change due to the small gradients





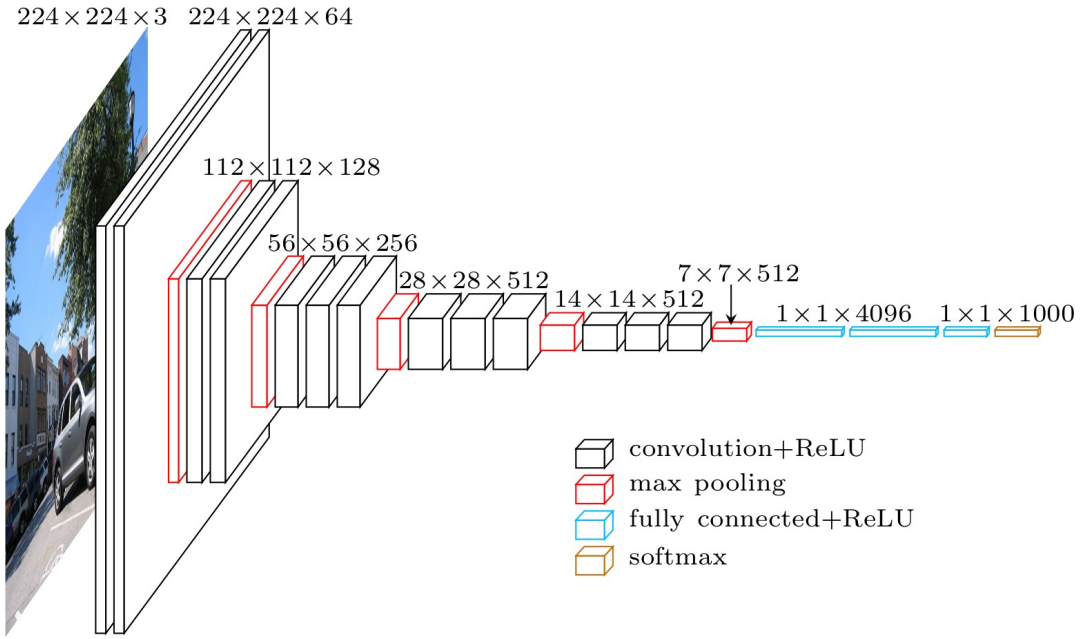


$$\mathrm{sigm}(x) = \frac{1}{1+e^{-x}} \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \qquad \mathrm{relu}(x) = \max(0,x)$$

$$\mathrm{sigm}'(x) = \mathrm{sigm}(x)(1-\mathrm{sigm}(x)) \qquad \tanh'(x) = 1-\tanh(x)^2 \qquad \mathrm{relu}'(x) = 1_{x>0}$$

Let's go deeper: VGG-16

- More layers than AlexNet
  - 138m vs 62m parameters
- Effectively use fixed size, small kernels to replace larger, variable size kernels
- Building blocks:
  - Conv, Conv, Max Pooling or
  - Conv, Conv, Conv, Max Pooling

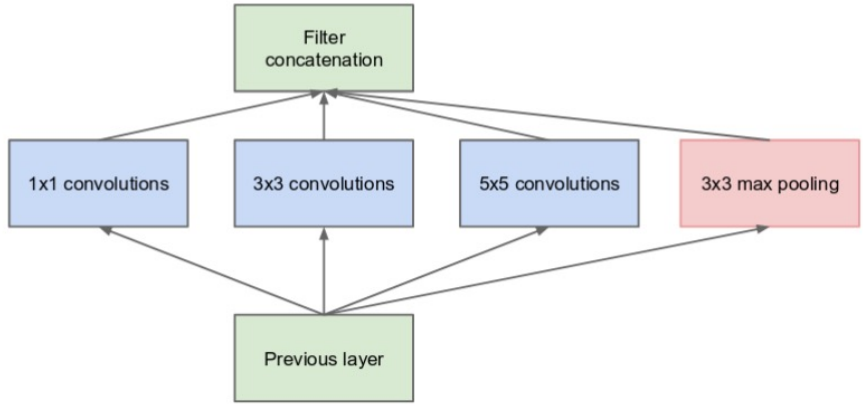


138 million parameters

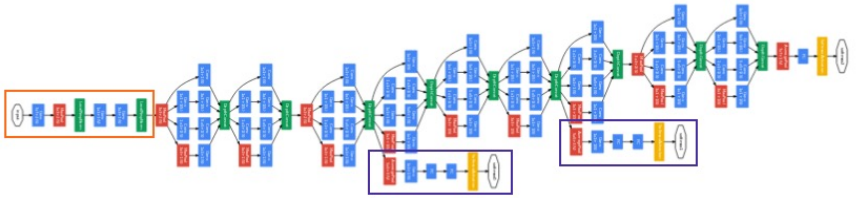
Simonyan, Karen, and Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014)

Let's go wider: GoogleNet Inception

- Capture global (semantic) information with larger kernel
- Capture local (spatial) information with smaller kernel
- Use multiple variable size kernels in the same layer
  - Fewer layers, easier to train
  - Still capture both global and local information

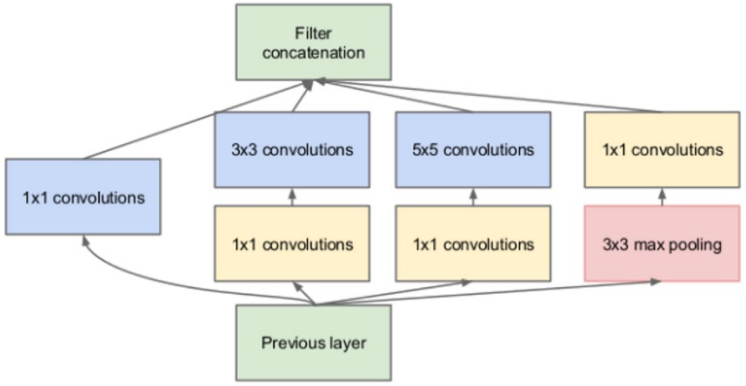


(a) Inception module, naïve version



1D convolution in GoogleNet

- Reduce the number of channels
  - Summed
- Selectively squeeze information from multiple channels in the previous layer to 1 channel
  - Like dimension reductions
  - Allow efficient use of multiple kernels



https://andre-ye.medium.com/the-clever-trick-behind-googles-inception-the-1-1-convolution-58815b20113

Can we go deeper again? ResNet

- More layers, deeper models
  - 34, 50, 101, 152 layers
- Skipped connections pass information across layers
  - Mitigate (but not solve) the vanishing gradient problem

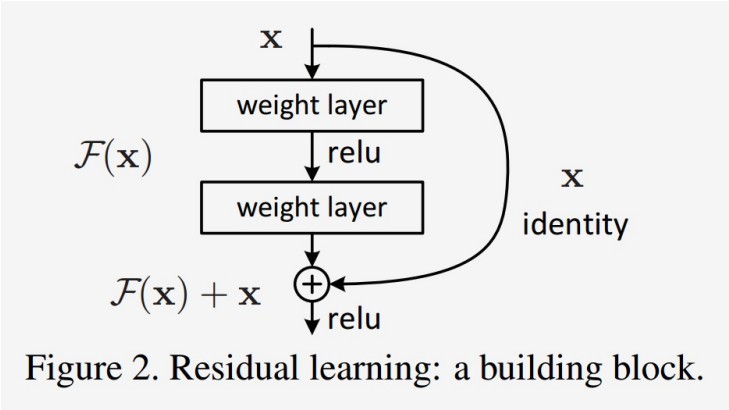
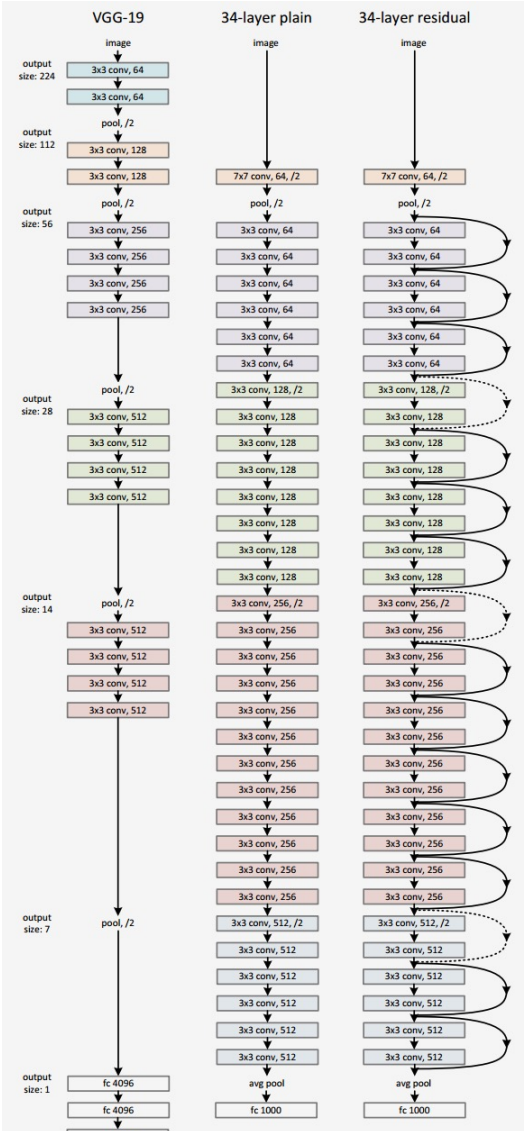


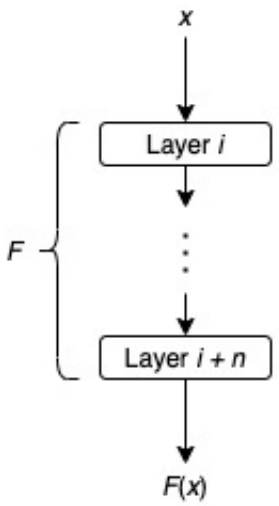
Figure 2. Residual learning: a building block.



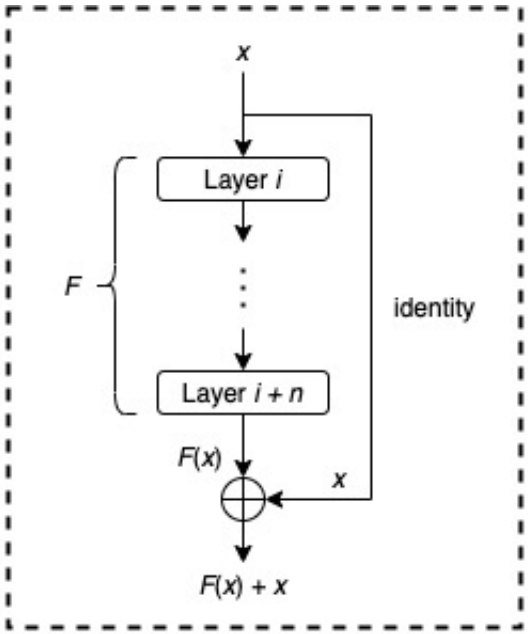
He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.

Residual Connections

- Also called the skipped or skip connections
- Used in many different types of neural networks to facilitate training (e.g., mitigate the vanish gradient problem)



Traditional Feedforward without Residual Connection

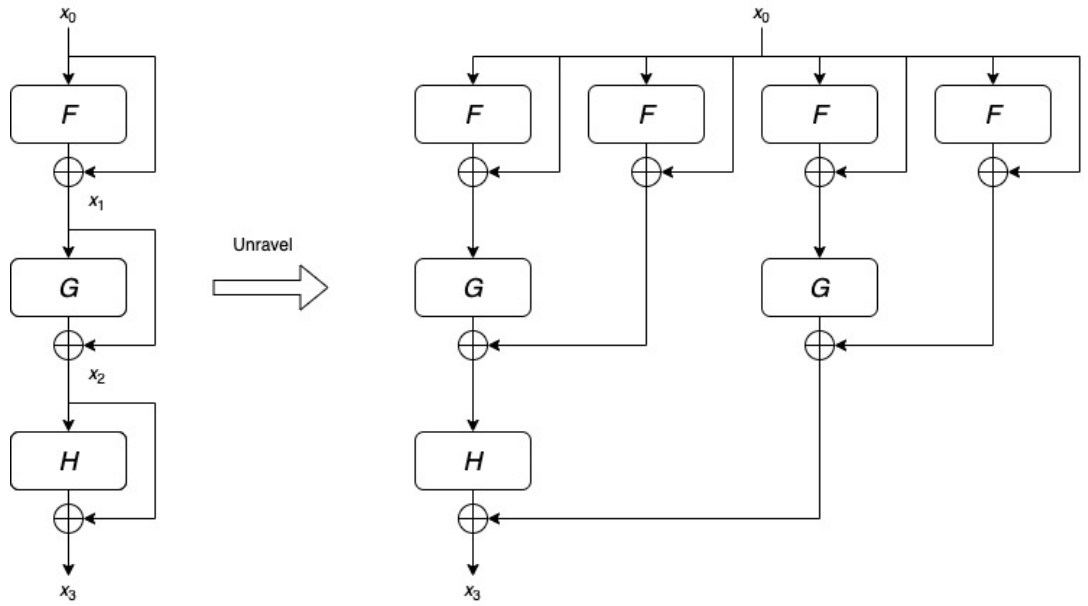


With Residual Connection

https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55

Residual Connections

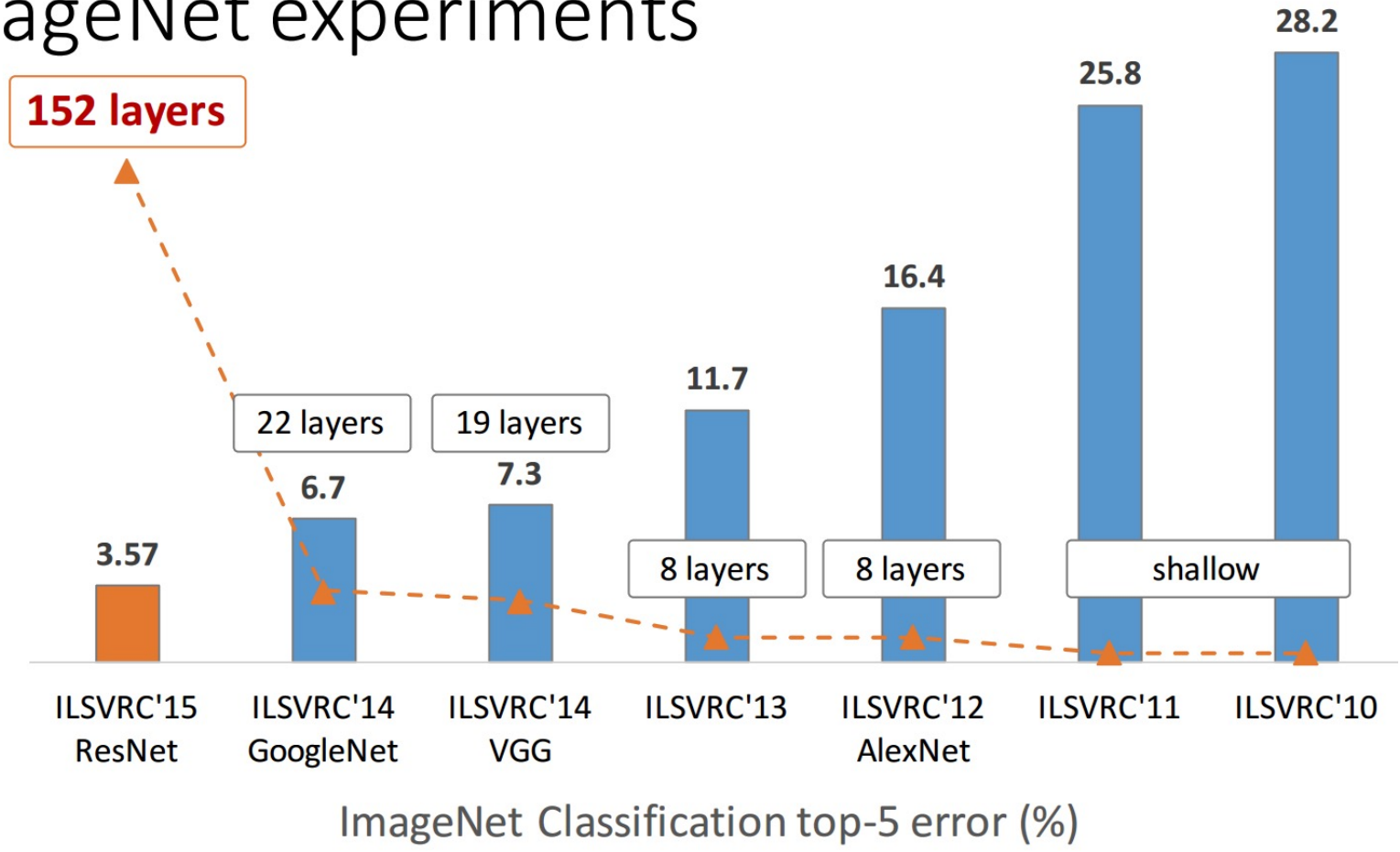
- As ensembles of shallow neural networks
- Empirically, residual connections often help generate better models and results, e.g., see DenseNet



$$
\begin{aligned}
x_3 &= H(x_2) + x_2 \\
&= H(G(x_1) + x_1) + G(x_1) + x_1 \\
&= H(G(F(x_0) + x_0) + F(x_0) + x_0) + G(F(x_0) + x_0) + F(x_0) + x_0
\end{aligned}
$$

https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55

ImageNet experiments



from Kaiming He slides "Deep residual learning for image recognition." ICML. 2016.

AlexNet, VGG, Inception, ResNet

- FLOP = floating point operations (required for a forward pass)
  - e.g., training AlexNet would take similar time as training Inception

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

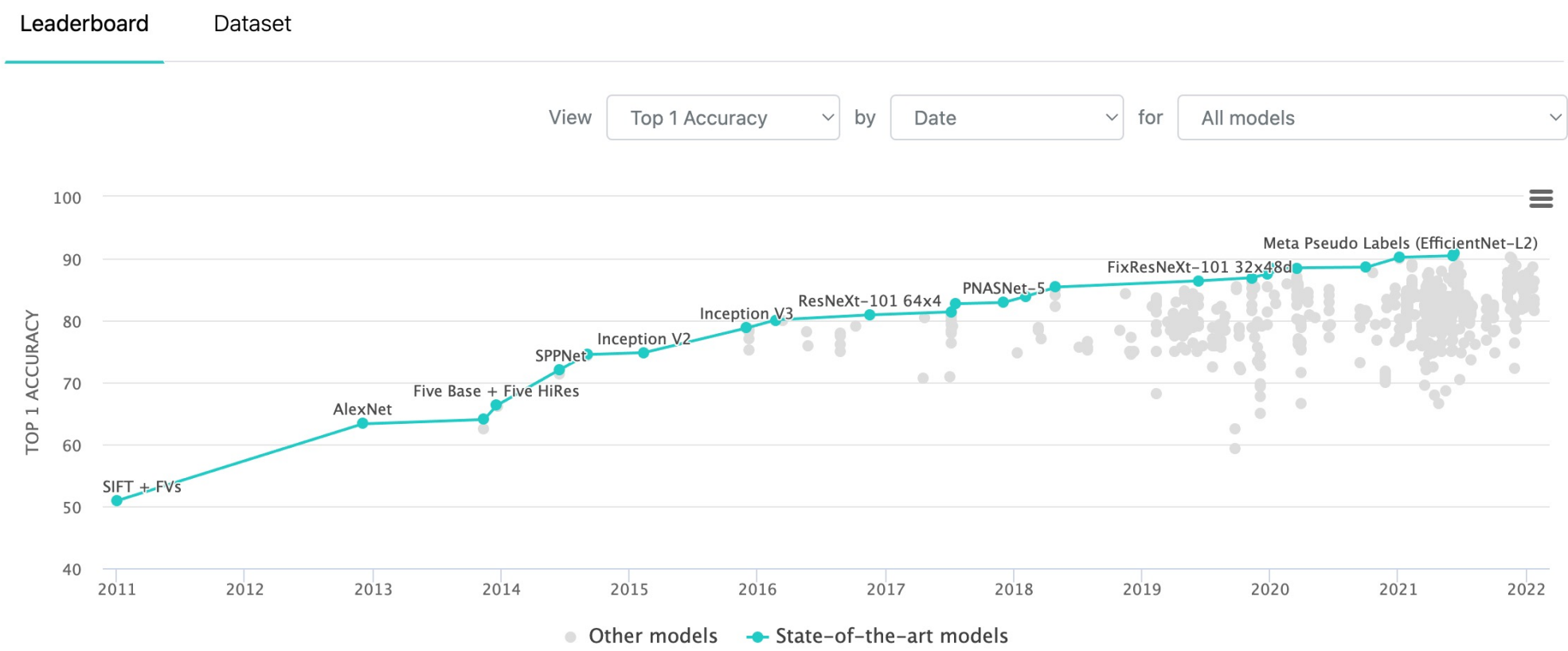
https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

State of the Art

Method	# Params	Extra Data	ImageNet Top-1	ImageNet Top-5	ImageNet-ReaL [6] Precision@1
ResNet-50 [24]	26M	–	76.0	93.0	82.94
ResNet-152 [24]	60M	–	77.8	93.8	84.79
DenseNet-264 [28]	34M	–	77.9	93.9	–
Inception-v3 [62]	24M	–	78.8	94.4	83.58
Xception [11]	23M	–	79.0	94.5	–
Inception-v4 [61]	48M	–	80.0	95.0	–
Inception-resnet-v2 [61]	56M	–	80.1	95.1	–
ResNeXt-101 [78]	84M	–	80.9	95.6	85.18
PolyNet [87]	92M	–	81.3	95.8	–
SENet [27]	146M	–	82.7	96.2	–
NASNet-A [90]	89M	–	82.7	96.2	82.56
AmoebaNet-A [52]	87M	–	82.8	96.1	–
PNASNet [39]	86M	–	82.9	96.2	–
AmoebaNet-C + AutoAugment [12]	155M	–	83.5	96.5	–
GPipe [29]	557M	–	84.3	97.0	–
EfficientNet-B7 [63]	66M	–	85.0	97.2	–
EfficientNet-B7 + FixRes [70]	66M	–	85.3	97.4	–
EfficientNet-L2 [63]	480M	–	85.5	97.5	–
ResNet-50 Billion-scale SSL [79]	26M	3.5B labeled Instagram	81.2	96.0	–
ResNeXt-101 Billion-scale SSL [79]	193M	3.5B labeled Instagram	84.8	–	–
ResNeXt-101 WSL [42]	829M	3.5B labeled Instagram	85.4	97.6	88.19
FixRes ResNeXt-101 WSL [69]	829M	3.5B labeled Instagram	86.4	98.0	89.73
Big Transfer (BiT-L) [33]	928M	300M labeled JFT	87.5	98.5	90.54
Noisy Student (EfficientNet-L2) [77]	480M	300M unlabeled JFT	88.4	98.7	90.55
Noisy Student + FixRes [70]	480M	300M unlabeled JFT	88.5	98.7	–
Vision Transformer (ViT-H) [14]	632M	300M labeled JFT	88.55	–	90.72
EfficientNet-L2-NoisyStudent + SAM [16]	480M	300M unlabeled JFT	88.6	98.6	–
Meta Pseudo Labels (EfficientNet-B6-Wide)	390M	300M unlabeled JFT	90.0	98.7	**91.12**
Meta Pseudo Labels (EfficientNet-L2)	480M	300M unlabeled JFT	**90.2**	**98.8**	91.02

Meta Pseudo Labels, Hieu Pham et al. (Jan 2021)

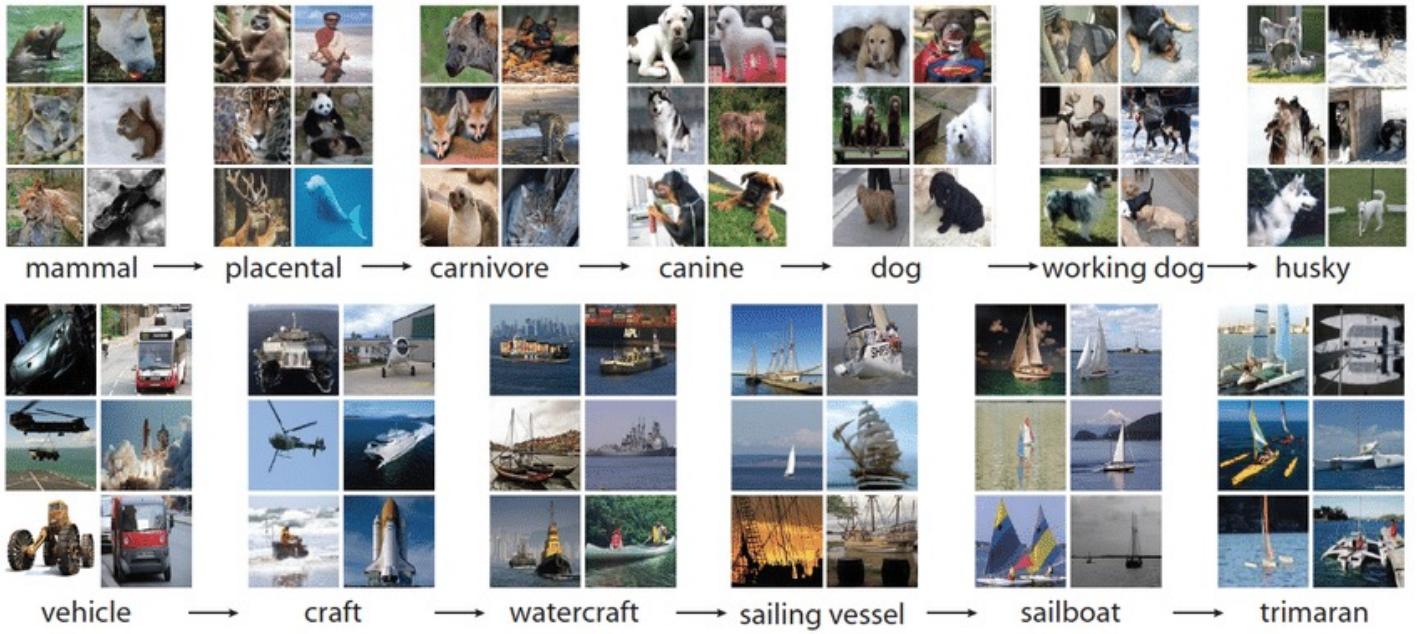
Image Classification on ImageNet



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

Pre-trained Models

- Training a model on ImageNet (14m+ annotated images) from scratch takes **days or weeks**.
- Many models trained on ImageNet and their weights are publicly available



ImageNet & Pre-trained Models

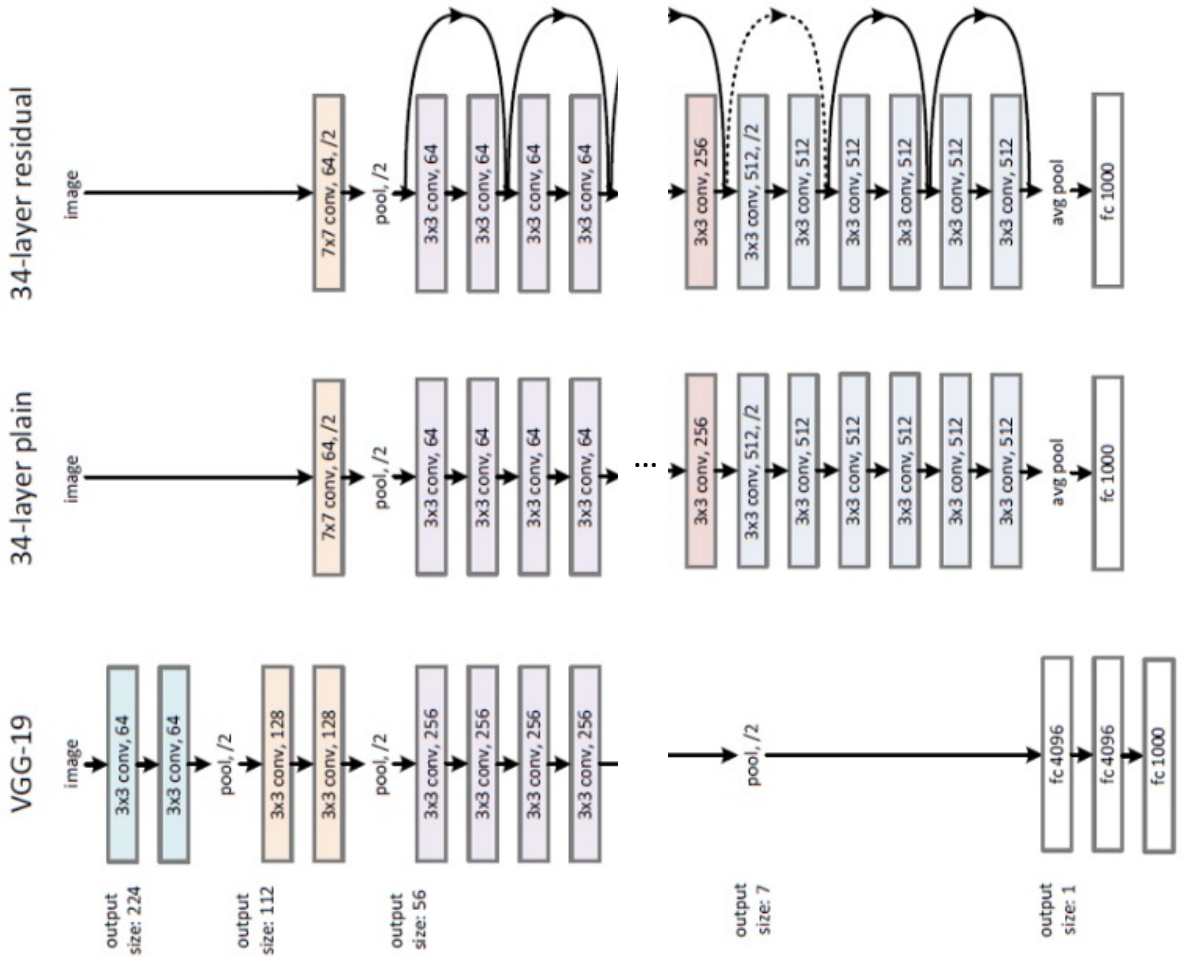
- Many models trained on ImageNet and their weights are publicly available
- Transfer learning
  - Use pre-trained weights, remove last layers to compute representations of images
  - Train a classification model from these features on a new classification task
  - The network is used as a generic feature extractor
  - Better than handcrafted feature extraction on natural images

Fine-tuning Pre-trained Models

- Fine-tuning
  - Retraining the (some) parameters of the network (given enough data)
  - Truncate the last layer(s) of the pre-trained network
  - Freeze the remaining layers weights
  - Add a (linear) classifier on top and train it for a few epochs
  - Then fine-tune the whole network or the few deepest layers
  - Use a smaller learning rate when fine tuning

PyTorch Tutorial

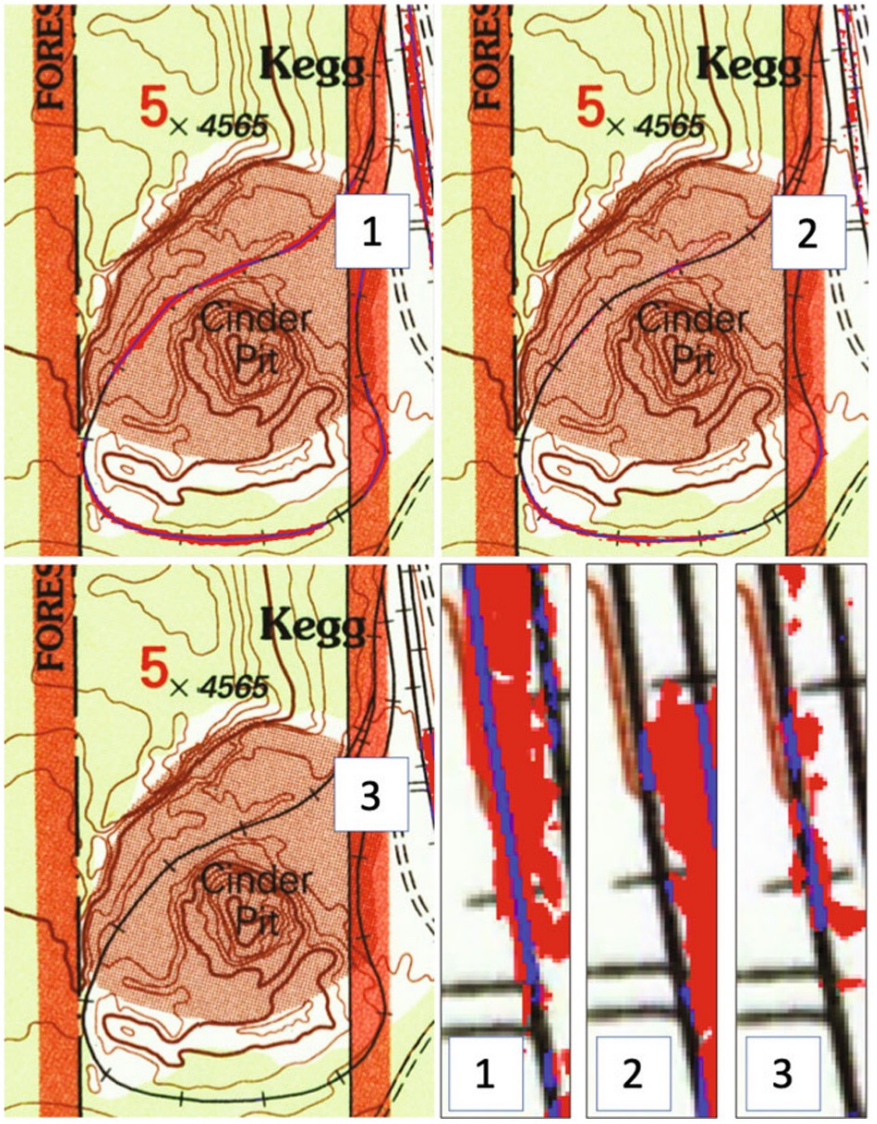
- **Finetuning the convnet**: initialize the network with a pretrained network. Rest of the training looks as usual.
- **ConvNet as fixed feature extractor**: freeze the weights for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.



https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.ht
https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8

Fine-tuning Example

- Semantic segmentation results (overlaying with the test map) using pre-trained PSPNet with large size training data and trained from the shallow (top left), middle (top right), and deep layers (bottom left) (FP in red and TP in blue)



Bias and Ethical Concerns

- Neural networks are great approximators
  - You get what you trained with
- Training data can have bias
- Model can have bias
- Applications can discriminate against certain populations
  - “In July 2020, the National Institute of Standards and Technology (NIST) conducted independent assessments to confirm these results. It reported that facial recognition technologies for 189 algorithms showed racial bias toward women of color. NIST also concluded that even the best facial recognition algorithms studied couldn’t correctly identify a mask-wearing person nearly 50% of the time.”

https://learn.g2.com/ethics-of-facial-recognition

CNN Summary

- A powerful neural network architecture for many tasks, especially computer vision
- Reduce the number of parameters (parameter sharing)
- Capture spatial interactions in a neighborhood (i.e., kernel size)
- Capture consistent spatial interactions across the input space (e.g., the input image) – could be a limitation
- Many types of CNN architectures and pretrained models are available
- Models can have bias and ethical concerns should be addressed

Acknowledgements

- Deep learning slides adapted from https://m2dsupsdlclass.github.io/lectures-labs/ by Olivier Grisel and Charles Ollion (CC-By 4.0 license)
- Gil, Yolanda (Ed.) Introduction to Computational Thinking and Data Science. Available from http://www.datascience4all.org

These materials are released under a CC-BY License

https://creativecommons.org/licenses/by/2.0/

You are free to:

Share — copy and redistribute the material in any medium or format

Adapt — remix, transform, and build upon the material

for any purpose, even commercially.

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

Artwork taken from other sources is acknowledged where it appears.

Artwork that is not acknowledged is by the author.

Please credit as: Chiang, Yao-Yi Introduction to Spatial Artificial Intelligence. Available from https://yaoyichi.github.io/spatial-ai.html

If you use an individual slide, please place the following at the bottom: "Credit: https://yaoyichi.github.io/spatial-ai.html

We welcome your feedback and contributions.

Credit: http://www.datascience4all.org/