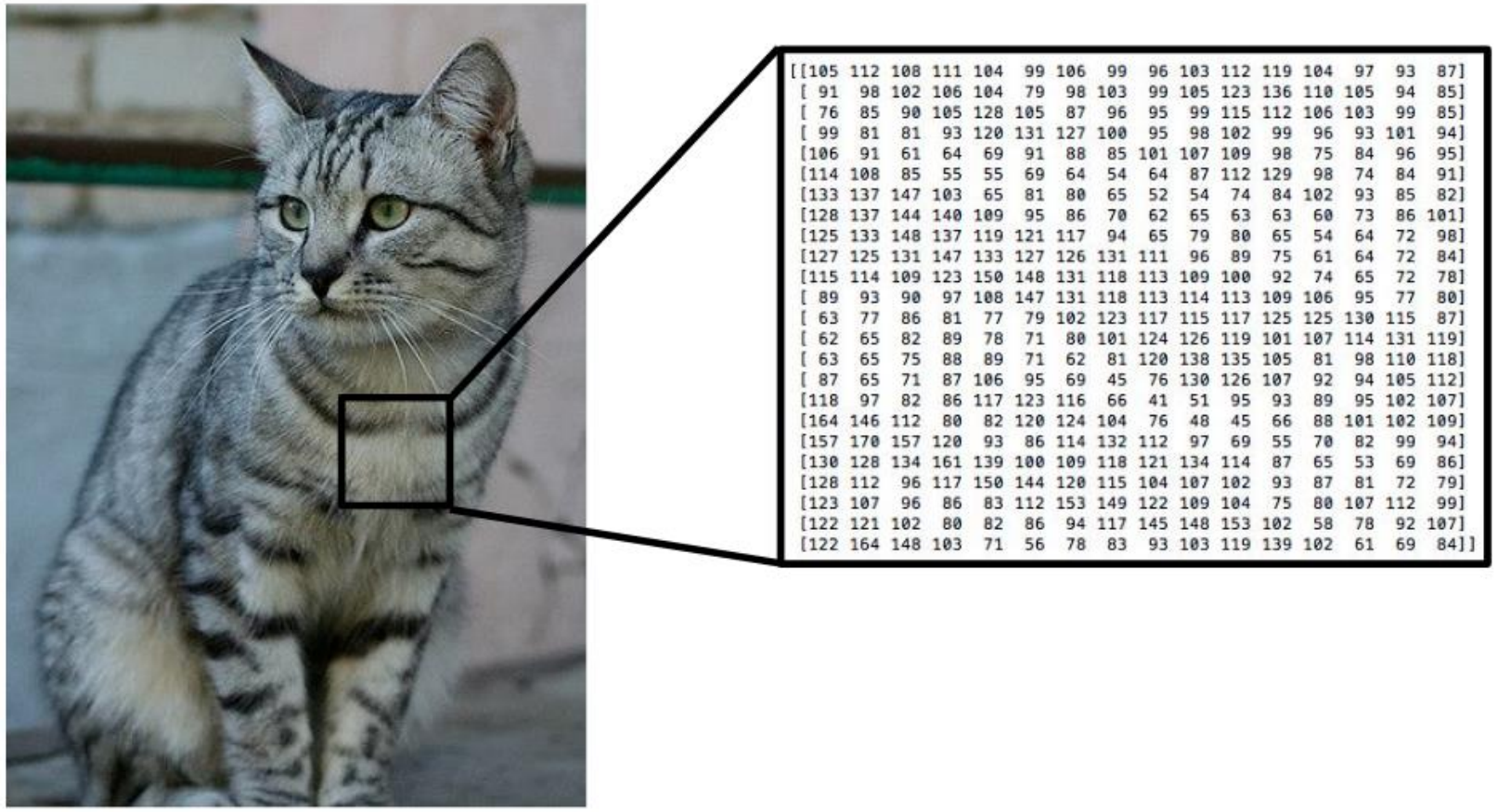
Introduction to Computer Vision

24/09/2019

Computer Vision is Challenging



What we see

What a computer sees

Challenges: Background Clutter

This image is CC0 1.0 public domain

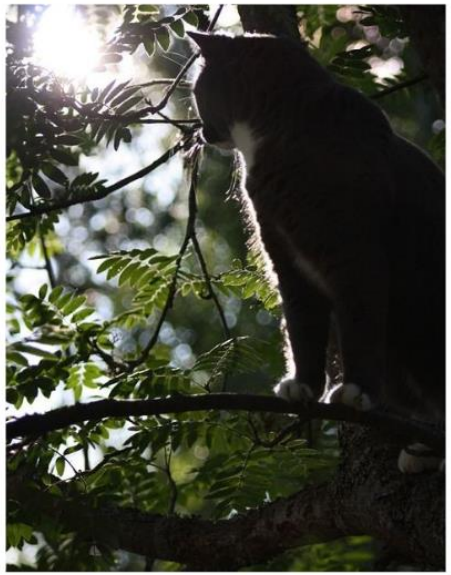
This image is CC0 1.0 public domain

Challenges: Illumination

This image is CC0 1.0 public domain

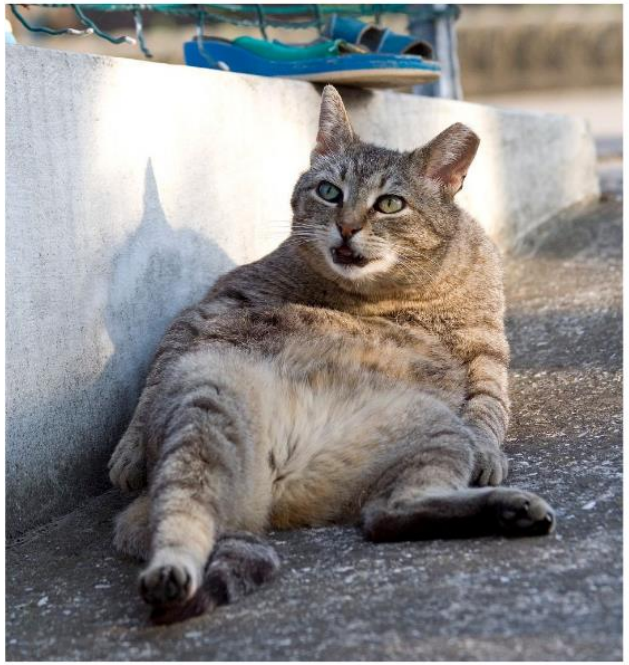
This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

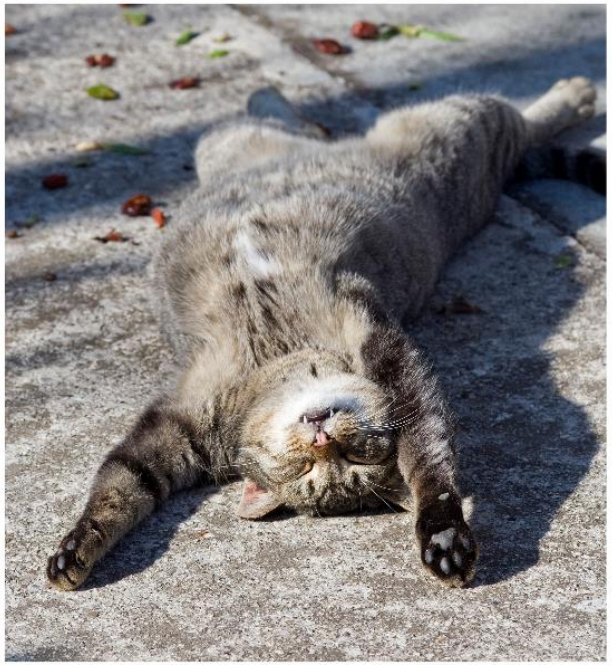


This image is CC0 1.0 public domain

Challenges: Deformation

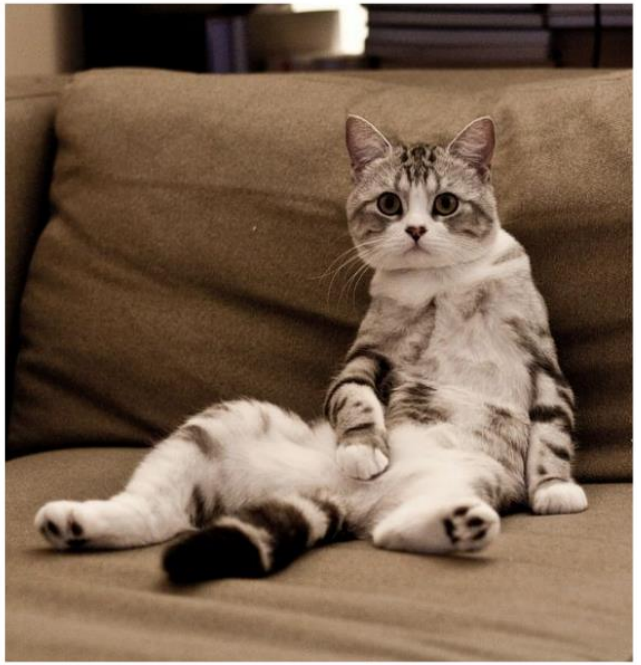


This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by Umberto Salvagnin is licensed under CC-BY 2.0

This image by sare bear is licensed under CC-BY 2.0



This image by Tom Thai is licensed under CC-BY 2.0

Challenges: Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

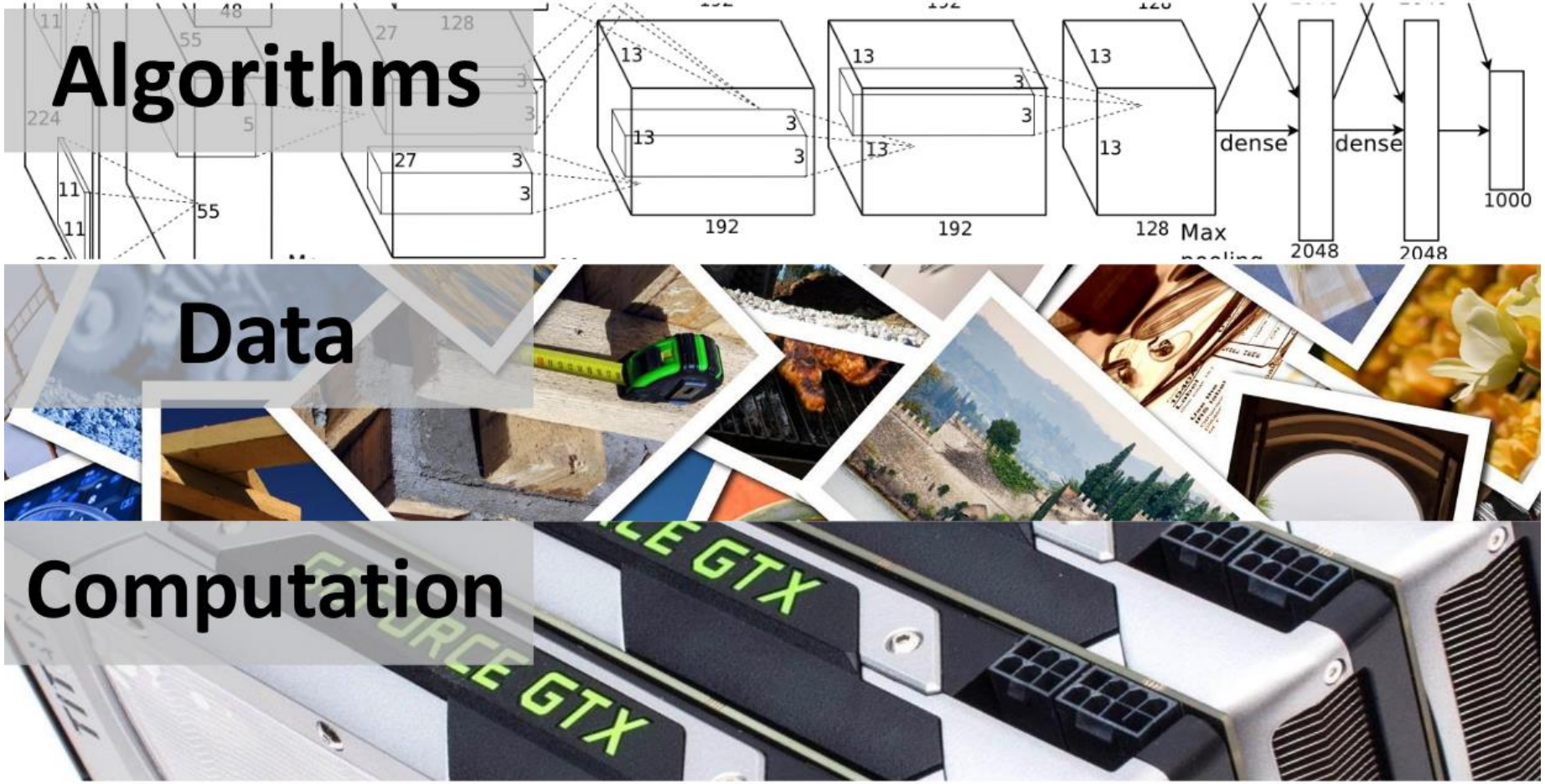
Challenges: Intraclass variation

This image is CC0 1.0 public domain


dog
mug
hat

How do we solve it?

Deep Learning



Deep Convolutional Neural Networks
have become an important tool for Computer Vision.

IMAGENET Large Scale Visual Recognition Challenge

Year 2010

NEC-UIUC

Dense descriptor grid: HOG, LBP

Coding: local coordinate, super-vector

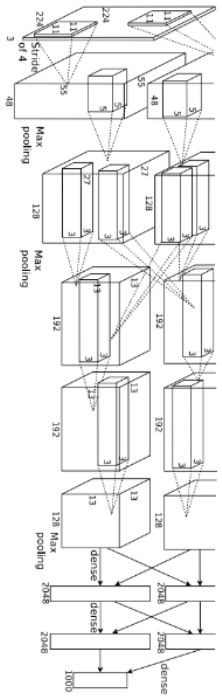
Pooling, SPM

Linear SVM

[Lin CVPR 2011]

Year 2012

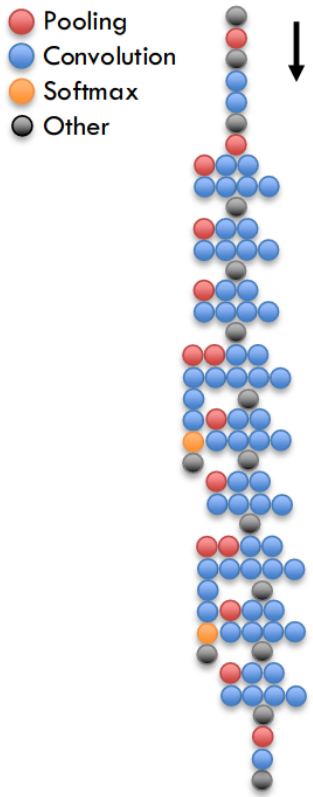
SuperVision



[Krizhevsky NIPS 2012]

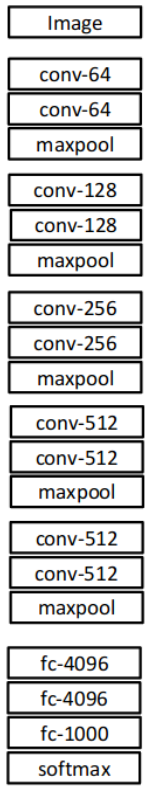
Year 2014

GoogLeNet



[Szegedy arxiv 2014]

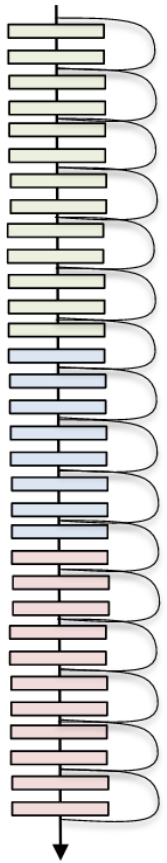
VGG



[Simonyan arxiv 2014]

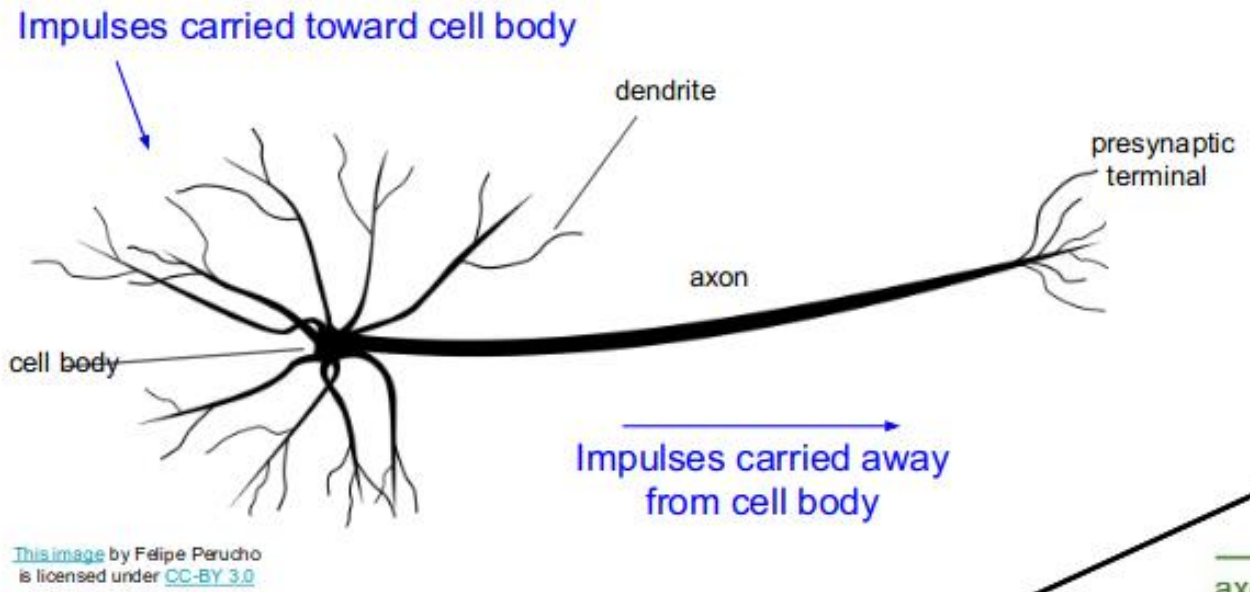
Year 2015

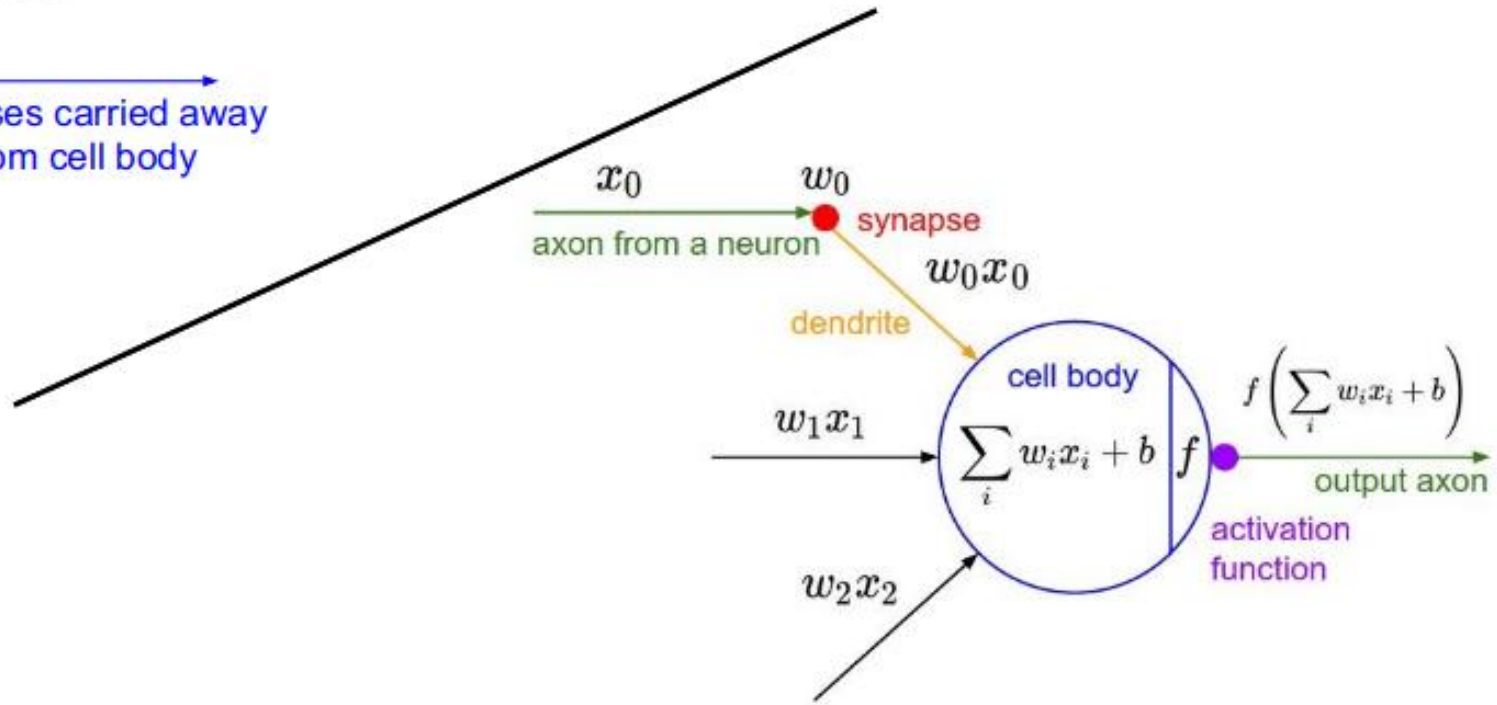
MSRA



[He ICCV 2015]

Neural Networks

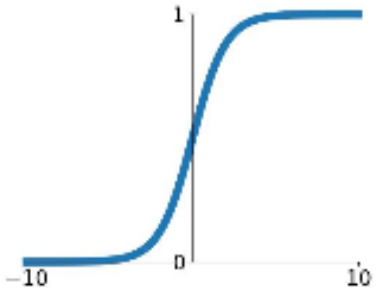




Activation Functions: Non-linear Computations

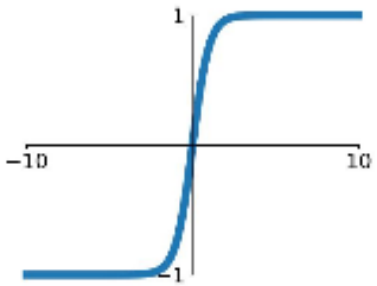
Sigmoid

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



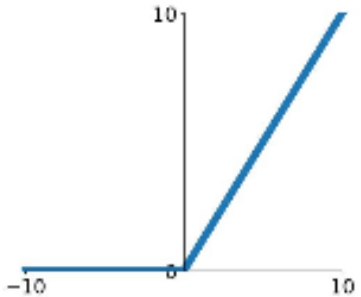
tanh

$$\tanh(x)$$



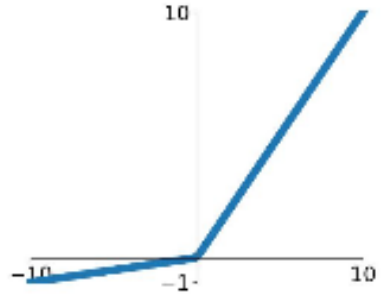
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

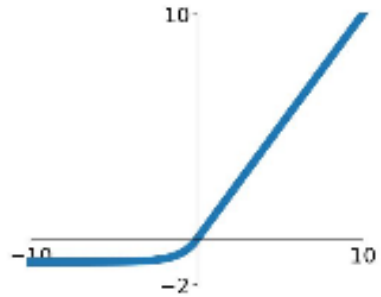


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

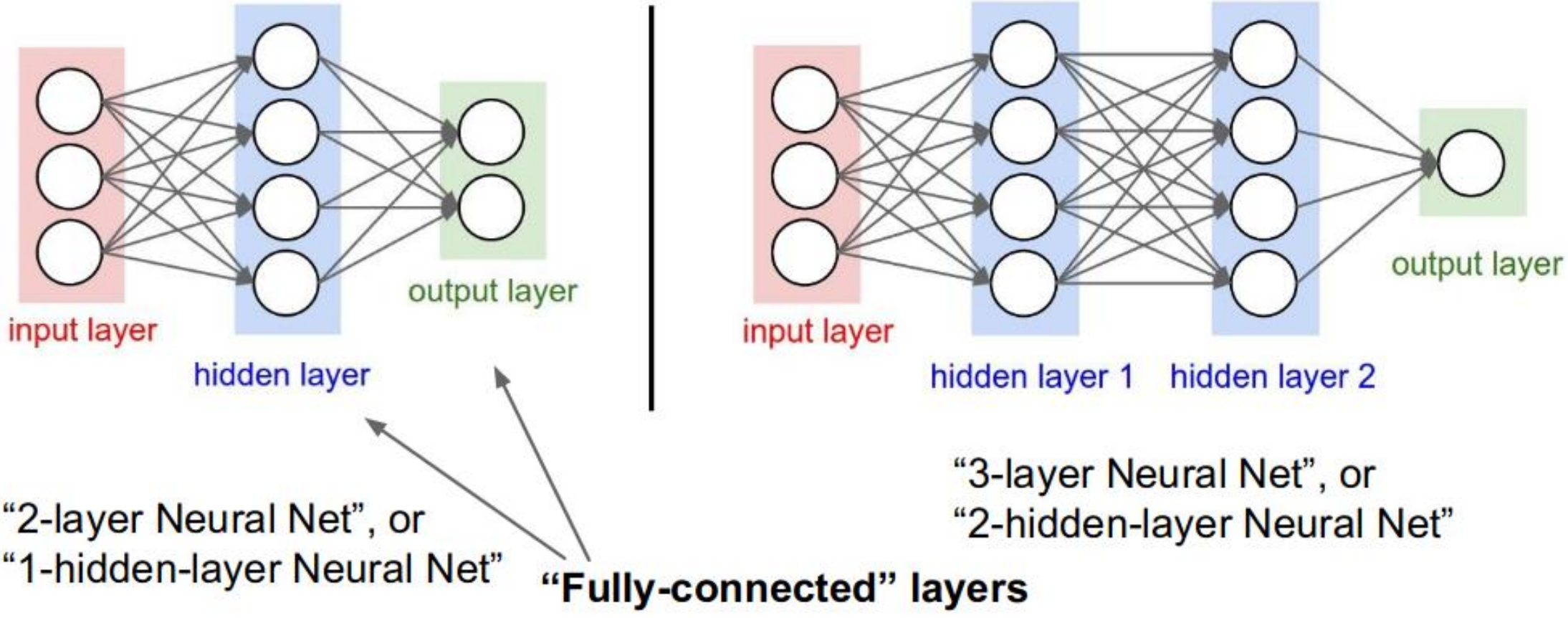
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural Networks

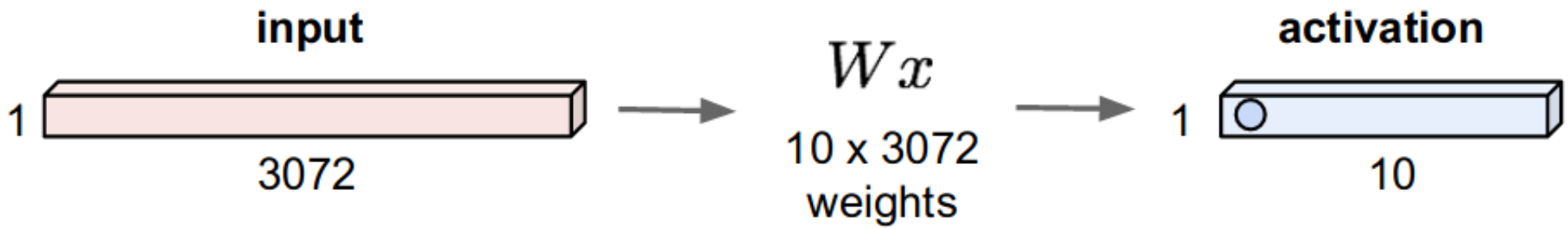
- Linear score function $f = Wx$
- 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
- 3-layer Neural Network $f = W_3 \max(0, W_2 \max(0, W_1 x))$

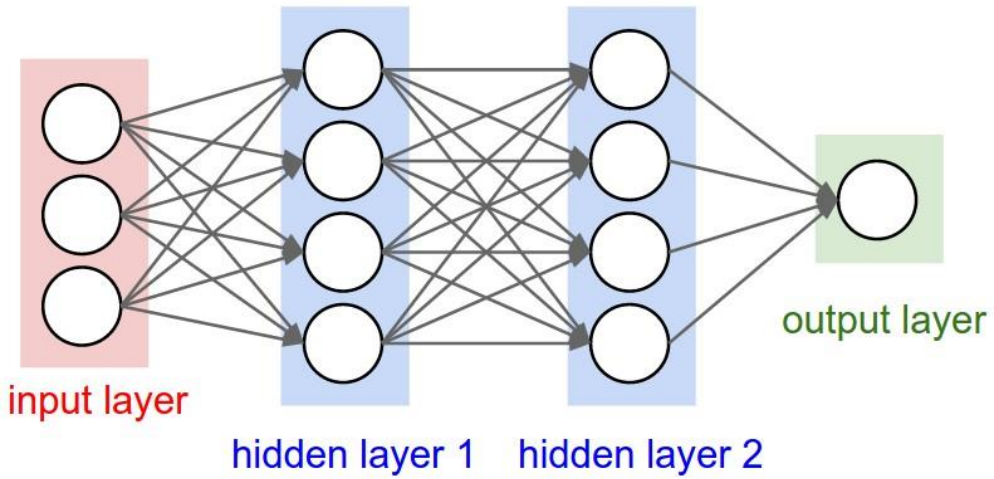
Neural Networks



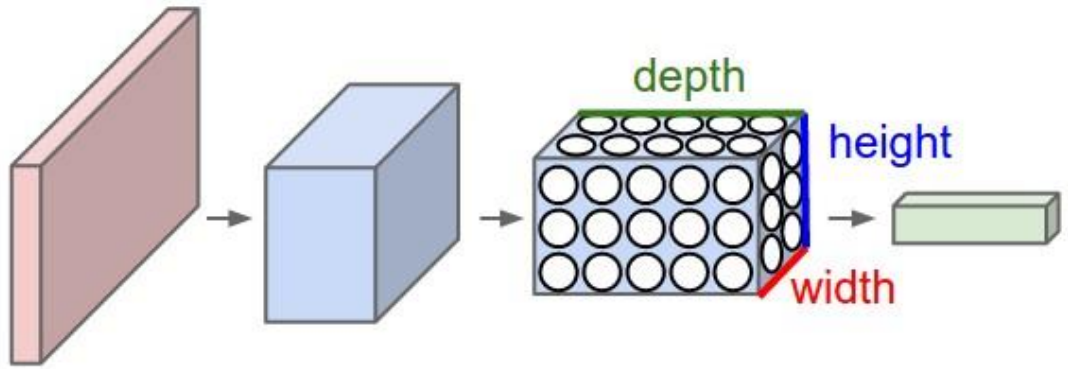
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



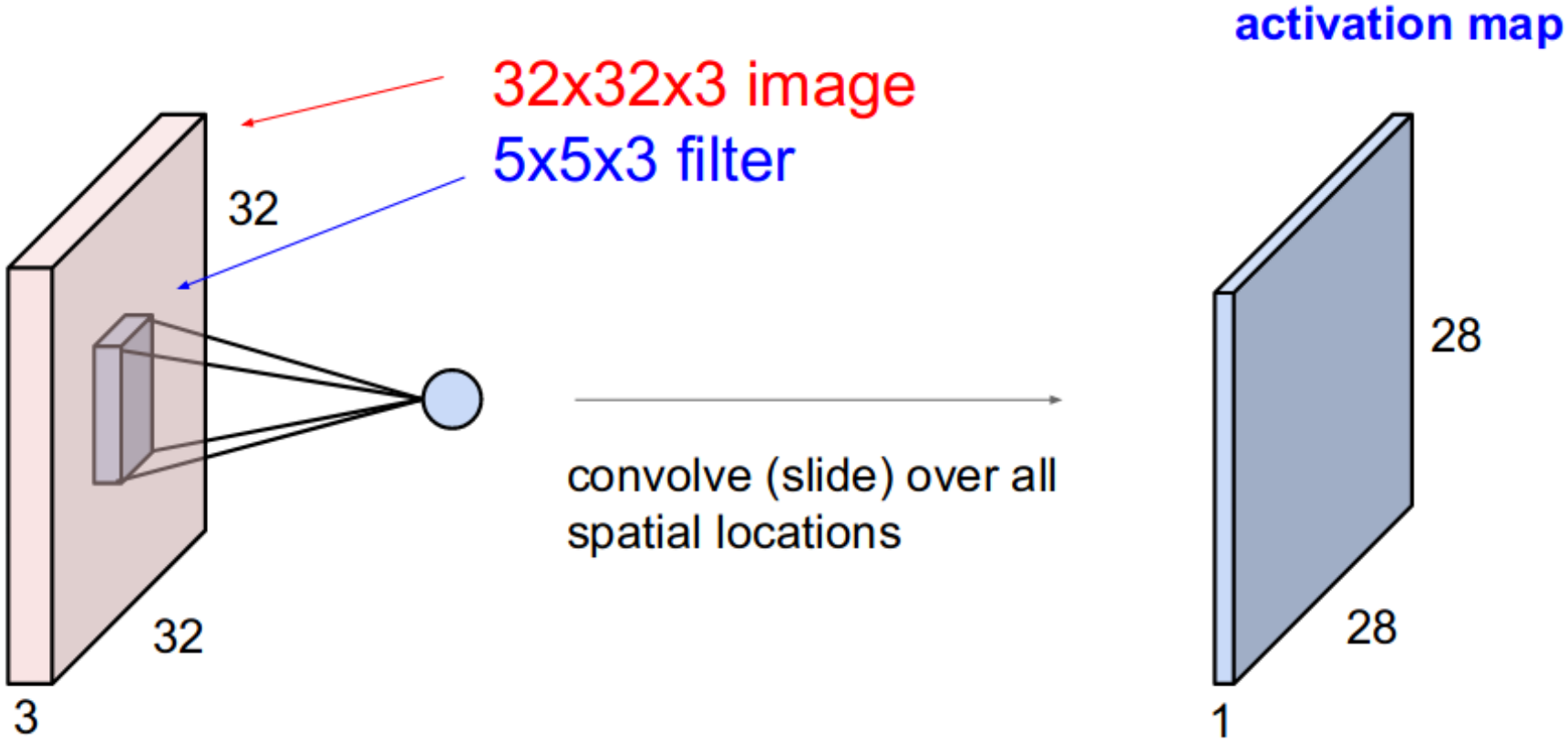


Fully-Connected Neural Network



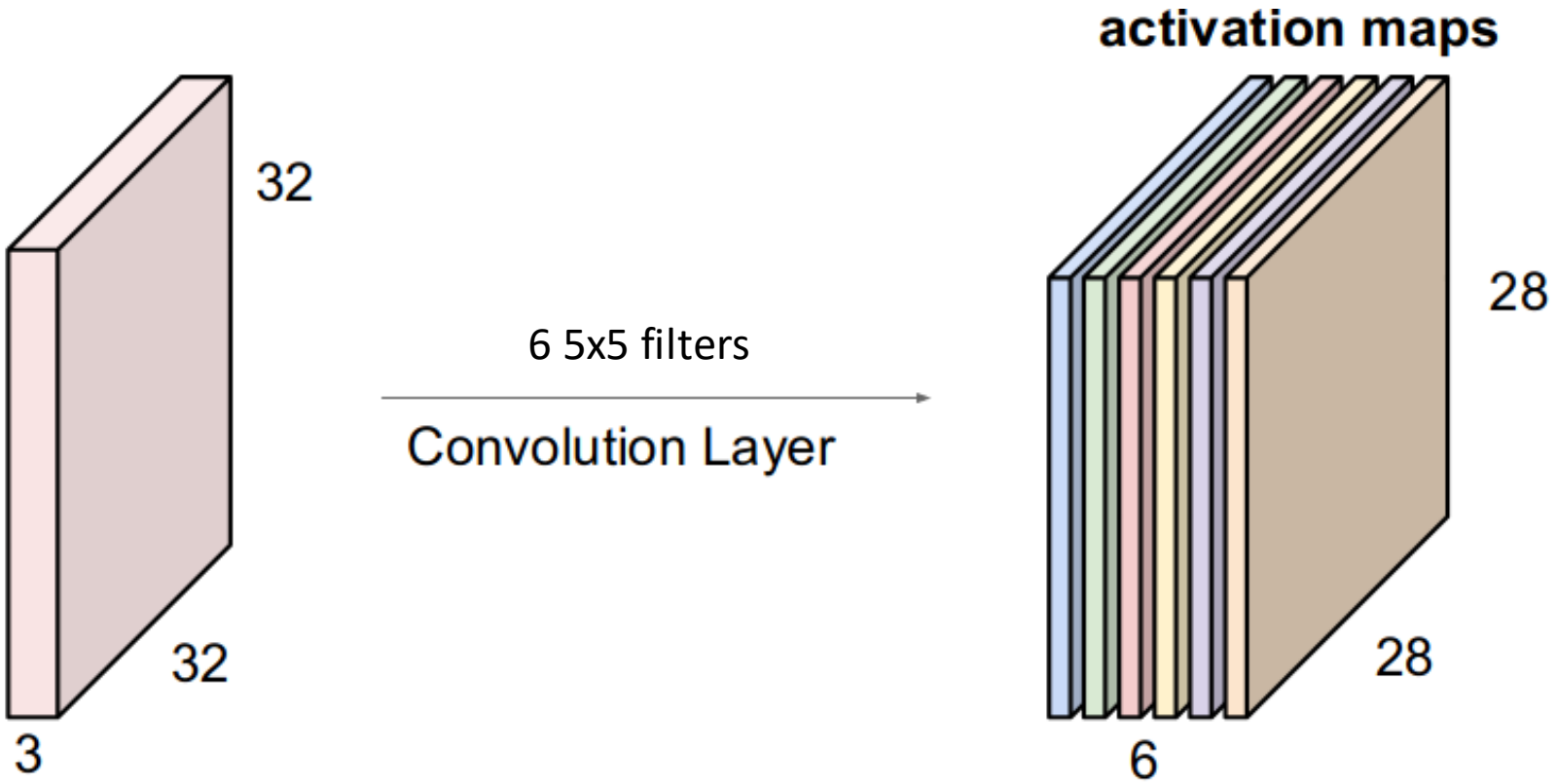
Convolutional Neural Network

Convolution Layer

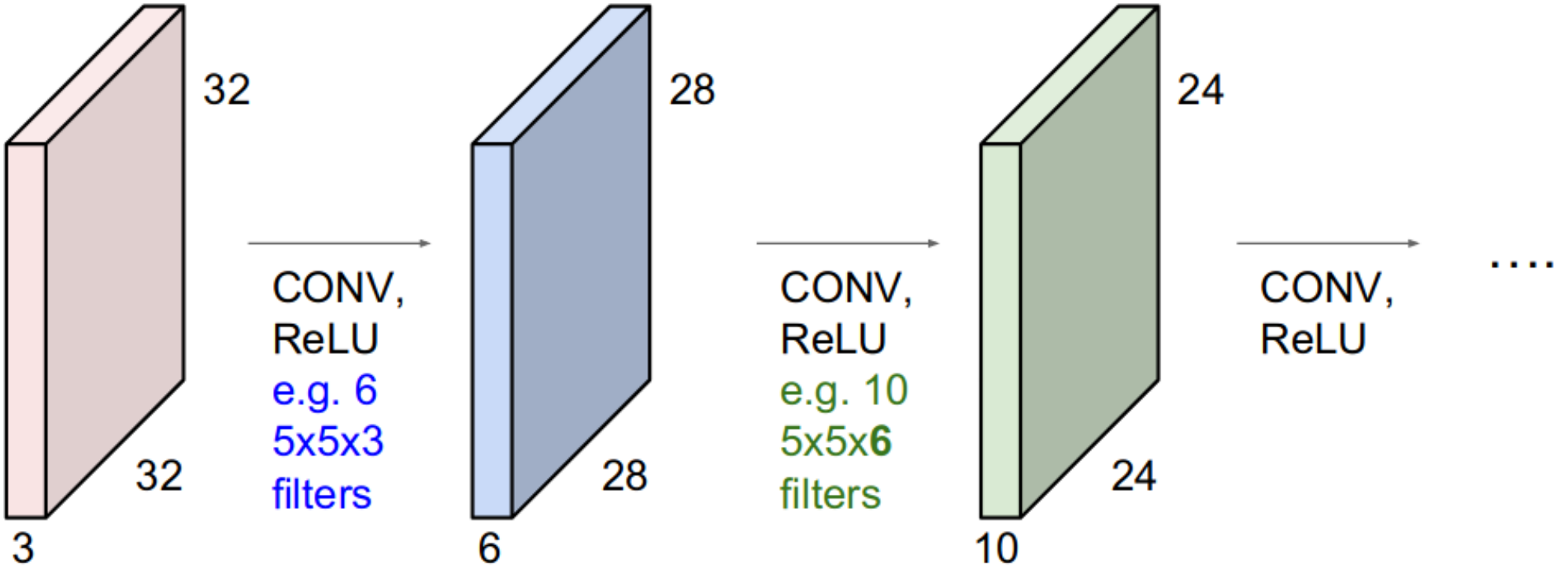


Convolve the filter with the image i.e. “slide over the image spatially, computing **dot products**”

Convolution Layer

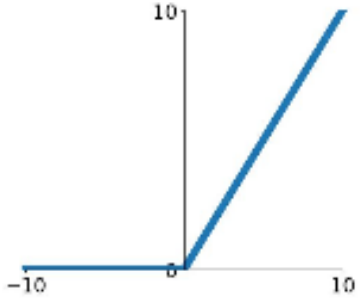


Convolution Layers



ReLU

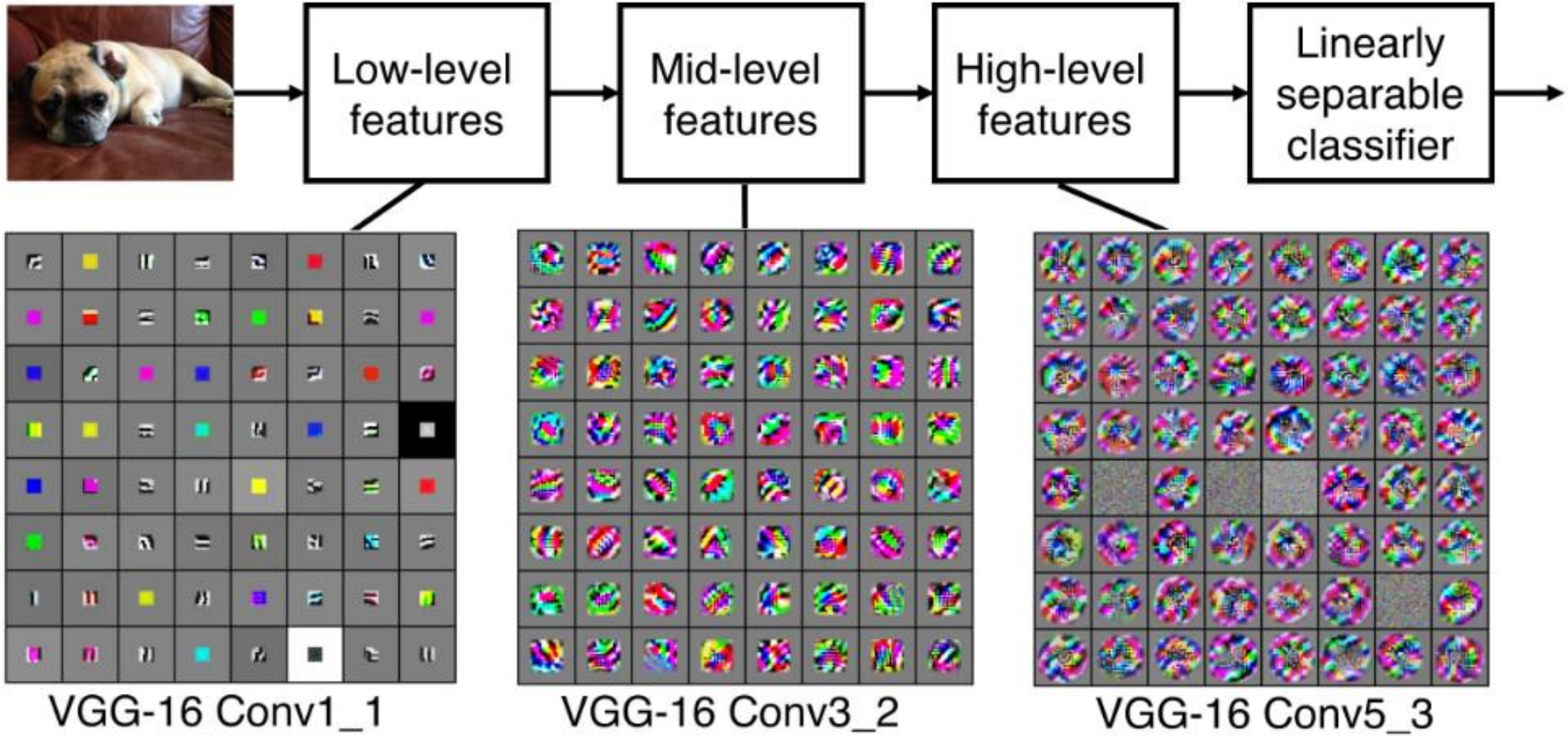
$$\max(0, x)$$



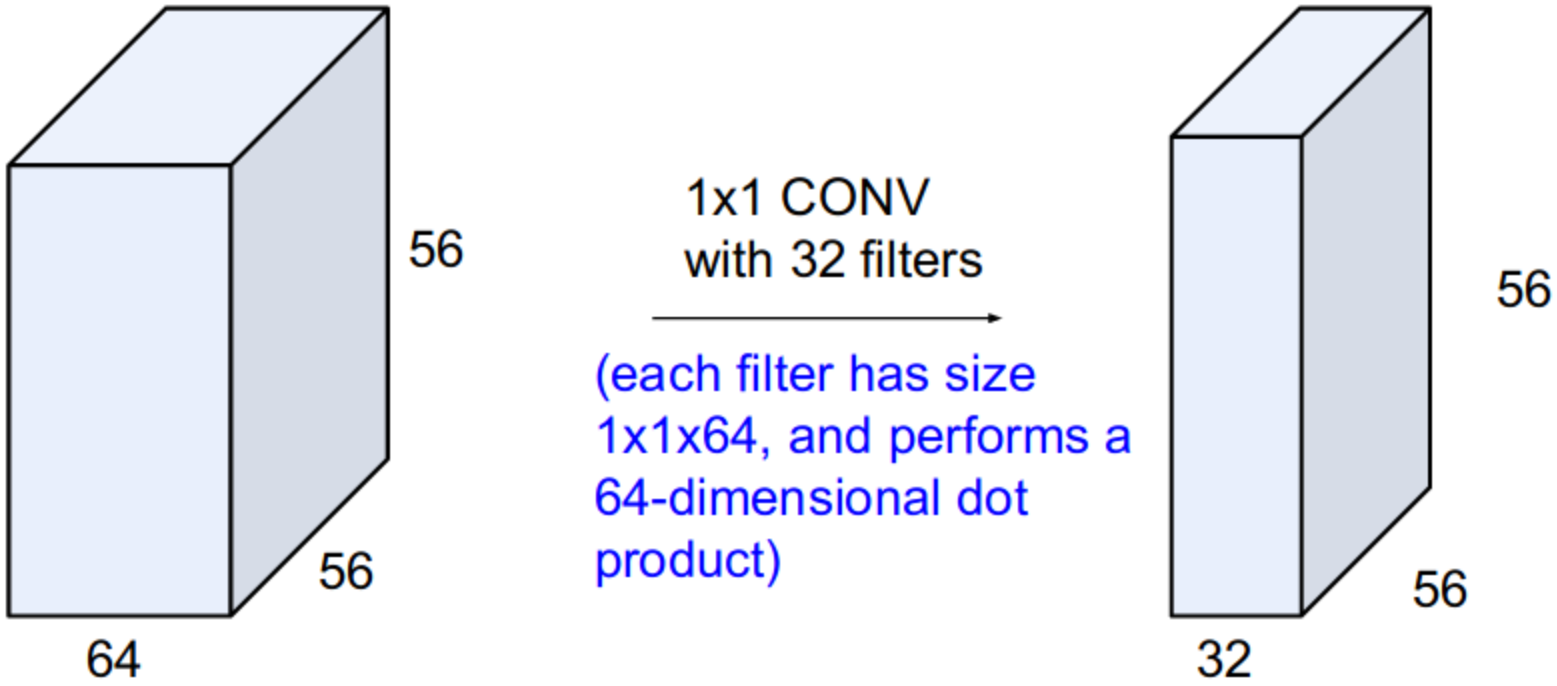
0	5	-1	2
-5	3	-4	6
6	4	0	-1
-1	8	-7	9

ReLU →

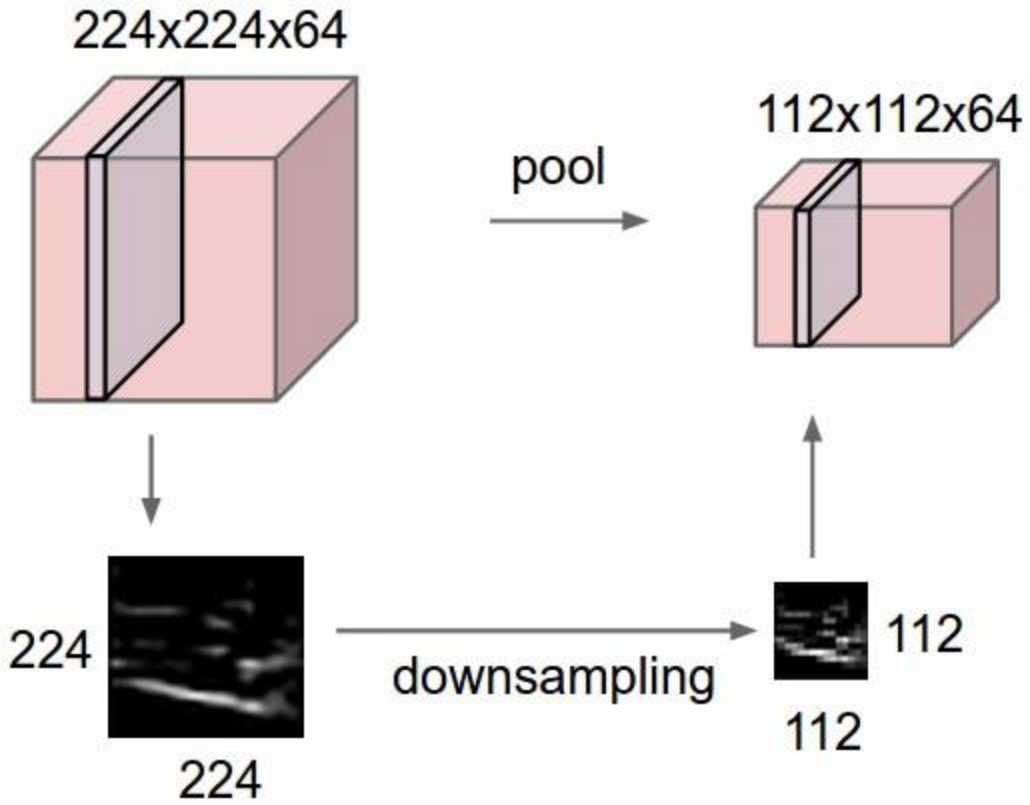
0	5	0	2
0	3	0	6
6	4	0	0
0	8	0	9


Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3

1x1 Convolution layer

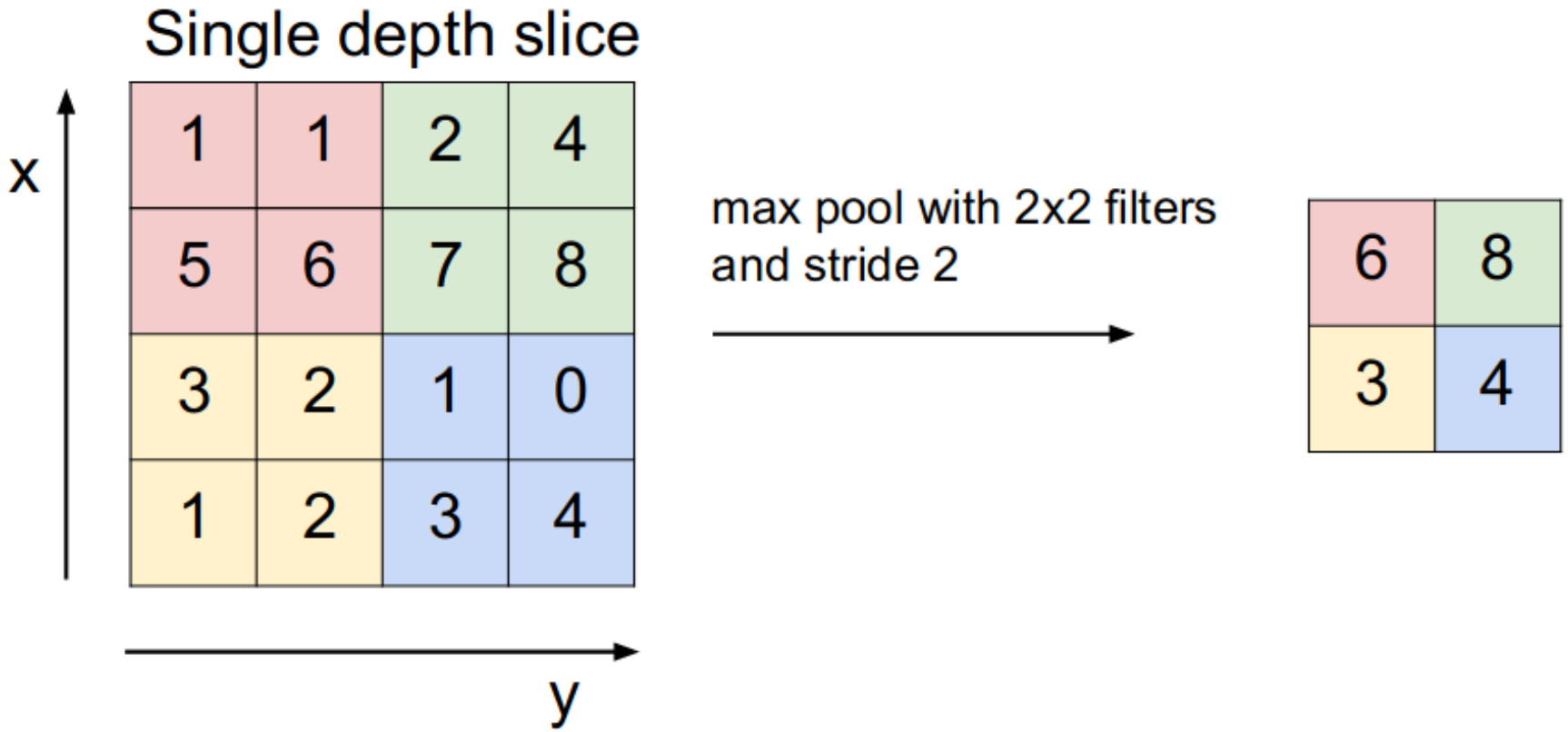


Pooling layer

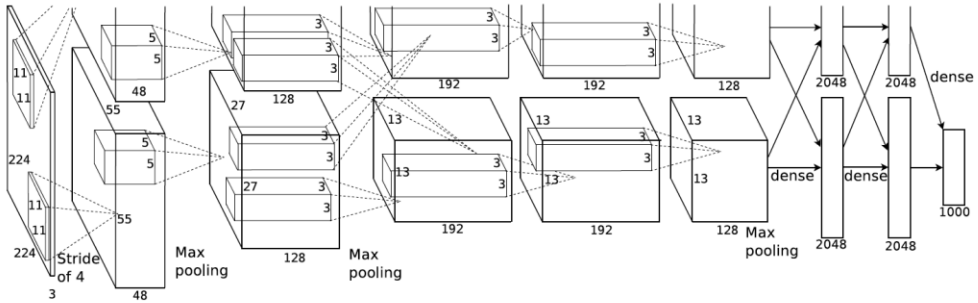


- makes the representations smaller and more manageable
- operates over each activation map independently

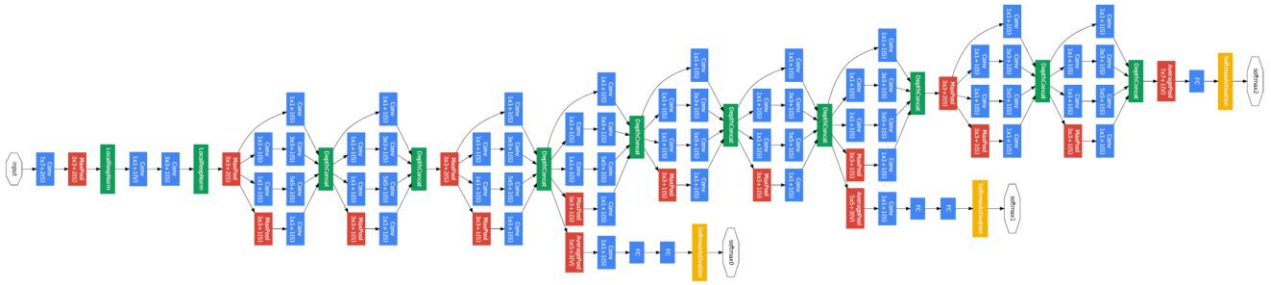
Max Pooling


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

CNN Architectures

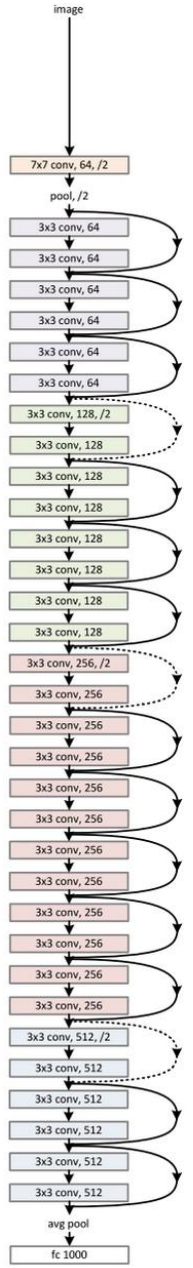


AlexNet



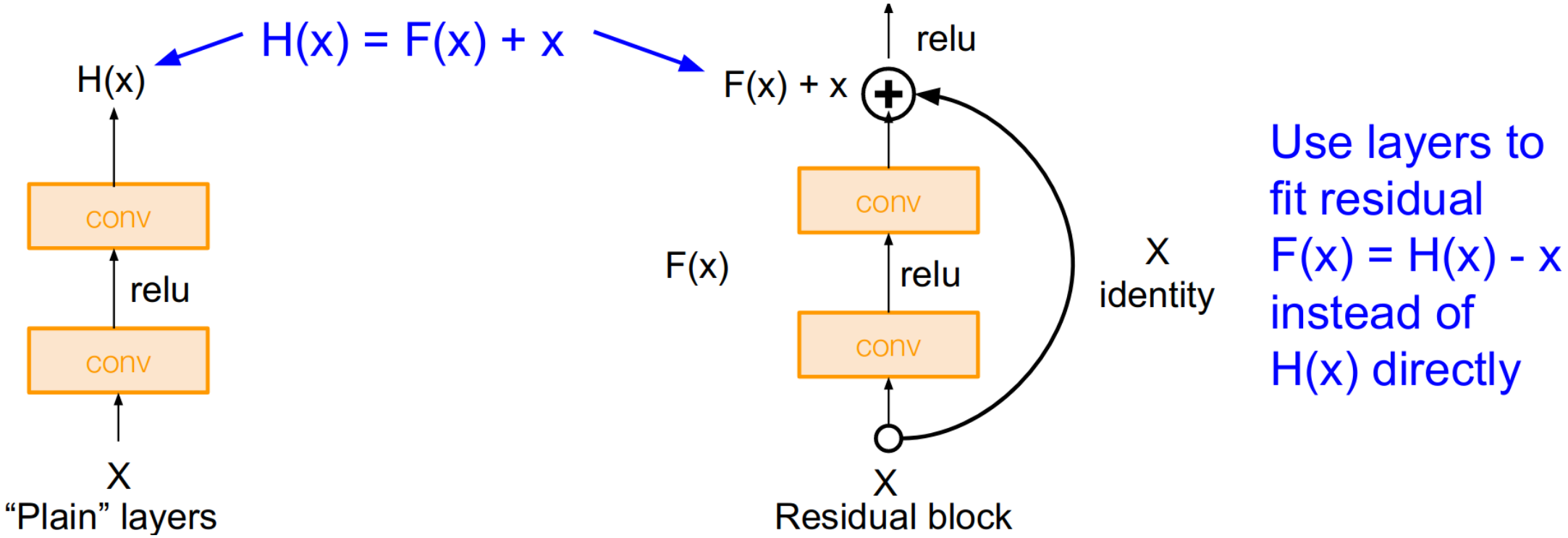
GoogleNet

VGGNet



ResNet

ResNet



Use layers to fit residual $F(x) = H(x) - x$ instead of $H(x)$ directly

Loss Functions

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:



cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

A **loss function** tells how good our current classifier is

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where x_i is image and
y_i is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)$$

Softmax Classifier (Multinomial Logistic Regression)

cat **3.2**

car 5.1

frog -1.7

scores = unnormalized log probabilities of the classes.

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

---

in summary: $$L_i = -\log(\frac{e^{sy_i}}{\sum_j e^{s_j}})$$

Softmax Classifier (Multinomial Logistic Regression)

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

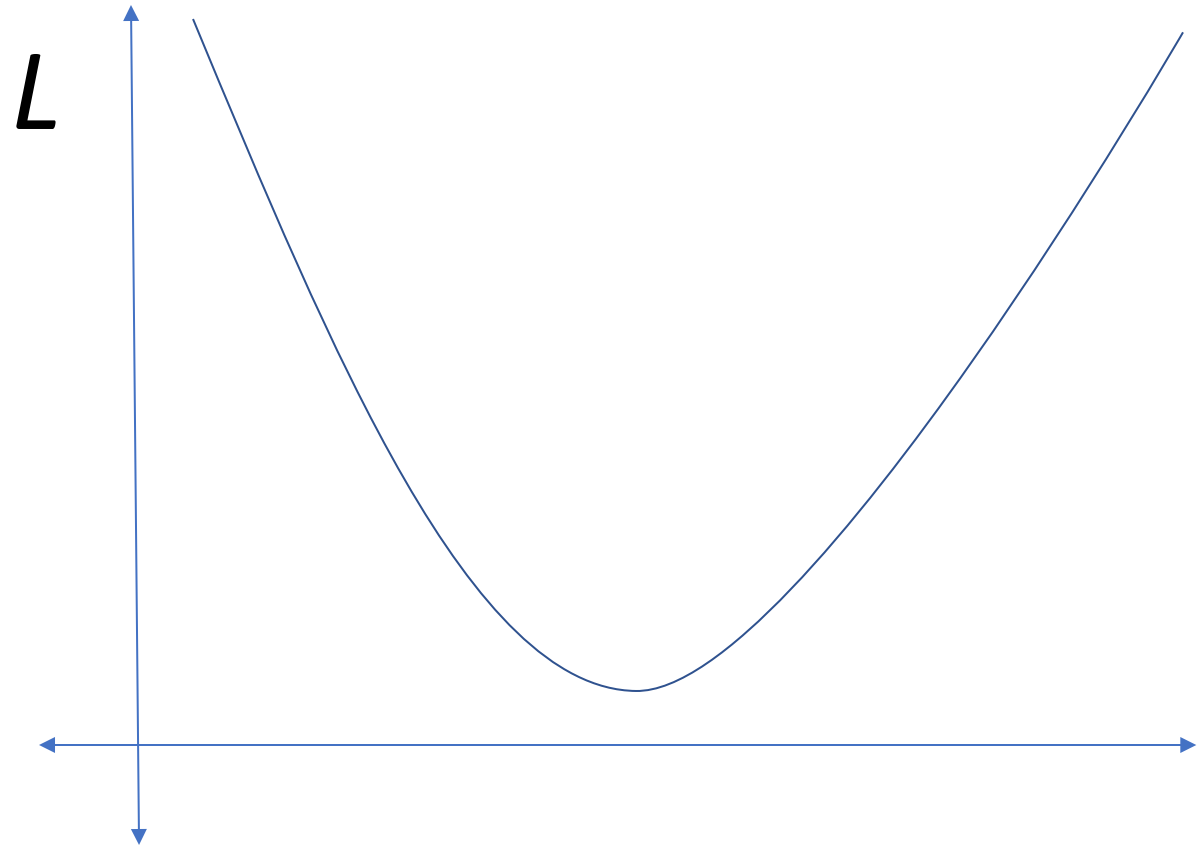
Softmax Function

$$L_i = -\log P(Y = y_i|X = x_i)$$

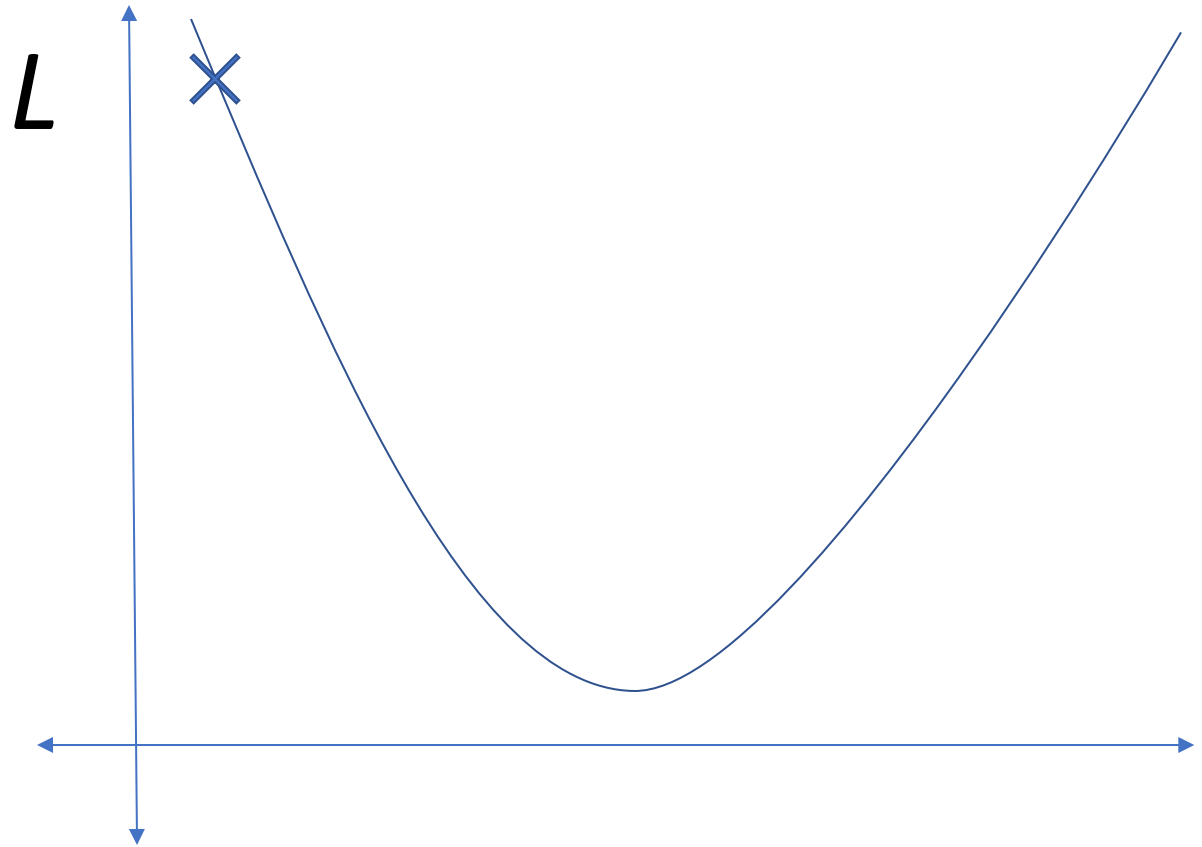
	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0	normalize	Probabilities must sum to 1	
cat	**3.2**	→	**24.5**	→	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1		164.0		0.87	
frog	-1.7		0.18		0.00	
			unnormalized probabilities		probabilities	

Optimization

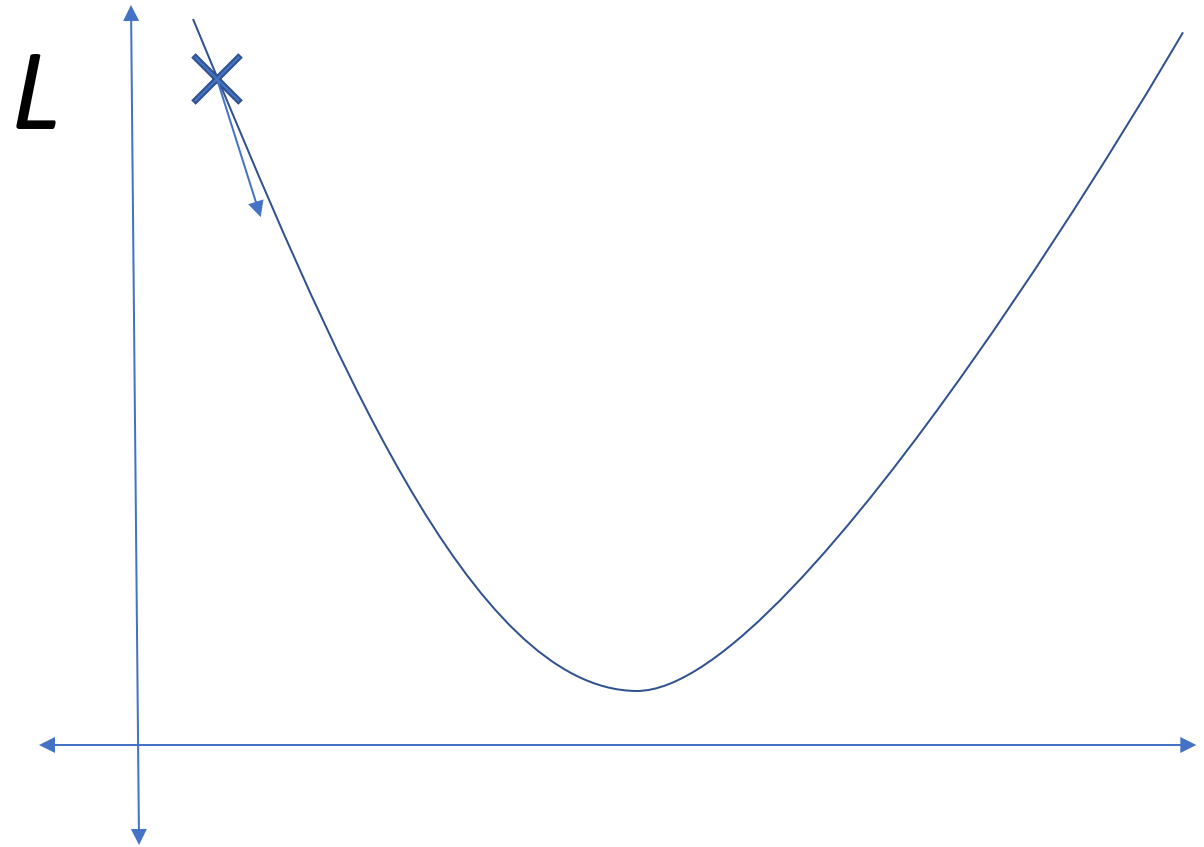
Optimization



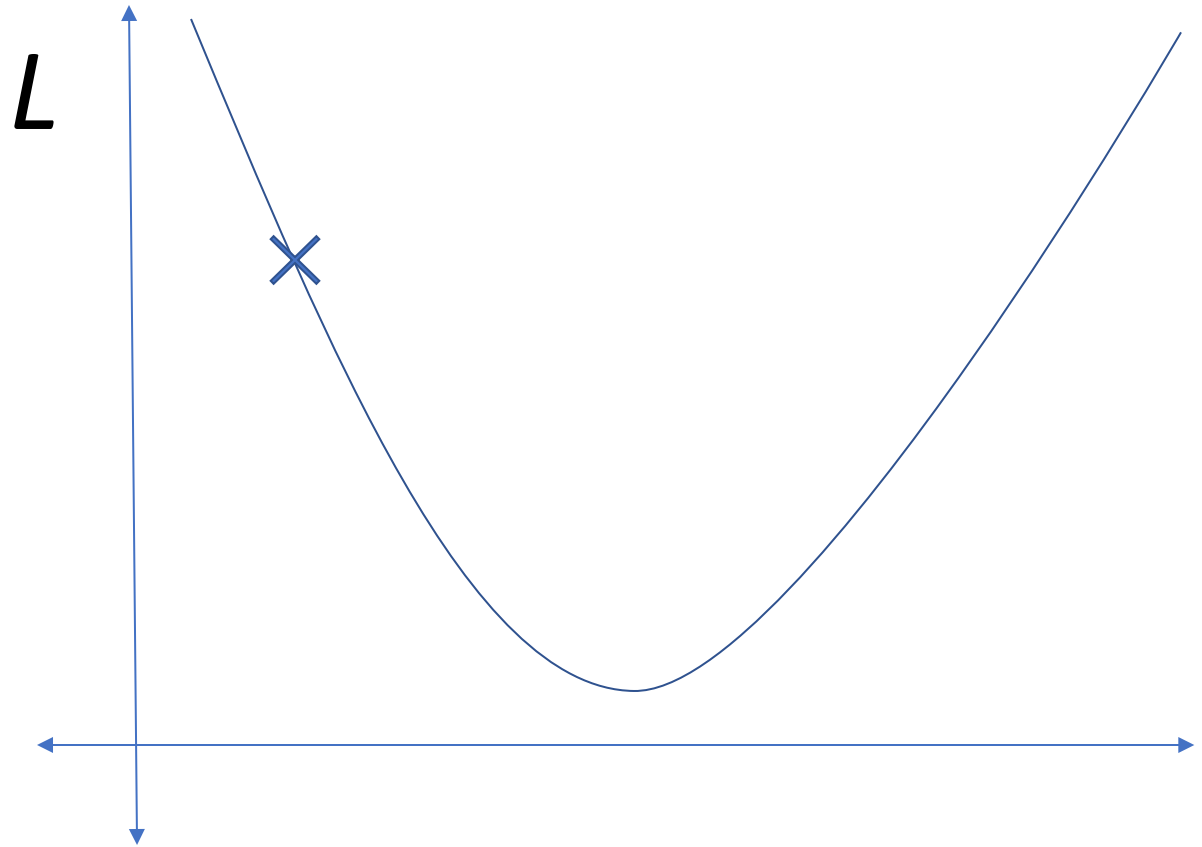
Optimization



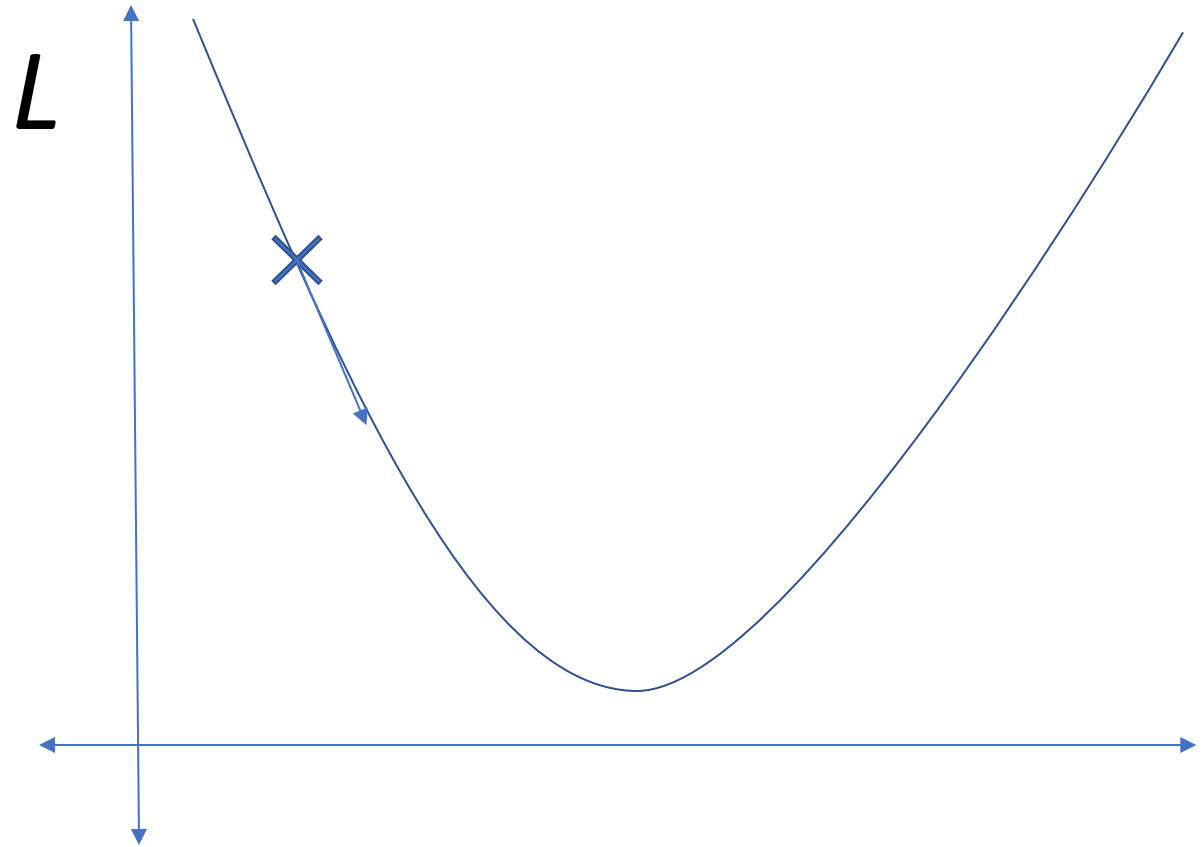
Optimization



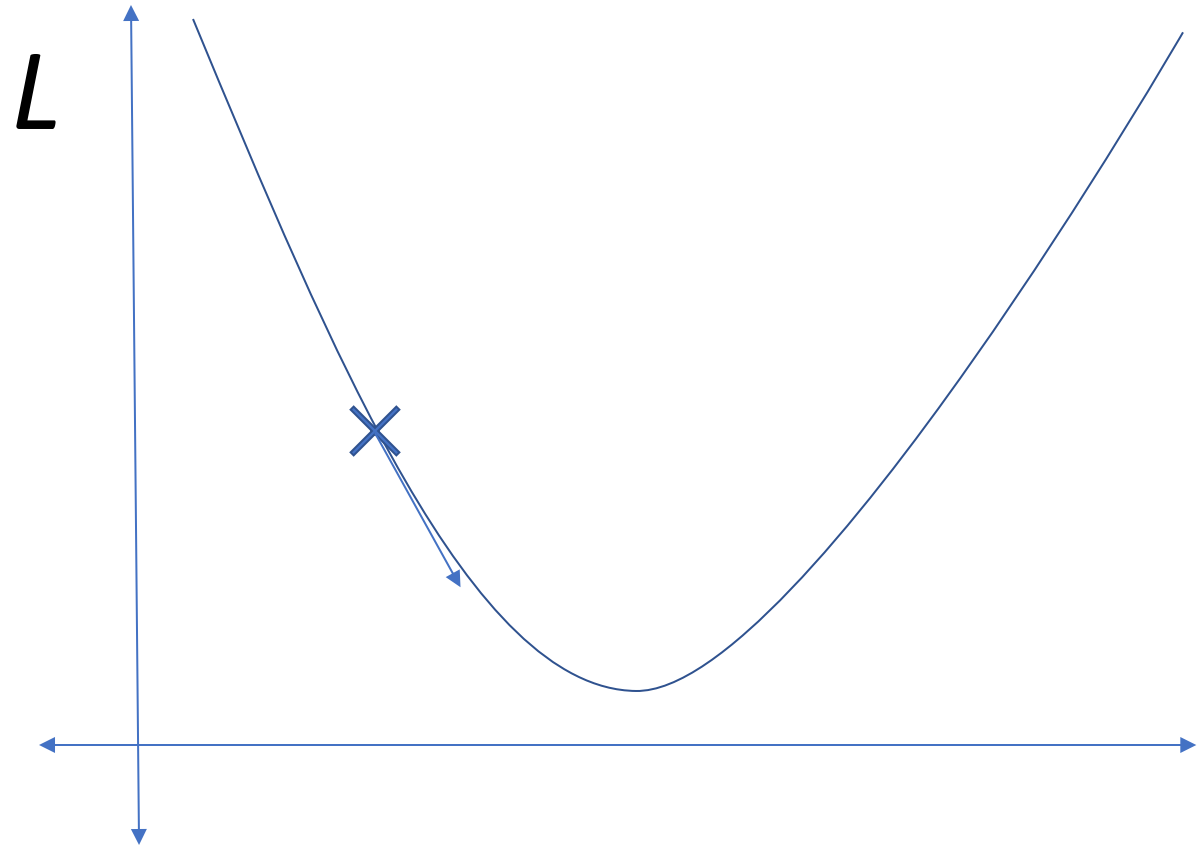
Optimization



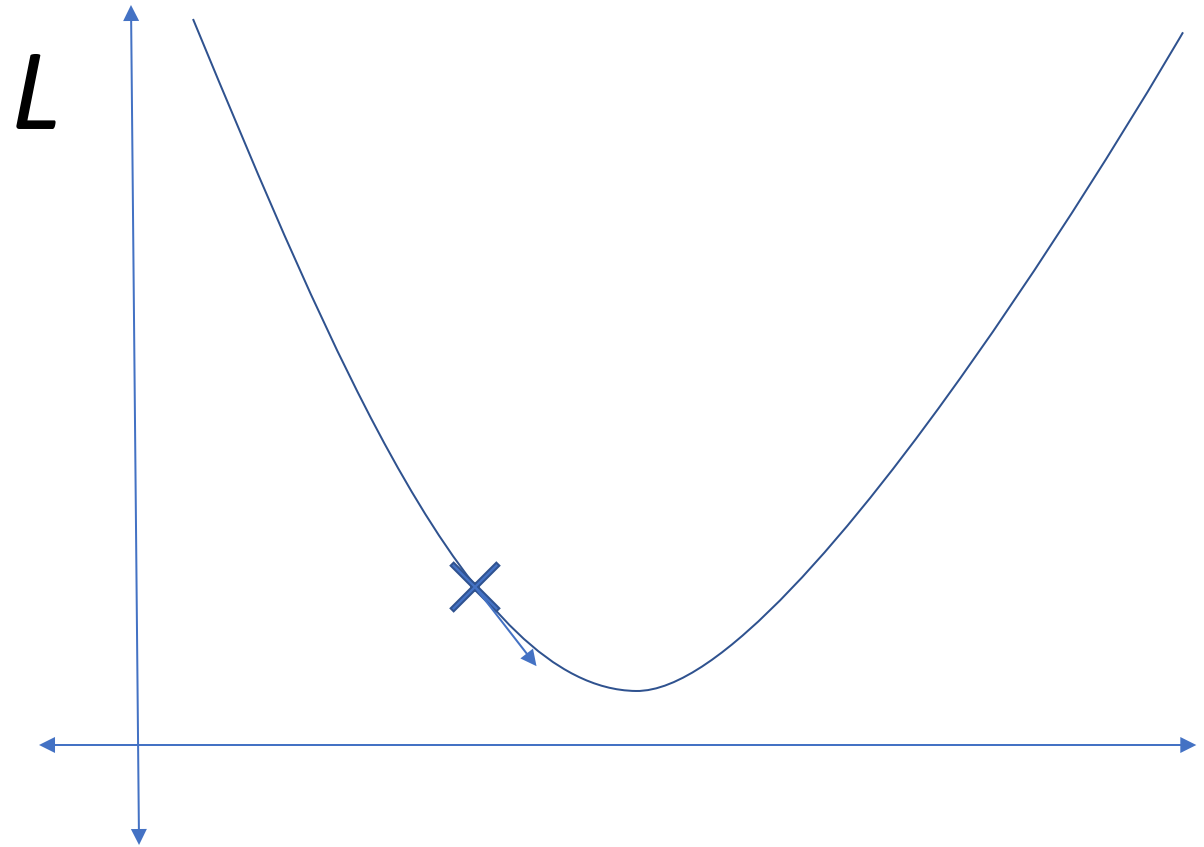
Optimization



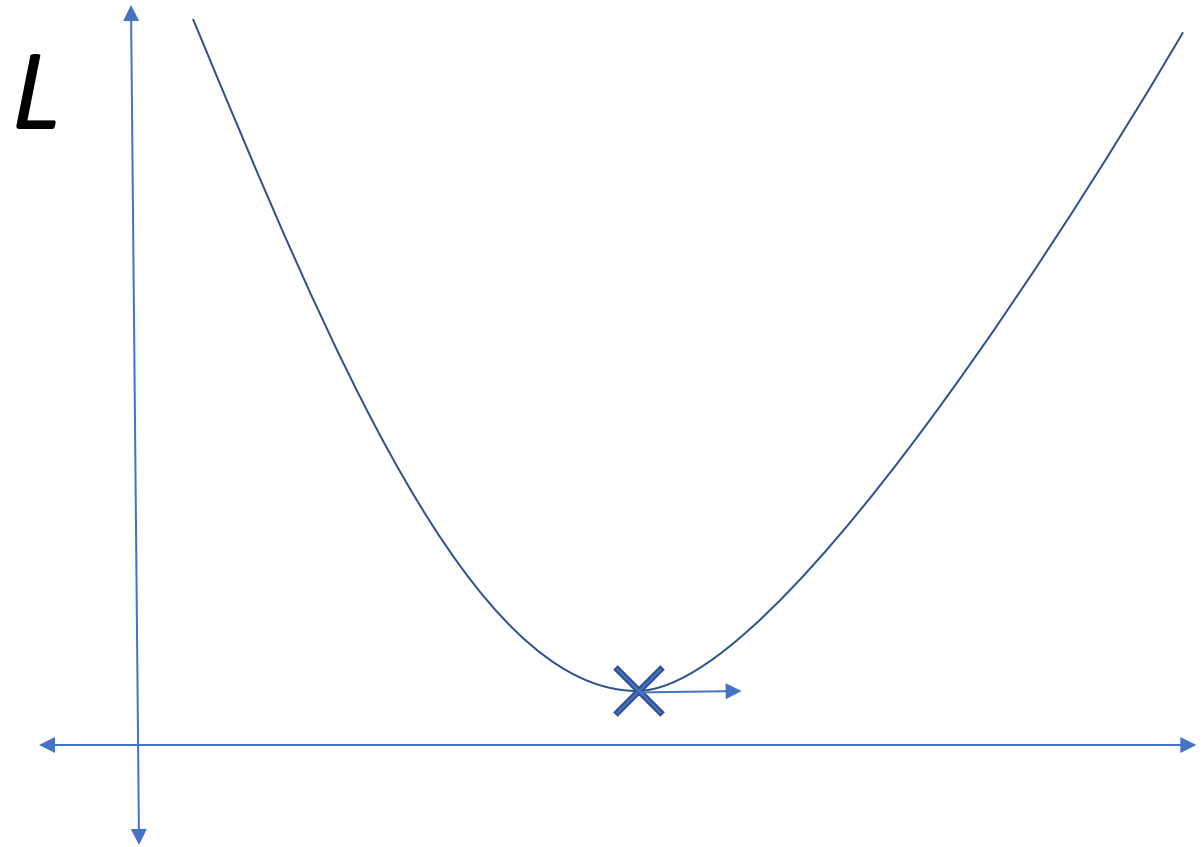
Optimization



Optimization



Optimization



Gradient Descent

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

R(W): Regularization term

Gradient Descent

```
while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Stochastic Gradient Descent

$$L(W) = \frac{1}{N}\sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N}\sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

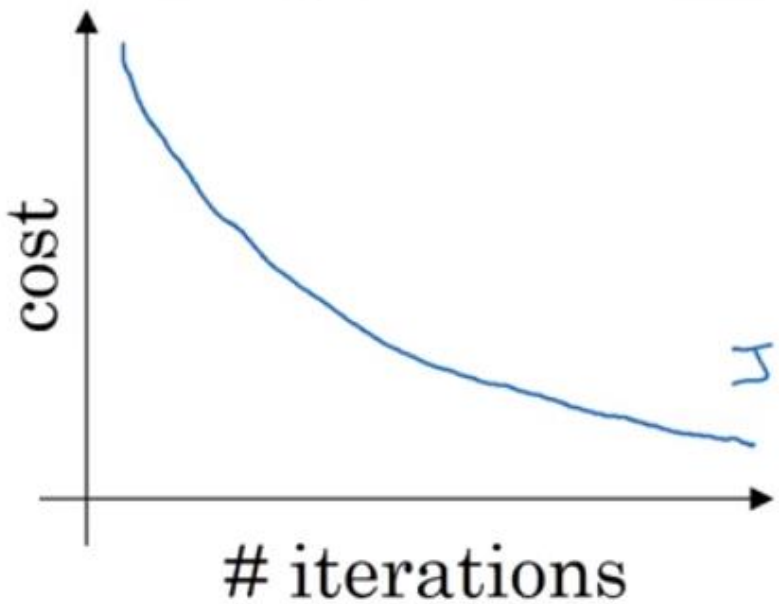
Full sum expensive when N is large!

Approximate sum using a **minibatch** of examples
32 / 64 / 128 common

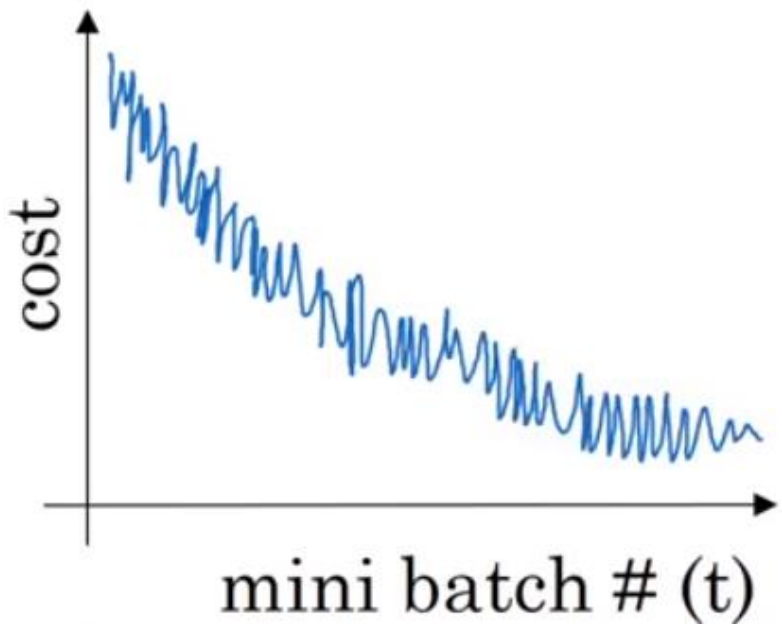
Stochastic Gradient Descent

```
while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

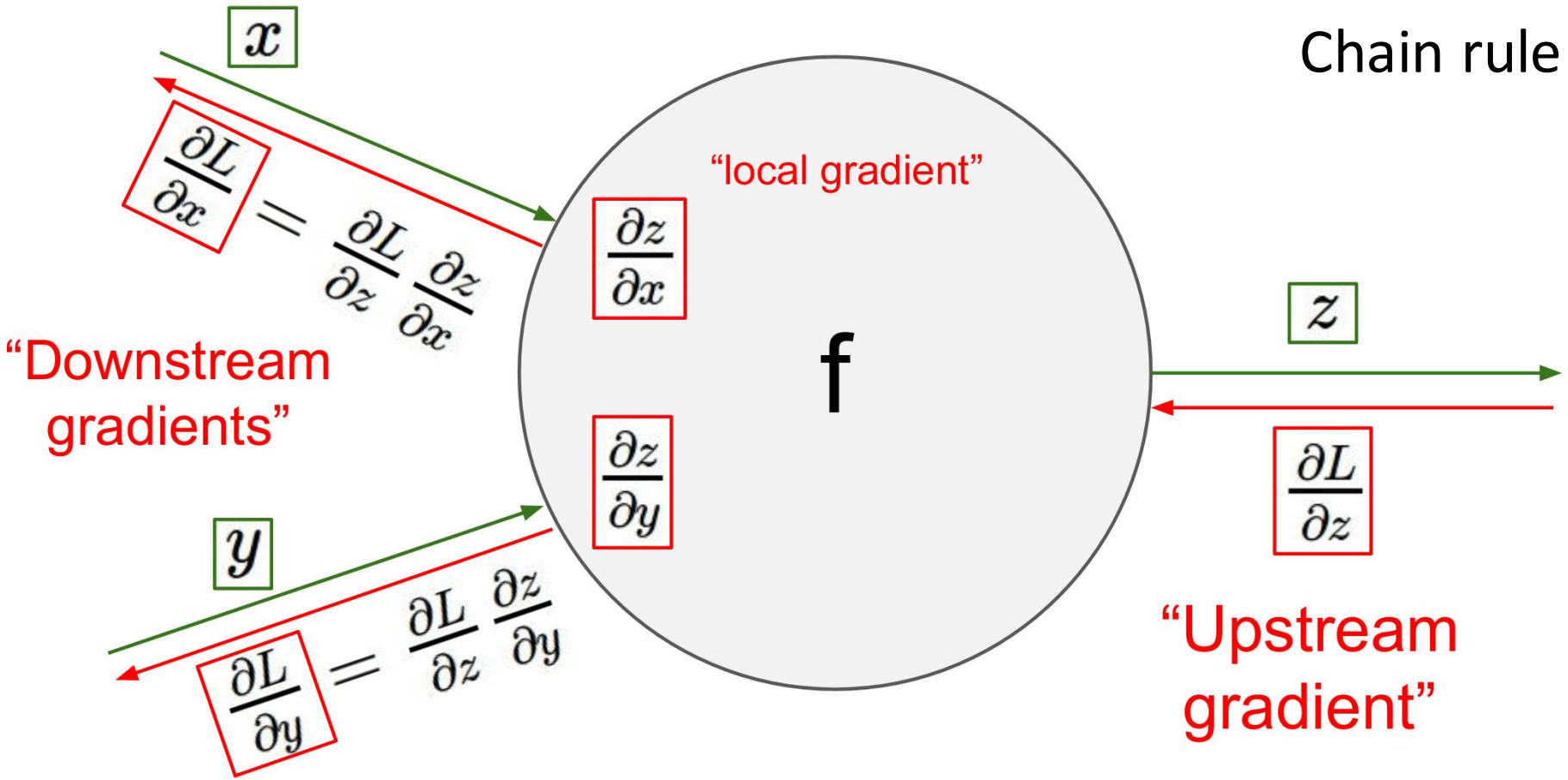
Batch gradient descent



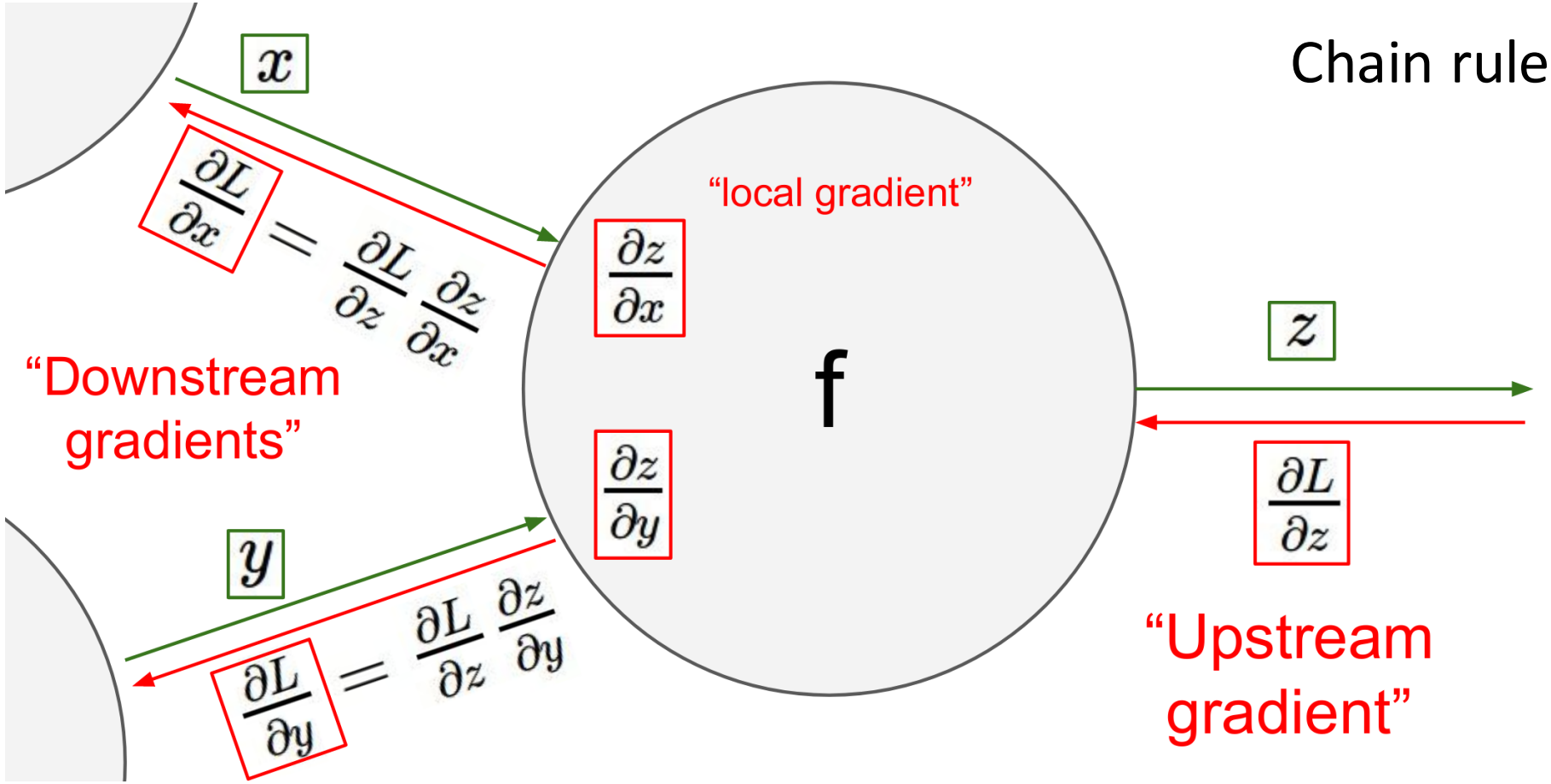
Mini-batch gradient descent



Backpropagation

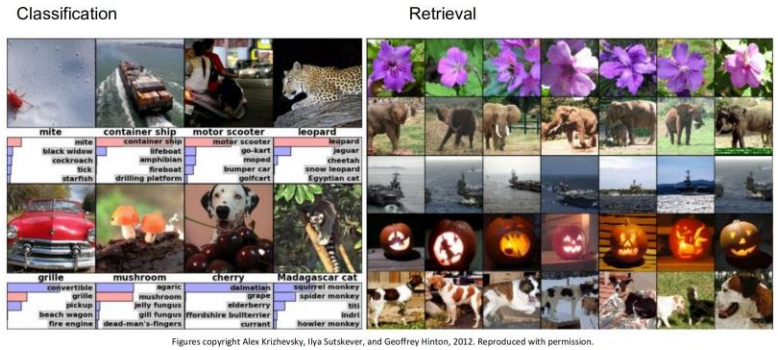


Backpropagation

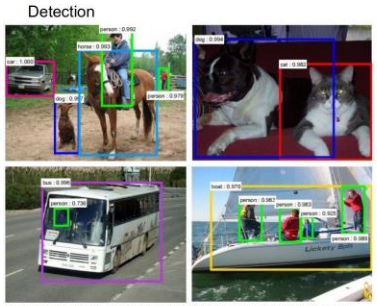


Chain rule

CNNs are Everywhere



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

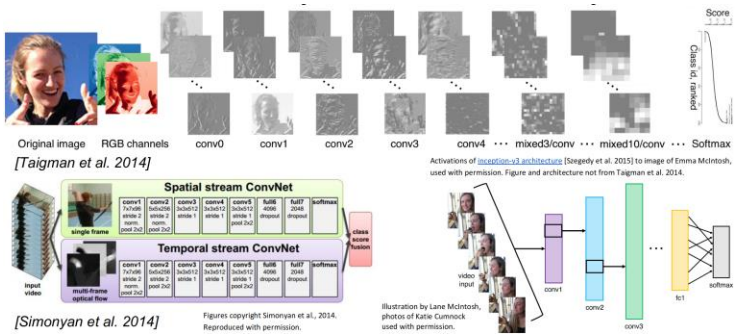


Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]



[Taigman et al. 2014]

[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.

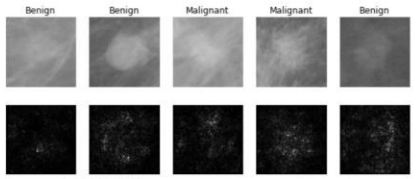
Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.


Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

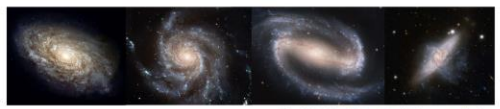
[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.



[Levy et al. 2016]

Figure copyright Levy et al. 2016. Reproduced with permission.


[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Sermanet et al. 2011]
[Ciresan et al.]

Photos by Lane McIntosh. Copyright CS231n 2017.


This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge


Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010