GETTING STARTED WITH DEEP LEARNING

Konstantin Kiselev
NVIDIA Deep Learning Institute Approved Instructor
NVIDIA Corporation

DEEP LEARNING INSTITUTE

DLI Mission

Helping people solve challenging problems using AI and deep learning.

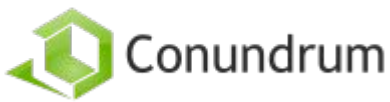
- Developers, data scientists and engineers
- Self-driving cars, healthcare and robotics
- Training, optimizing, and deploying deep neural networks

TOPICS

1. Introduction to AI and Machine learning
2. Deep learning motivation
3. Introduction to Deep learning
4. Convolutional networks
5. Applications
6. Deployment

Konstantin Kiselev

Emails: mr.konstk@gmail.com, kk@conundrum.ai



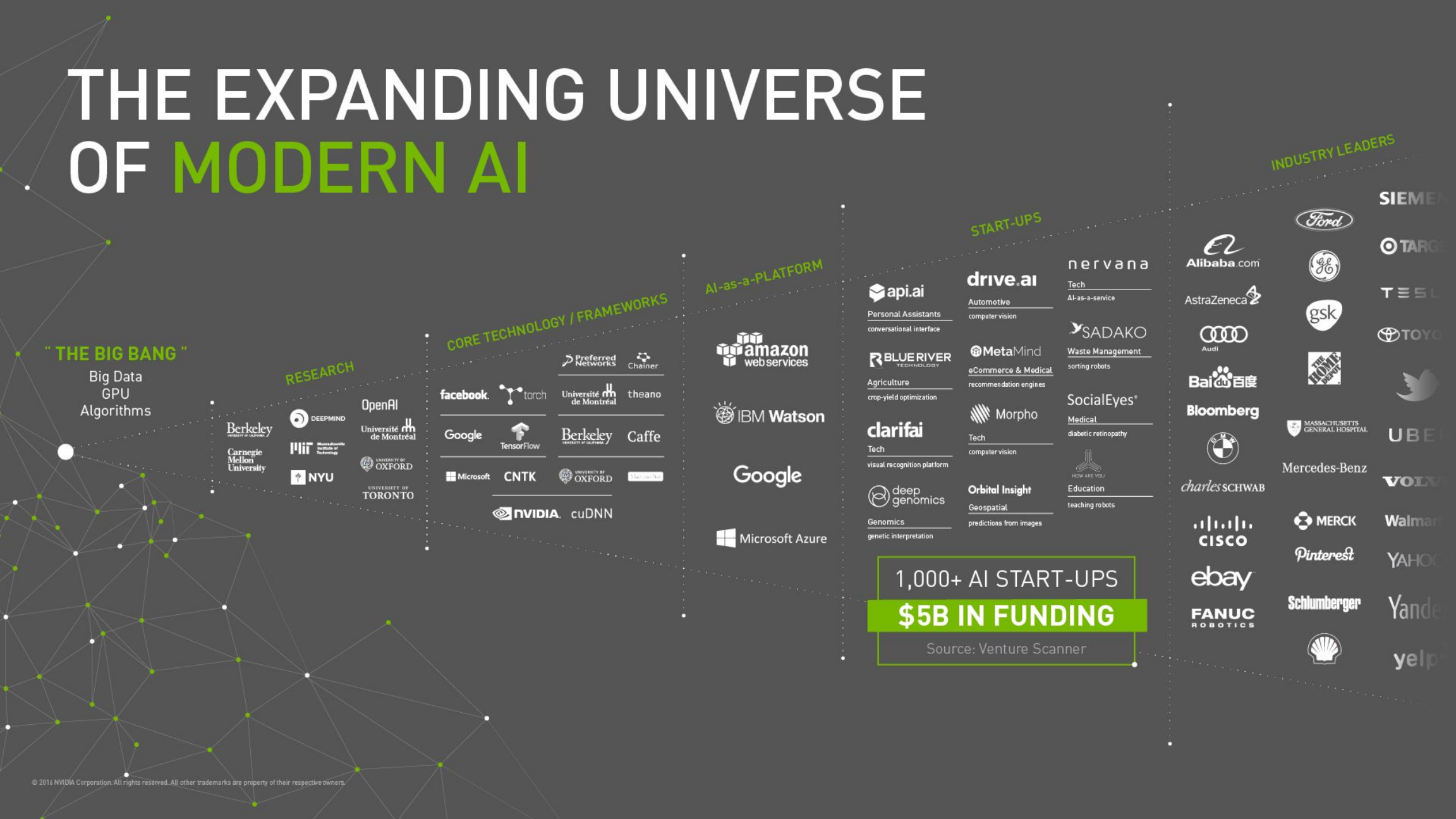
conundrum.ai

Consulting and R&D in the field of machine learning, deep learning and large scale processing for business

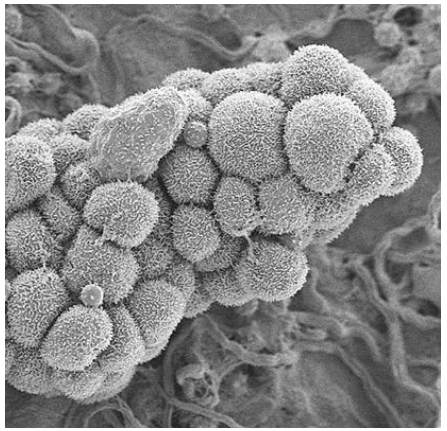
ylabs.ai

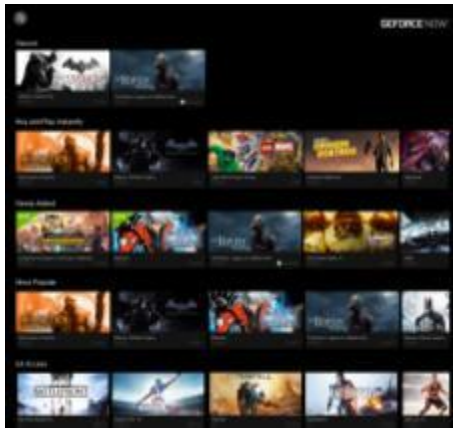
R&D in skin care: skin disease and pathologies recognition

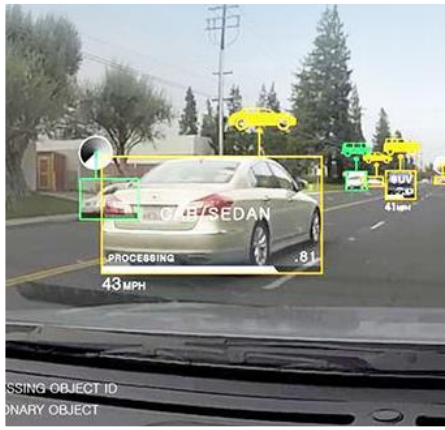
THE EXPANDING UNIVERSE OF MODERN AI



DEEP LEARNING EVERYWHERE







INTERNET & CLOUD

Image Classification
Speech Recognition
Language Translation
Language Processing
Sentiment Analysis
Recommendation

MEDICINE & BIOLOGY

Cancer Cell Detection
Diabetic Grading
Drug Discovery

MEDIA & ENTERTAINMENT

Video Captioning
Video Search
Real Time Translation

SECURITY & DEFENSE

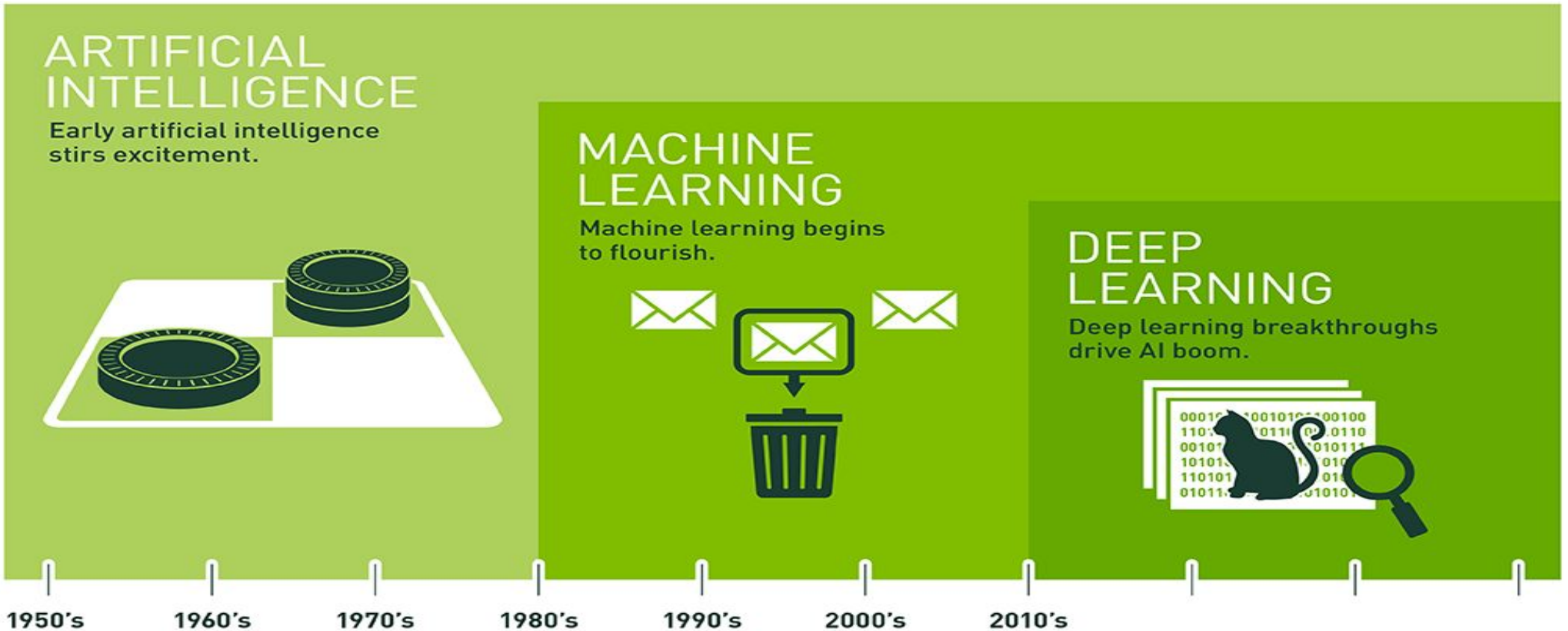
Face Detection
Video Surveillance
Satellite Imagery

AUTONOMOUS MACHINES

Pedestrian Detection
Lane Tracking
Recognize Traffic Sign

MACHINE LEARNING, BRIEFLY

DEFINITIONS



MACHINE LEARNING

Supervised learning

Labeled data for training

$X \rightarrow Y: p(Y|X)$

Unsupervised learning

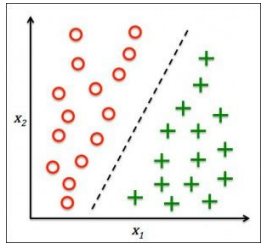
Unlabeled data

$X \rightarrow p(X)$

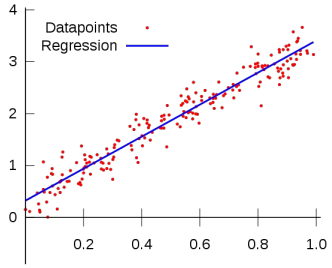
Semi-supervised learning

Reinforcement learning

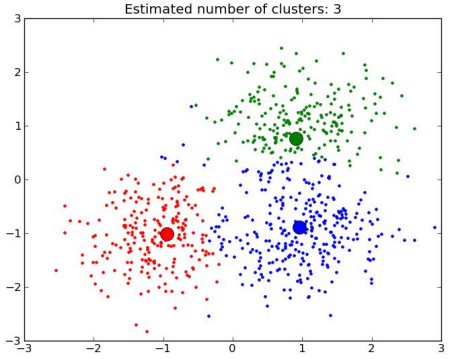
Classification



Regression



Clustering



ML COMPONENTS

Data

Nature → data

Described by probability distribution

$p_{data}(Y|X)$ - unknown

Model

Find approximation

$p_{model}(Y|X) \approx p_{data}(Y|X)$

Usually we choose model family - specific function is defined by model parameters (or weights)

Cost function

How to find the distance between model model and data probability distributions?

KL divergence:

$$D_{\mathrm{KL}}\left(\hat{p}_{\mathrm{data}} \| p_{\mathrm{model}}\right)=\mathbb{E}_{\mathbf{x} \sim \hat{p}_{\mathrm{data}}}\left[\log \hat{p}_{\mathrm{data}}(\boldsymbol{x})-\log p_{\mathrm{model}}(\boldsymbol{x})\right]$$

Minimization of KL divergence - maximization of log-likelihood

$$\boldsymbol{\theta}_{\mathrm{ML}}=\arg \max _{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x} \sim \hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x} ; \boldsymbol{\theta})$$

Cost function:

$$J(\boldsymbol{\theta})=\mathbb{E}_{\mathbf{x}, \mathrm{y} \sim \hat{p}_{\mathrm{data}}} L(\boldsymbol{x}, y, \boldsymbol{\theta})=\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)$$

$$L(\boldsymbol{x}, y, \boldsymbol{\theta})=-\log p(y \mid \boldsymbol{x} ; \boldsymbol{\theta})$$

Optimization algorithm

How to find model parameters which minimize cost function?

Gradient descent based algorithms

MODELS EXAMPLES

Linear regression

Data - from normal distribution

Model:

$$p(y \mid \boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{N}(y; \boldsymbol{\theta}^\top \boldsymbol{x}, \boldsymbol{I})$$

Cost function:

$$\mathrm{MSE}_{\mathrm{train}} = \frac{1}{m} \sum_{i=1}^{m} ||\hat{y}^{(i)} - y^{(i)}||^2$$

Logistic regression - binary classification

Data - from Bernoulli distribution

Model:

$$p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x}) \qquad \sigma(z) \equiv \frac{1}{1 + e^{-z}}.$$

Cost function:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^{m} (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) + y^{(i)} \log h_\theta(x^{(i)}) \right]$$

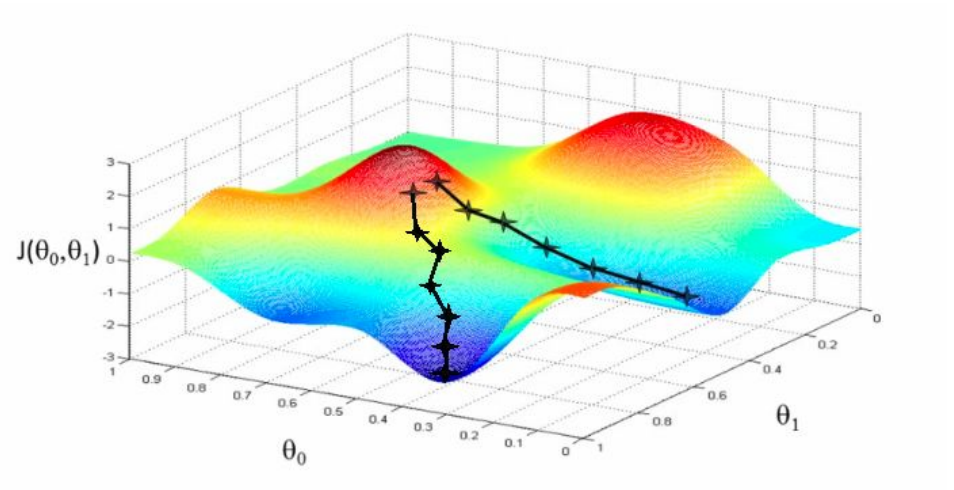
OPTIMIZATION ALGORITHM

SGD (Stochastic gradient descent)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\boldsymbol{g} = \frac{1}{m'} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m'} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta})$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \boldsymbol{g}$$



SGD is useful for large dataset - computation complexity is lower

ML CHALLENGES

Underfitting

Reasons:
- not enough train data
- low model function capacity
- noisy data

Overfitting

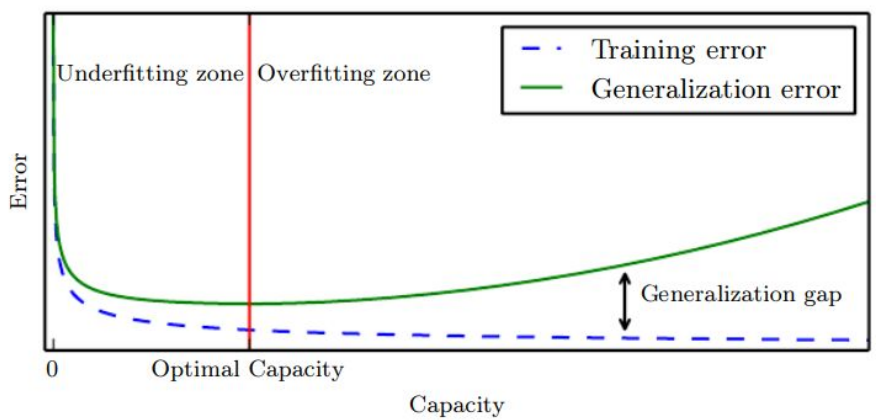
Reasons:
- not enough train data
- high complexity of model function

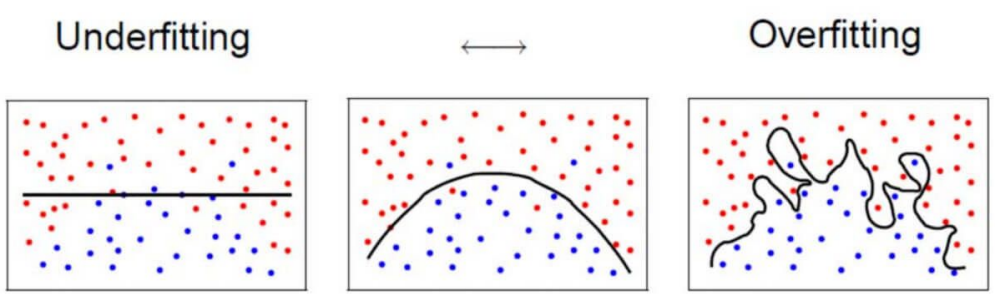
Model tries to memorize train dataset

Solutions:
- regularization (L_1, L_2)
- more train data
- others (to be discussed)

Capacity

Is your model function able to learn real data distribution?





ML CHALLENGES: ADVANCED

Curse of dimensionality

Higher dimension - more train data need!

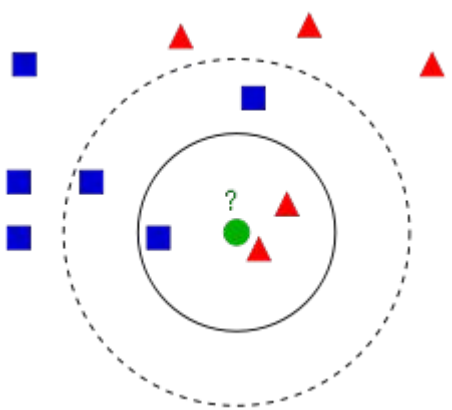
How traditional ML algorithms overcome this problem?

Smoothness prior:

$$f^*(\boldsymbol{x}) \approx f^*(\boldsymbol{x} + \epsilon)$$

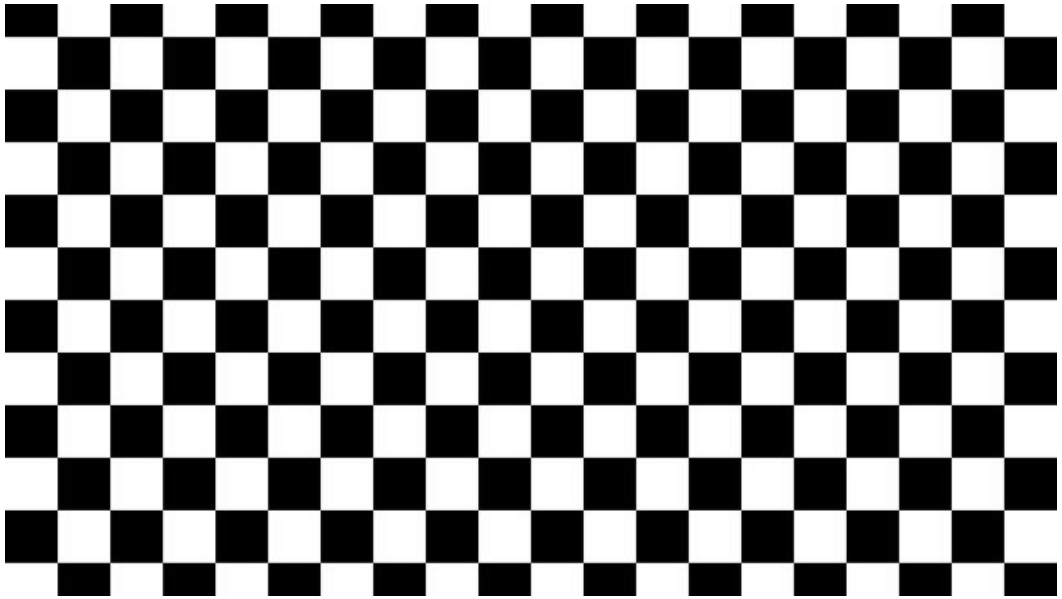
Traditional ML algorithms separate space on fixed number of regions, examples

- kNN
- Kernels machines (kernel smoothness)
- Decision tree or random forest



ML CHALLENGES: EXAMPLE

How to learn a function which is a kind of checkerboard?



ML CHALLENGES: SOLUTIONS

Whether it's possible to represent a complicated function efficiently?

Whether it's possible for estimated function to generalized well to new inputs?

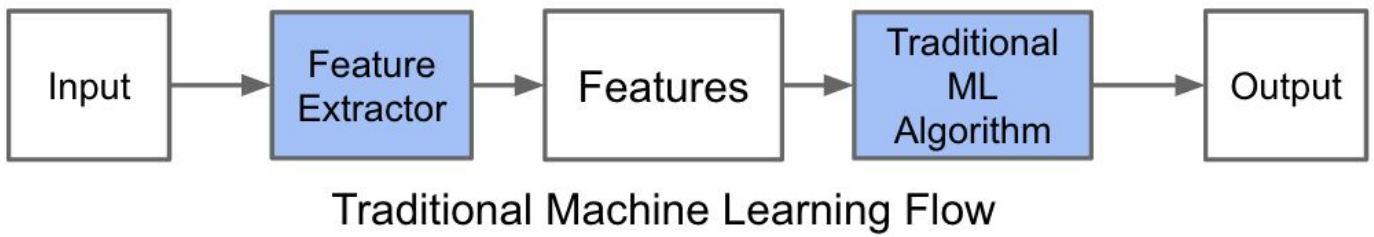
Yes, there are several approaches:

1. Define dependencies between regions - more regions
2. Task-dependence assumptions (priors)
3. **Deep learning**

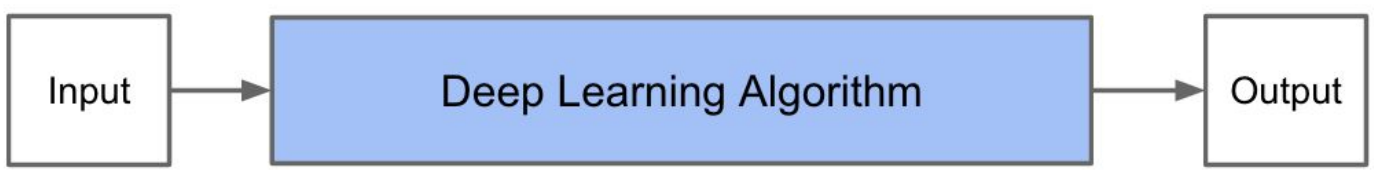
The core idea in **deep learning** is that we assume that the data was generated by the composition of factors or features, potentially at multiple levels in a hierarchy

WHAT IS DEEP LEARNING?

DEEP LEARNING vs TRADITIONAL ML



Traditional Machine Learning Flow



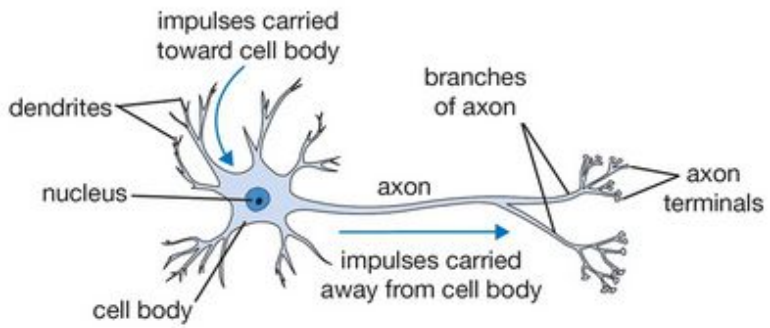
Deep Learning Flow

DEEP LEARNING BENEFITS

- Robust
  - No need to design the features ahead of time – features are automatically learned to be optimal for the task at hand
  - Robustness to natural variations in the data is automatically learned
- Generalizable
  - The same neural net approach can be used for many different applications and data types
- Scalable
  - Performance improves with more data, method is massively parallelizable

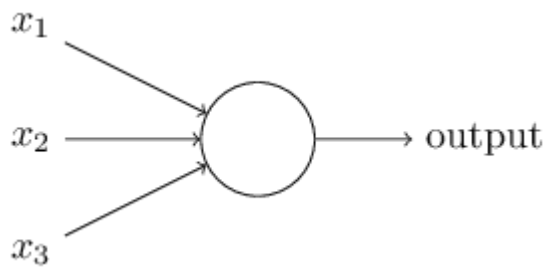
FEEDFORWARD NETWORK

Biological neuron



From Stanford cs231n lecture notes

Artificial neuron



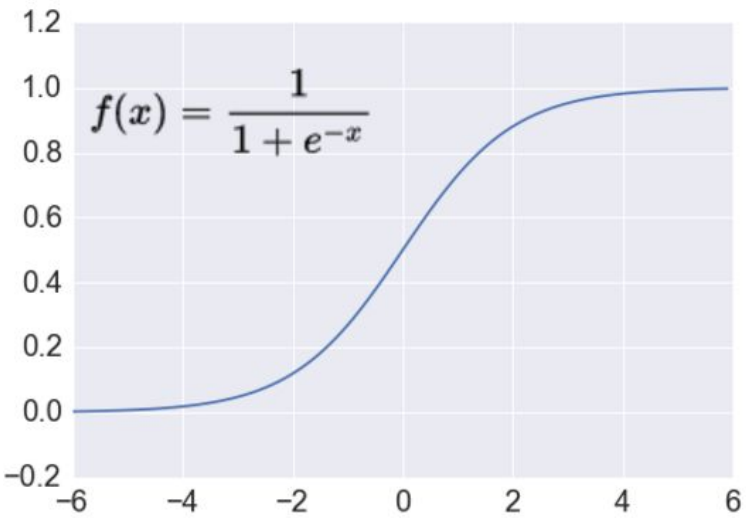
$$y = f(w_1x_1 + w_2x_2 + w_3x_3)$$

Some simple function:

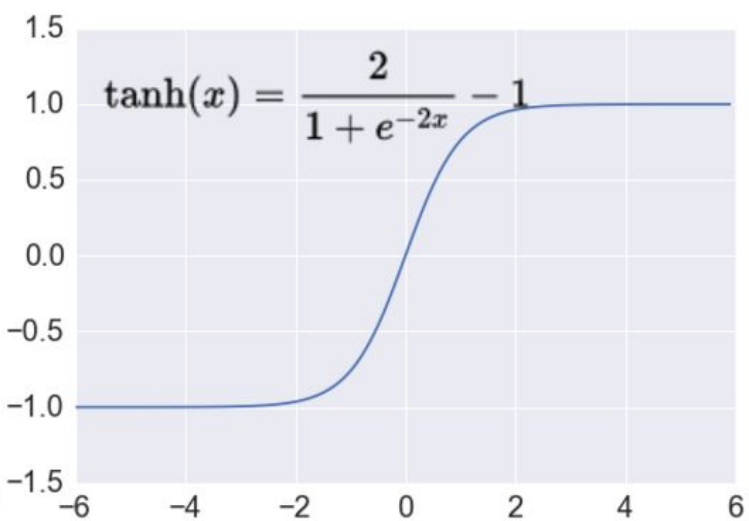
$$\mathrm{y} = \begin{cases} 0 \text{ if } w \cdot x + b \leq 0 \\ 1 \text{ if } w \cdot x + b > 0 \end{cases}$$

ACTIVATION FUNCTIONS

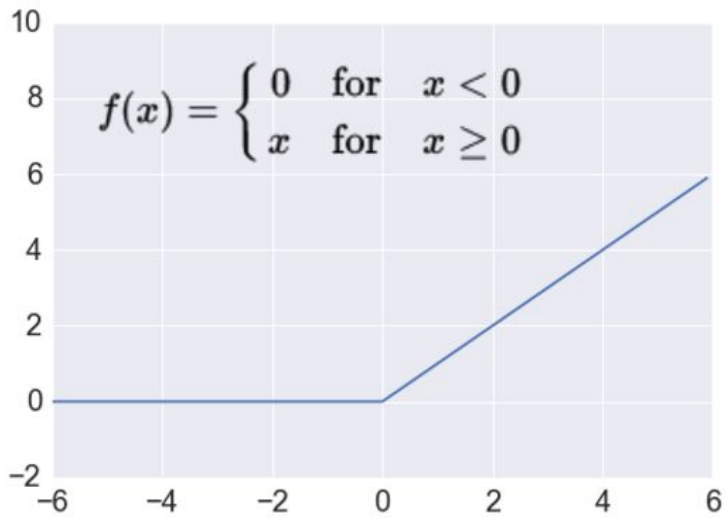
Sigmoid



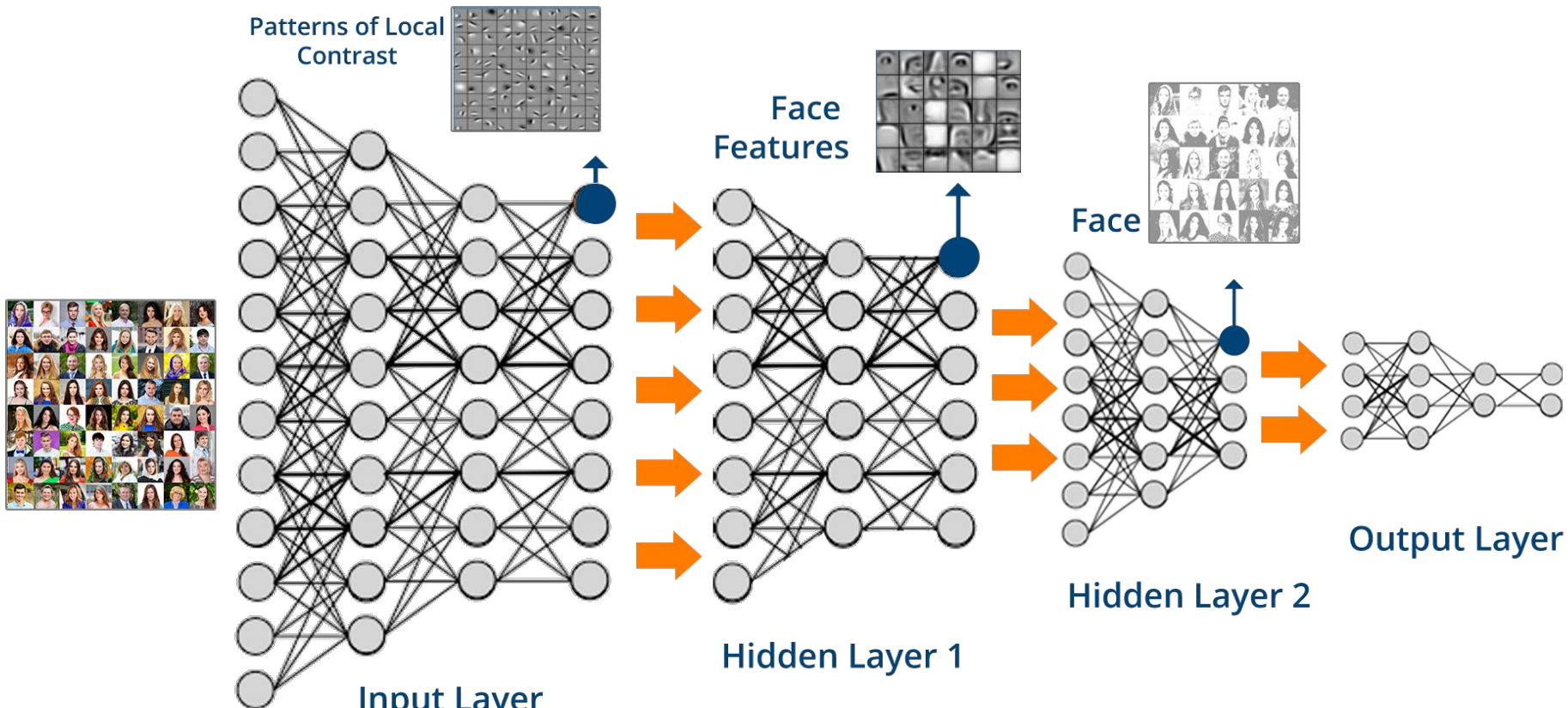
TanH



ReLU



FEEDFORWARD NETWORK



FEEDFORWARD NETWORK: LAYERS

Goal is to avoid saturation of gradients

Output Units

1. Linear Units for Gaussian Output Distributions
2. Sigmoid Units for Bernoulli Output Distributions
3. Softmax Units for Multinoulli Output Distributions

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

$$\log \text{softmax}(\boldsymbol{z})_i = z_i - \log \sum_j \exp(z_j)$$

Hidden Units

Rectified Linear Units

$$g(z) = \max\{0, z\} \qquad \boldsymbol{h} = g(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b})$$

$$z_i < 0: \; h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

Absolute value rectification

$$\alpha_i = -1 \text{ to obtain } g(z) = |z|$$

Leaky ReLU: α_i is small

Maxout $\quad g(\boldsymbol{z})_i = \max_{j \in \mathbb{G}^{(i)}} z_j$

Logistic

Tanh and others

BACKPROPAGATION

Backpropagation

1. Input
2. Feedforward
3. Output error

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

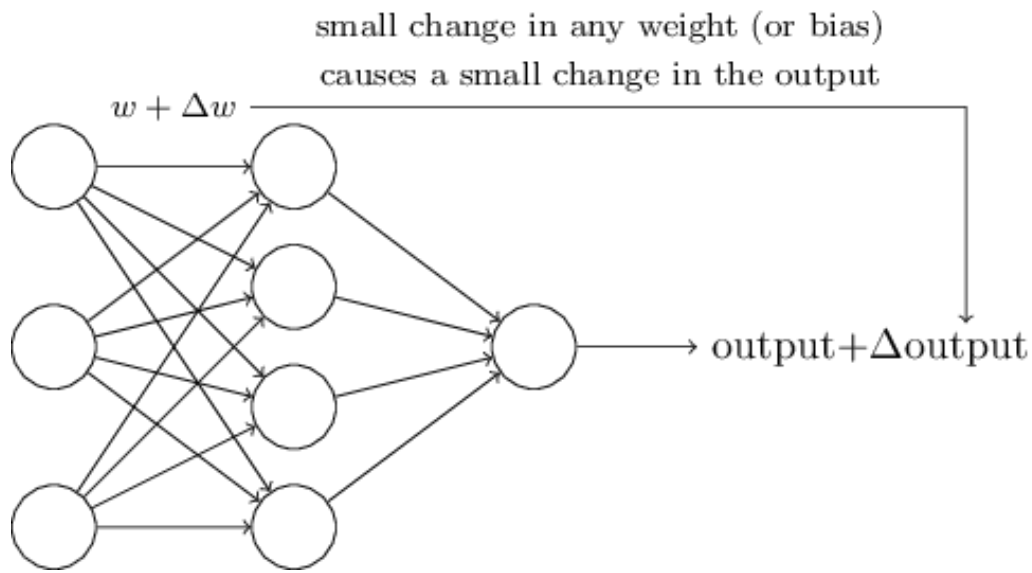
4. Backpropagate the error

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

5. Output

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$



REGULARIZATION METHODS

Parameter Norm Penalties

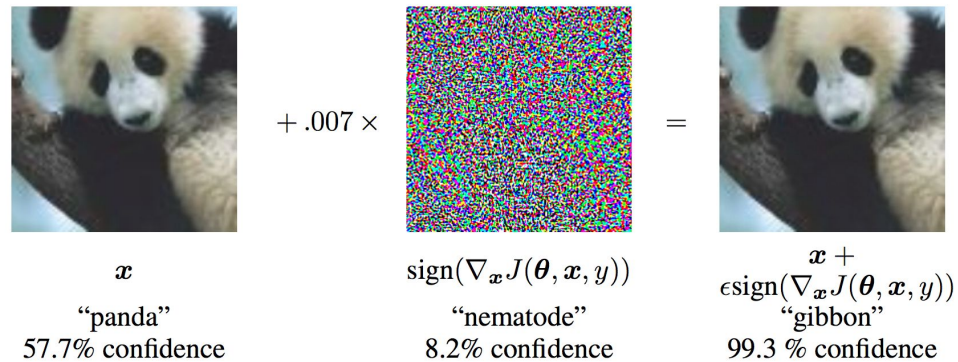
$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

L^2 Parameter Regularization (Tikhonov regularization)

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

L^1 Parameter Regularization (LASSO)

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|$$



Adversarial example

Noise robustness

- Noise applied to the hidden units is very important topic. Noise in weights:

$$\eta \mathbb{E}_{p(\boldsymbol{x},y)} \left[\|\nabla_{\boldsymbol{W}} \hat{y}(\boldsymbol{x})\|^2 \right]$$

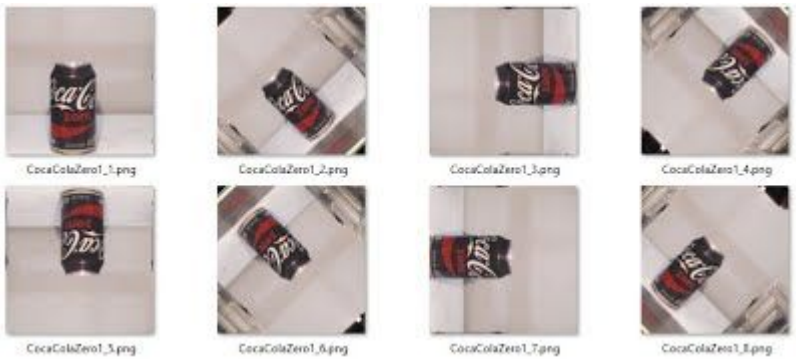
encourages the parameters to go to regions of parameter space where small perturbations of the weights have a relatively small influence on the output

- Injecting Noise at the Output Targets - for example, **label smoothing**

$\frac{\epsilon}{k-1}$ and $1 - \epsilon$

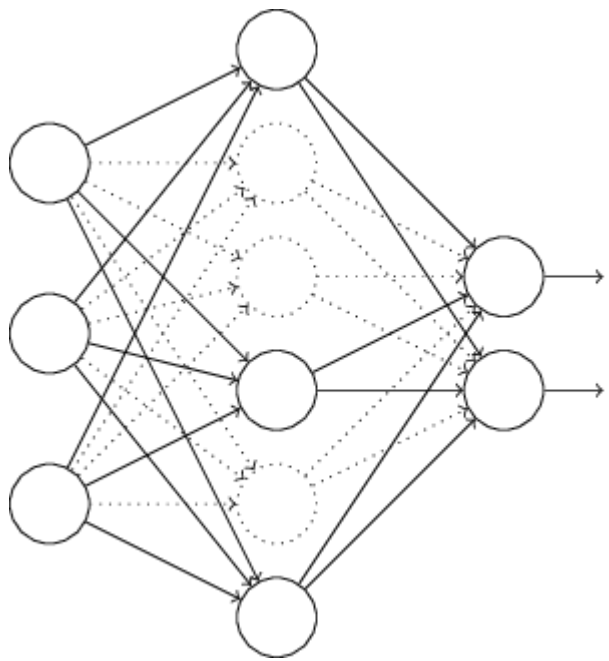
Parameter Sharing - based on prior knowledge about data
Sparse Representations - apply regularisation to the input of activation function
Bagging and Other Ensemble Methods - decrease the generalization error
Adversarial training - against adversarial attacks

REGULARIZATION: DATA AUGMENTATION



REGULARIZATION: DROPOUT

Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks



OPTIMIZATION: METHODS

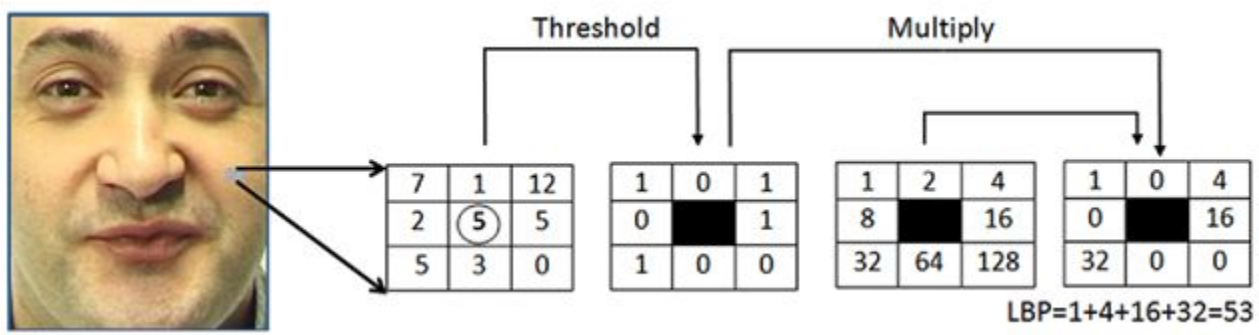
Optimization methods

1. SGD
2. Momentum - the step size is largest when many successive gradients point in exactly the same direction
3. Nesterov momentum - the gradient is evaluated after the current velocity is applied
4. AdaGrad (adaptive gradient algorithm)
5. RMSProp (Root Mean Square Propagation)
6. Adam (Adaptive Moment Estimation)

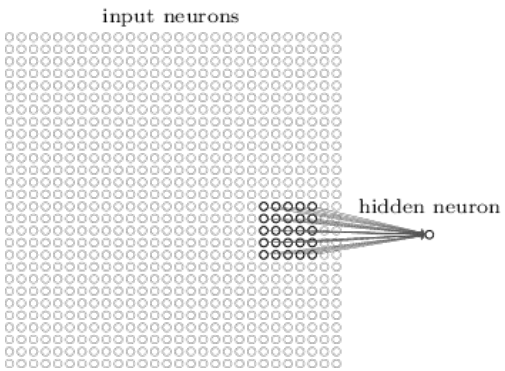
Parameters initialization

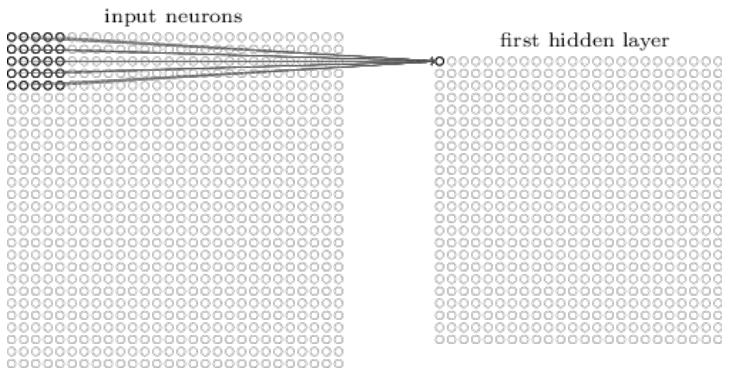
- to “break symmetry” between different units

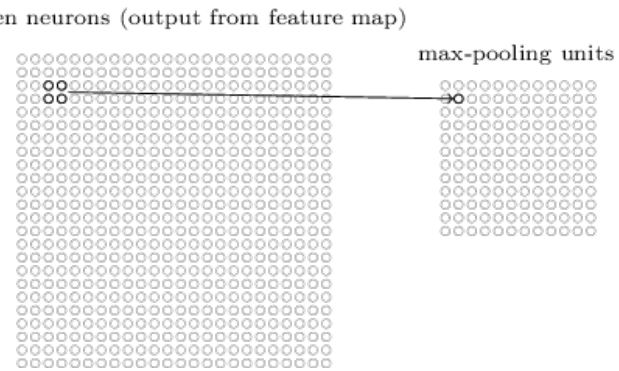
ARCHITECTURE DESIGN: EXAMPLE

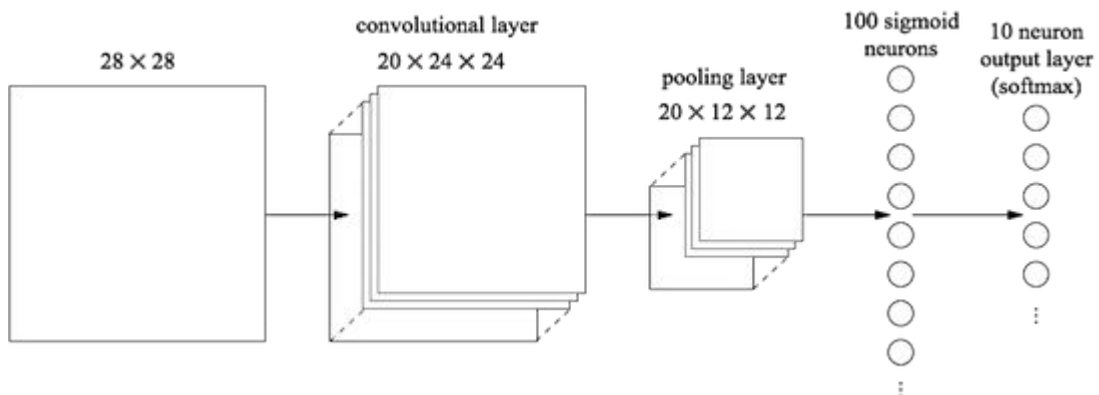


CONVOLUTIONAL NETWORKS



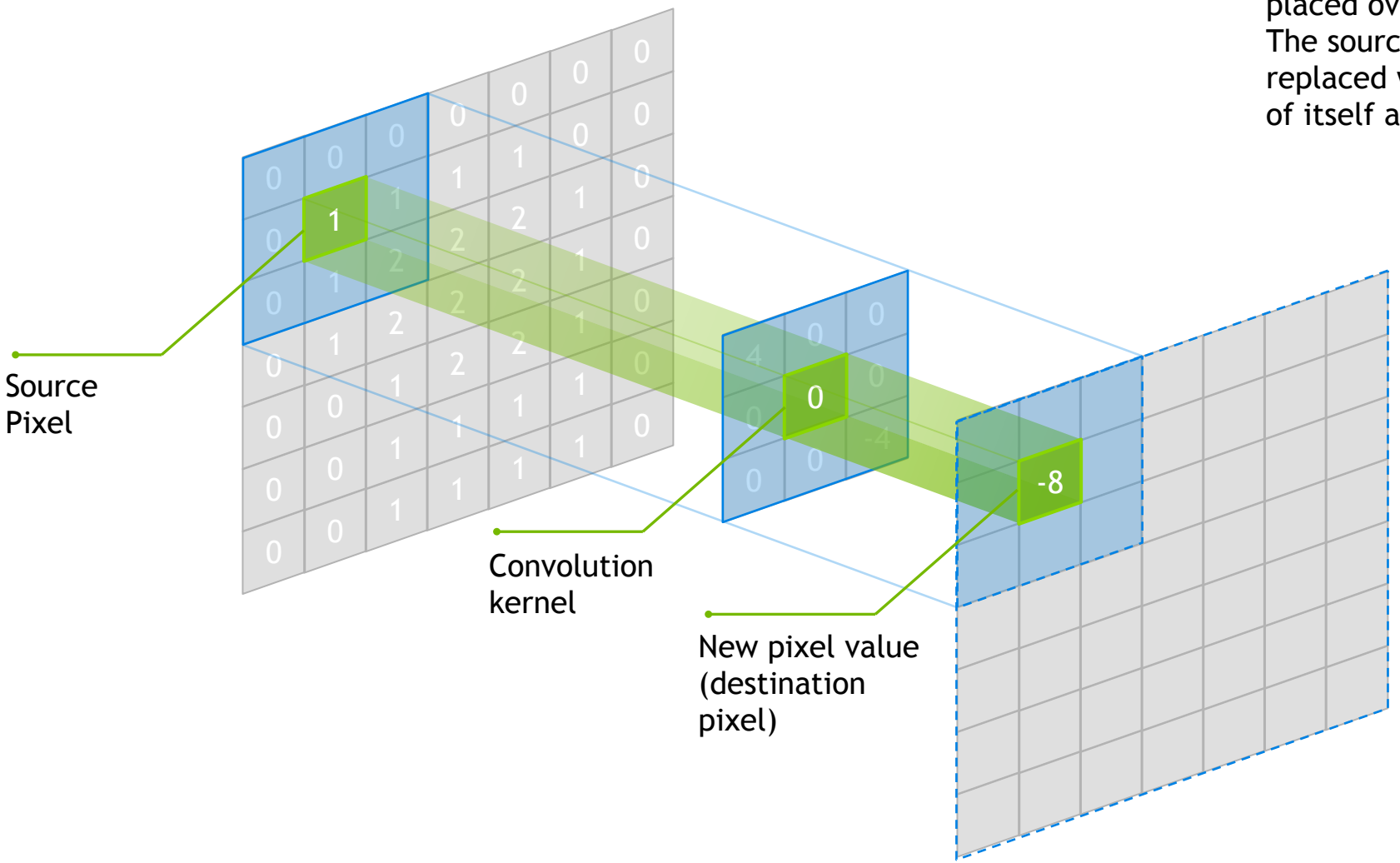




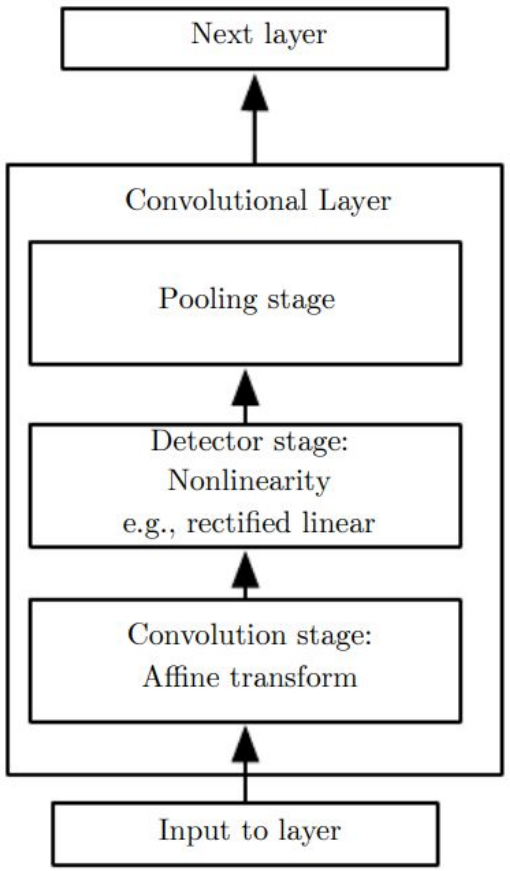


CONVOLUTION

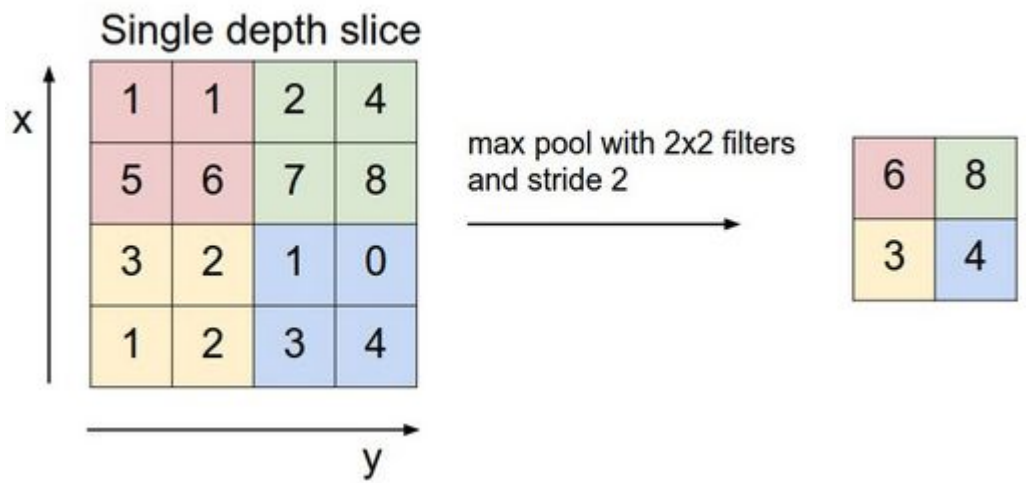
Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.



CNN DESIGN

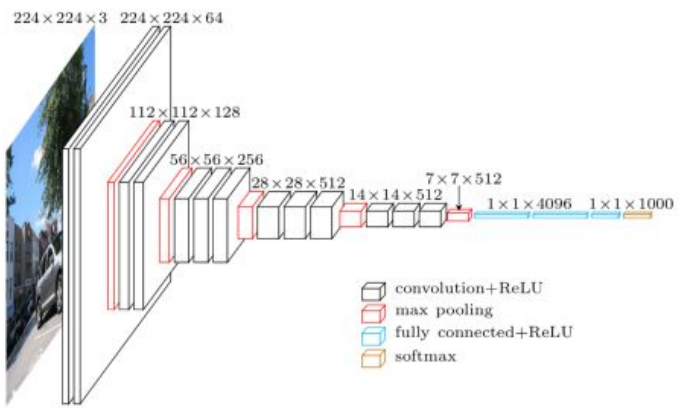


Pooling layers help to make the representation become approximately invariant to small translations of the input

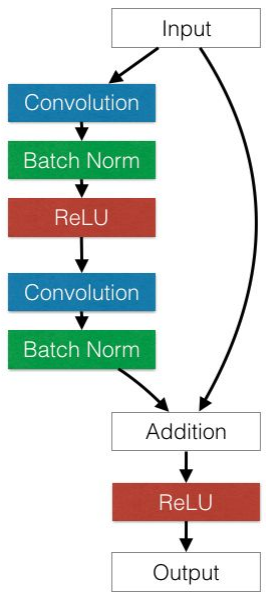


CNN DESIGN: EXAMPLES

VGG16



Residual block of ResNet



V-Net

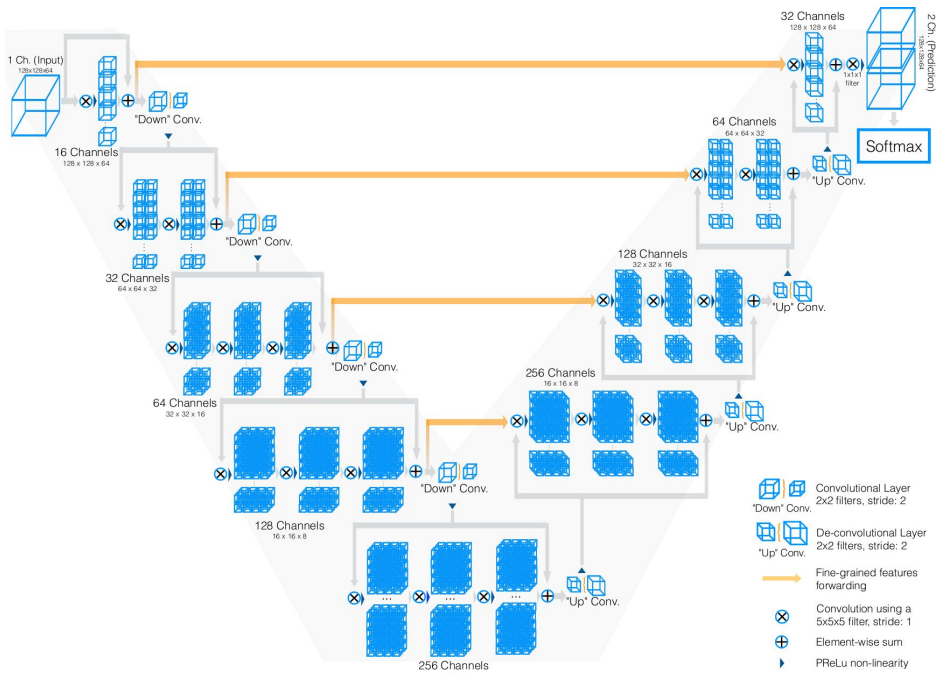
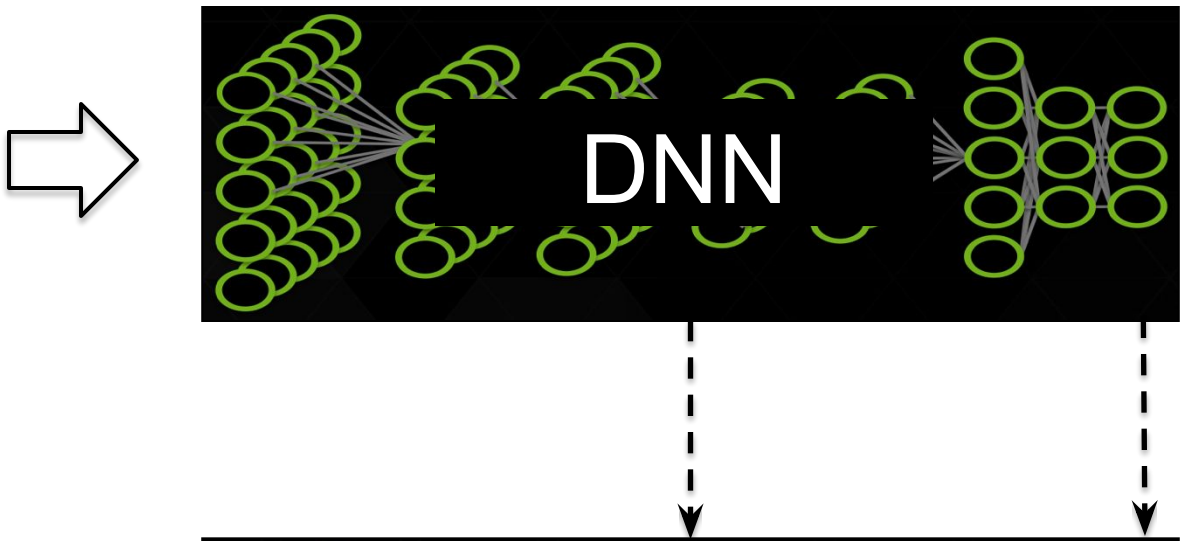


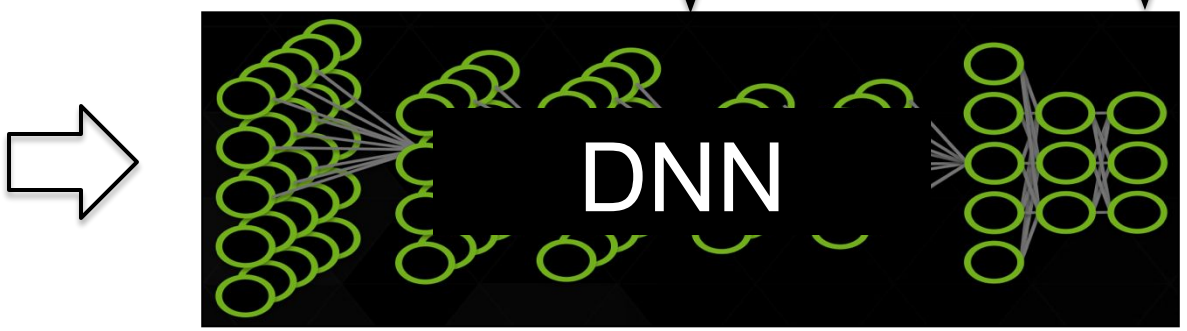
IMAGE CLASSIFICATION

Train:



Dog ✓
Cat ✓
Raccoon ✗

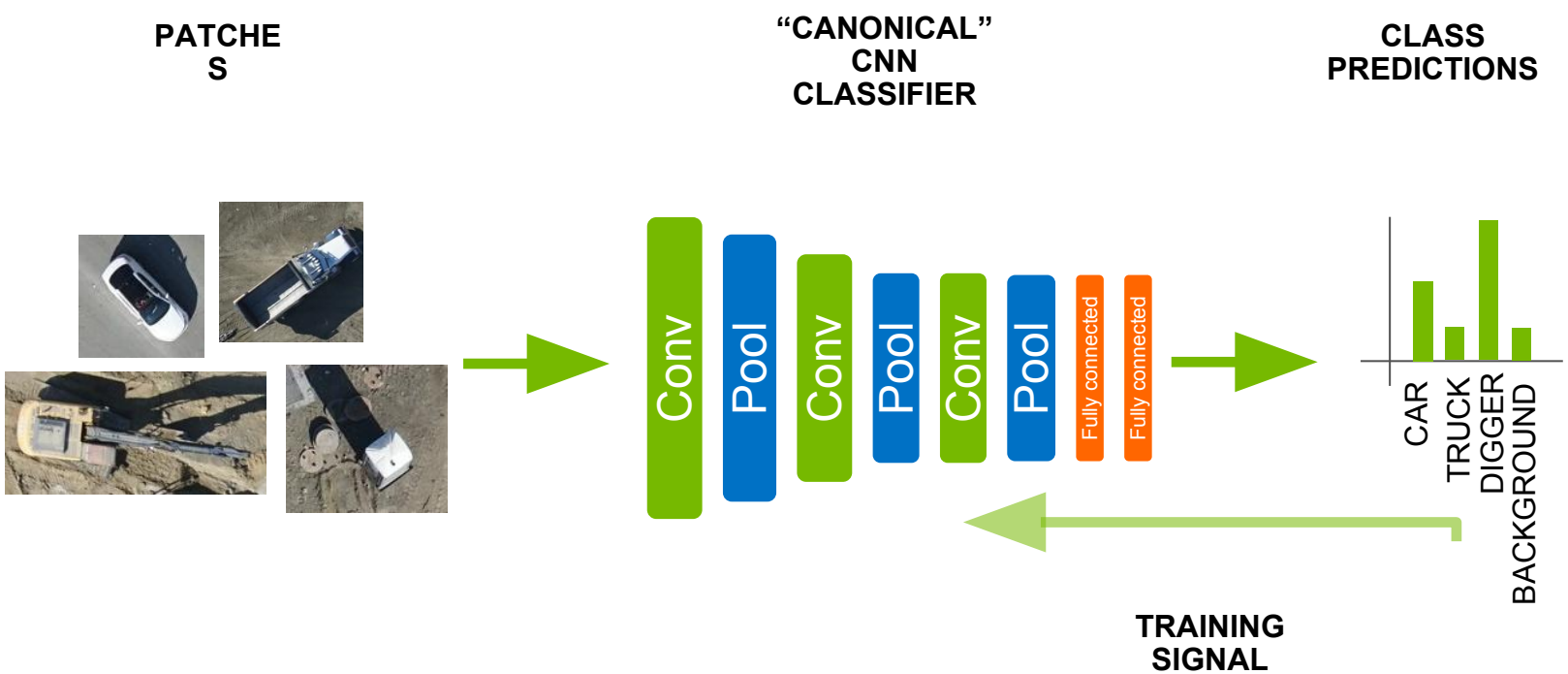
Deploy:



Dog ✓

SEGMENTATION: APPROACH 1

Patch based CNN classifier

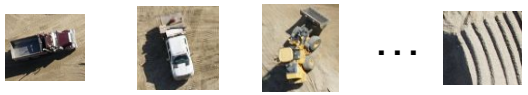


SEGMENTATION: APPROACH 1

Candidate generator feeding CNN classifier

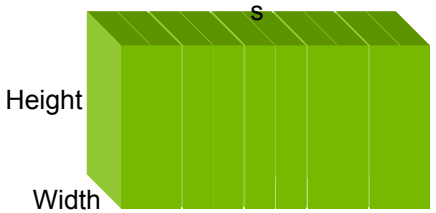
1. RAW IMAGE

2. GENERATE CANDIDATE DETECTIONS OR "BRUTE FORCE" SLIDING WINDOW

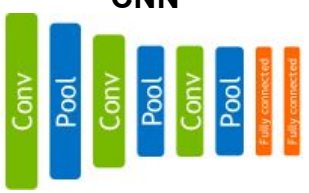


3. EXTRACT PATCHES

4. CREATE MINIBATCH



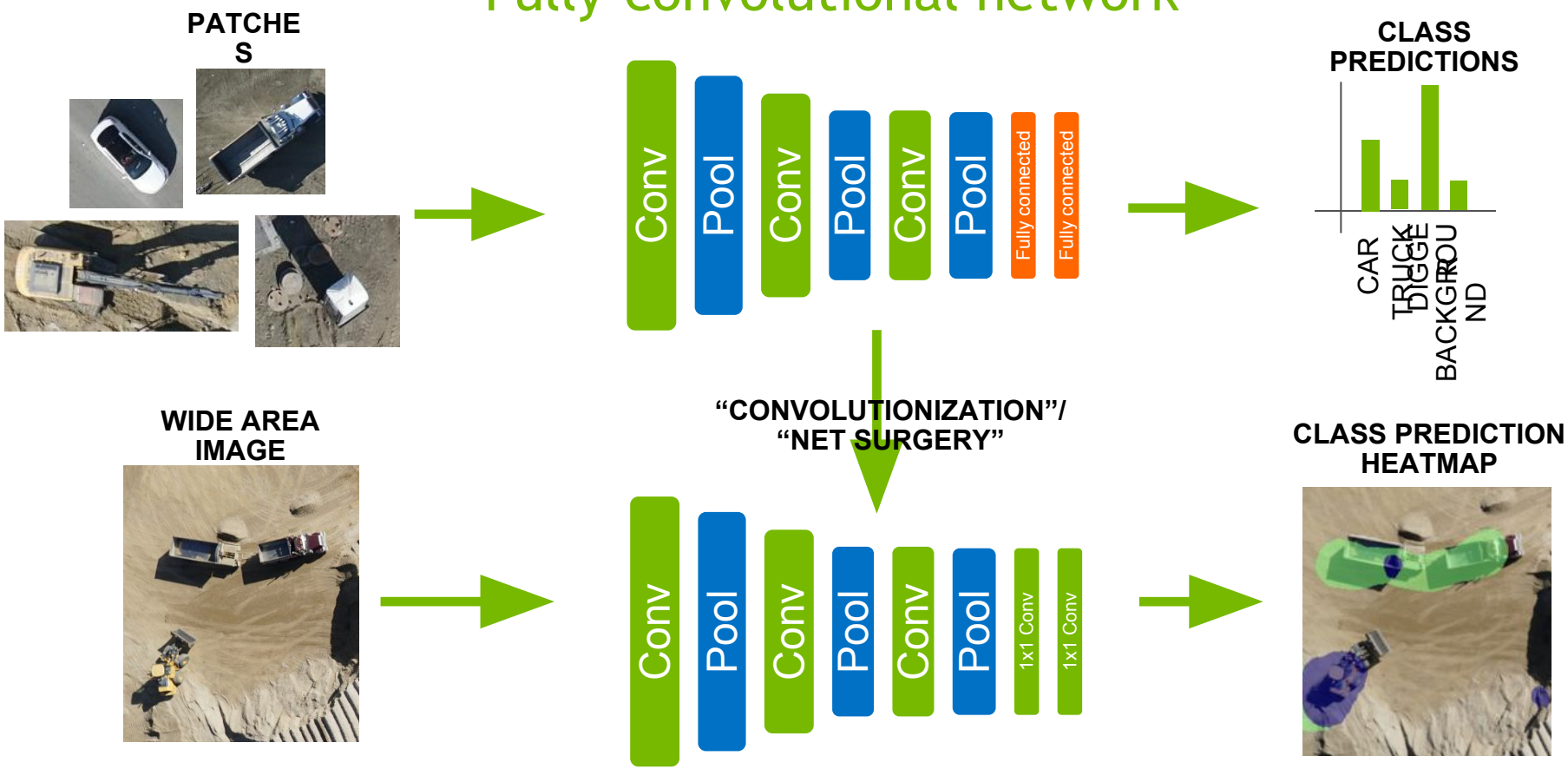
5. FEEDFORWARD THRU CNN



6. FILTER BOUNDING BOXES

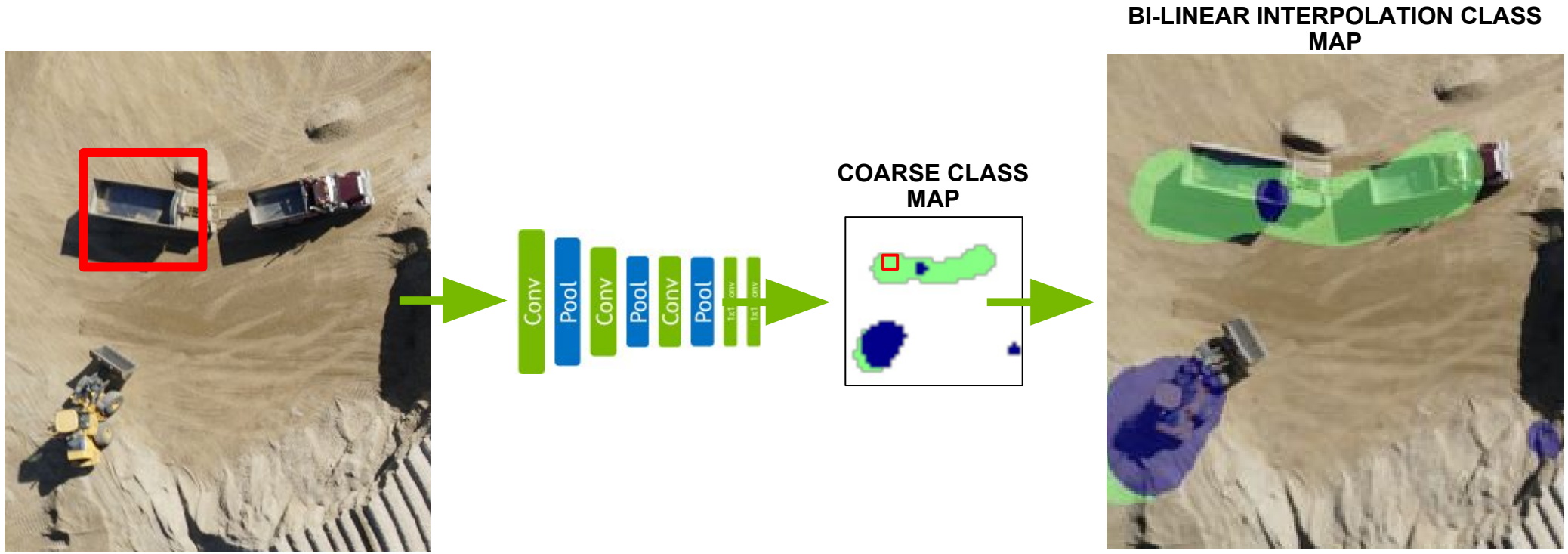
SEGMENTATION: APPROACH 2

Fully-convolutional network



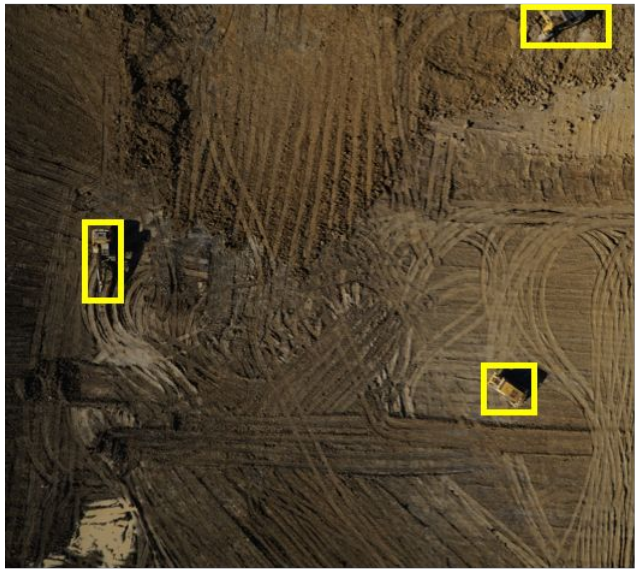
SEGMENTATION: APPROACH 2

Fully-convolutional network

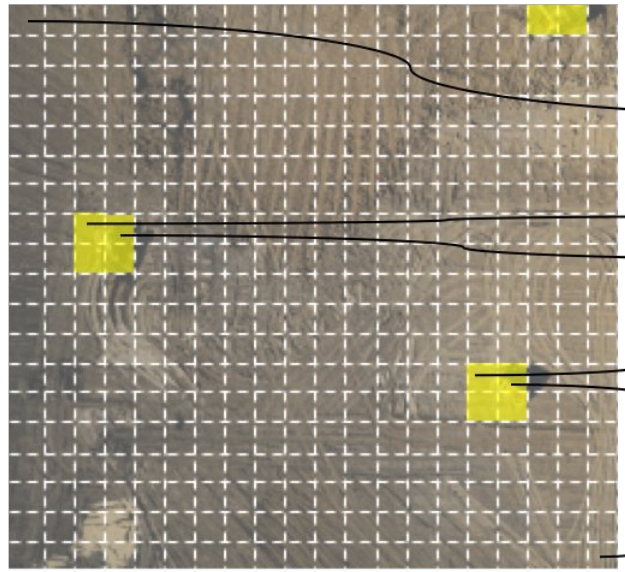


SEGMENTATION: APPROACH 3

Train on wide-area images with bounding box annotations



Training image with bounding box annotations



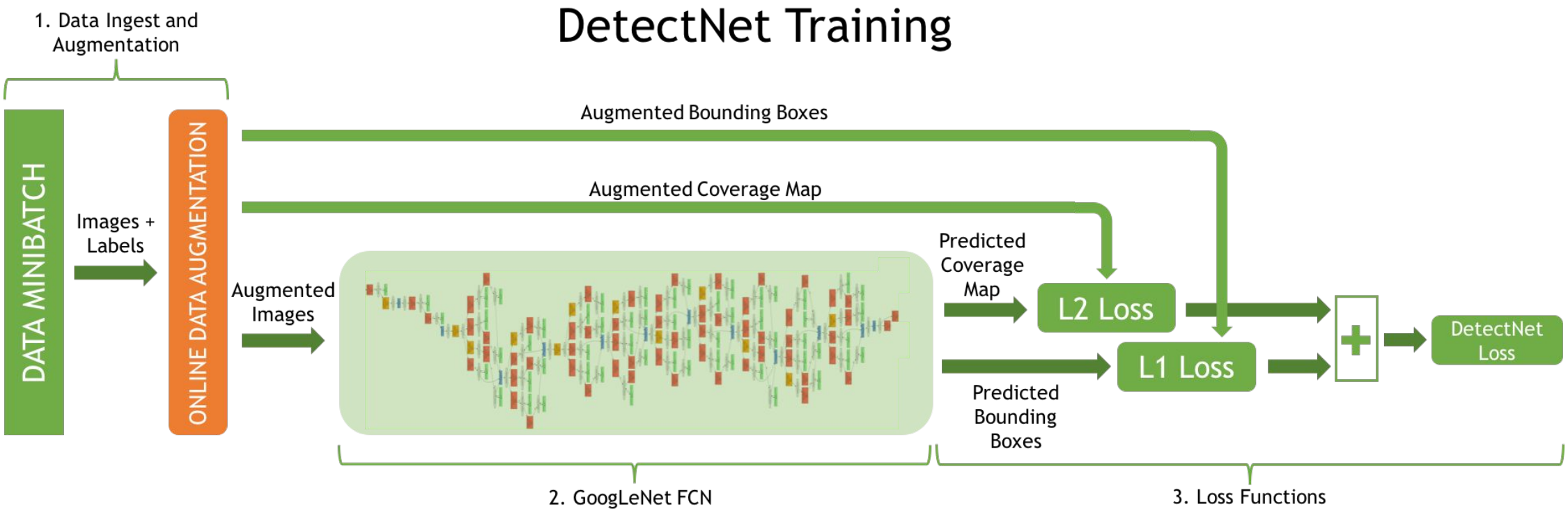
Bounding boxes mapped to grid squares

Bounding box coordinates in pixels relative to center of grid square

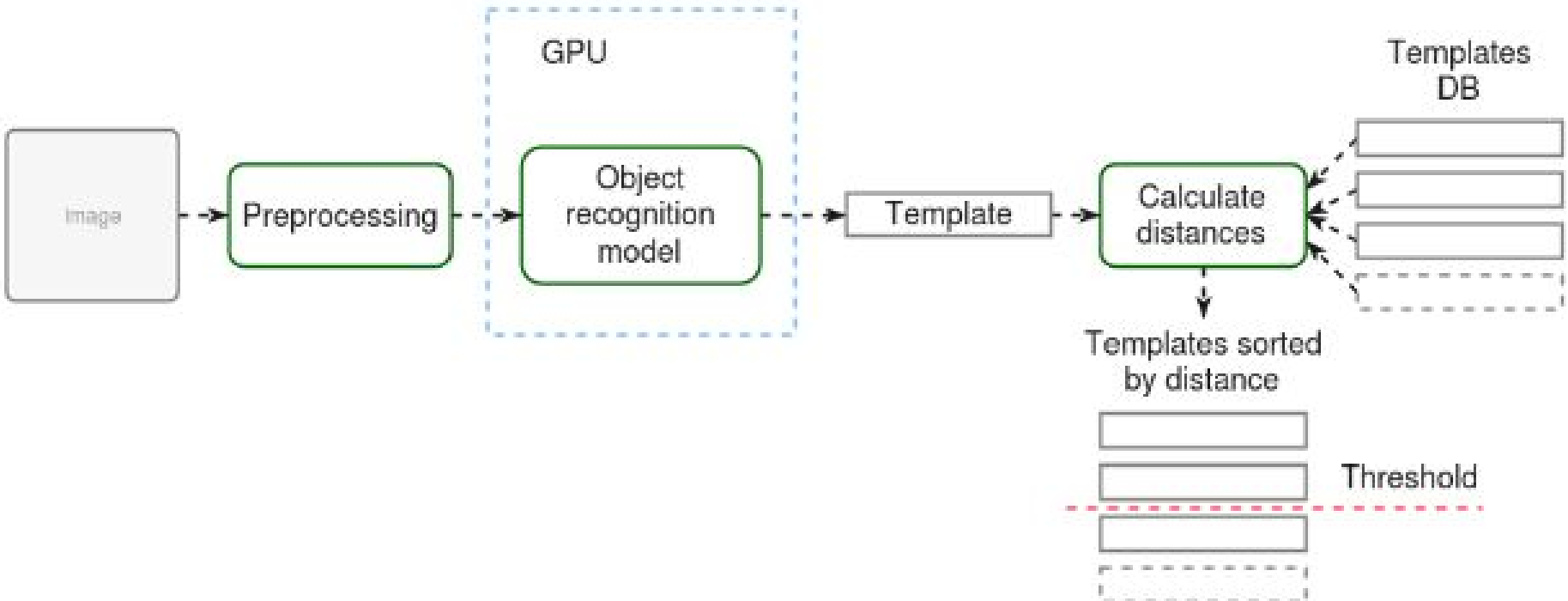
class	x_1	y_1	x_2	y_2	coverage
dontcare	0	0	0	0	0
…	…	…	…	…	…
digger	-2	-8	18	24	1
digger	-18	-8	2	24	1
…	…	…	…	…	…
digger	-6	-8	22	24	1
digger	-24	-8	8	24	1
…	…	…	…	…	…
dontcare	0	0	0	0	0

DetectNet input data representation

SEGMENTATION: APPROACH 3



FACE RECOGNITION

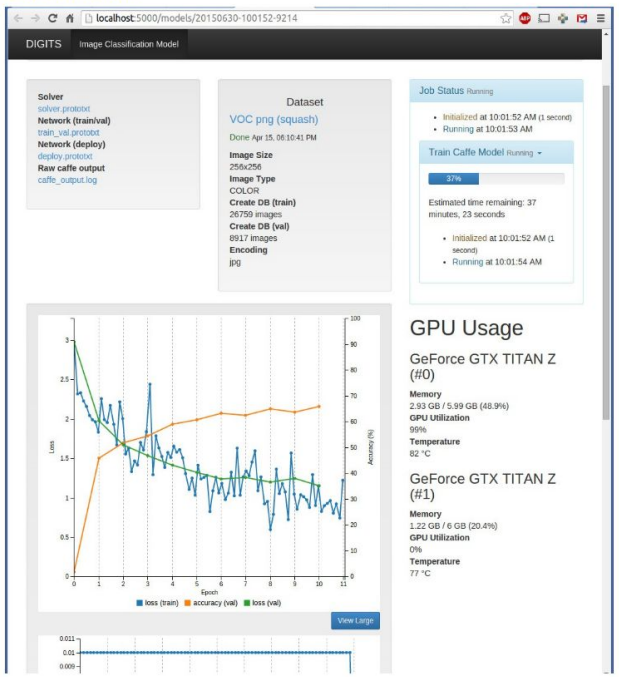


DEEP LEARNING SOFTWARE

NVIDIA DIGITS™

Interactively manage data and train deep learning models for image classification without the need to write code.

Learn more



Deep Learning Frameworks

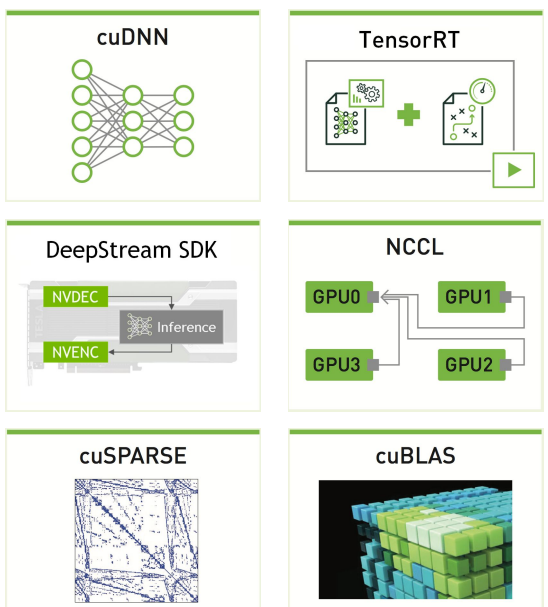
Design and train deep learning models using a high-level interface. Choose a deep learning framework that best suits your needs based on your choice of programming language, platform, and target application.

Learn more

Caffe2
Chainer
DL4J Deeplearning4j
MINERVA
mxnet
Purine
KERAS
Microsoft Cognitive Toolkit
MatConvNet
TensorFlow
theano
torch

NVIDIA Deep Learning SDK

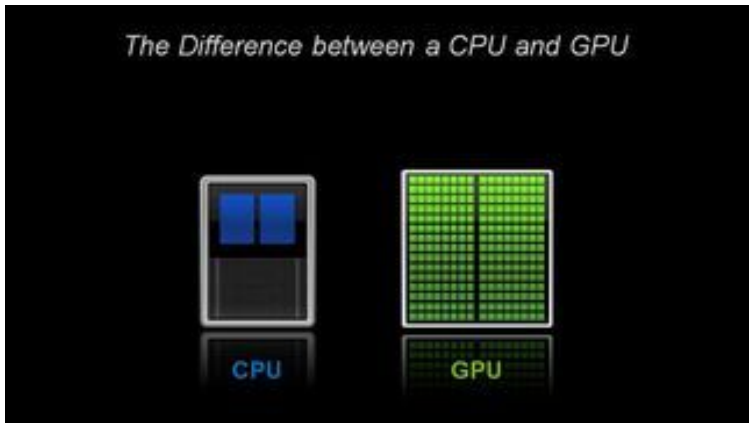
This SDK delivers high- performance multi-GPU acceleration and industry-vetted deep learning algorithms, and is designed for easy drop-in acceleration for deep learning frameworks.



developer.nvidia.com/deep-learning

HARDWARE

GPU Architecture



Costs comparison (2015):

Cost per GFLOP for CPU set = $0.51

Cost per GFLOP for GPU = $0.102

END-TO-END PRODUCT FAMILY

TRAINING

FULLY INTERGRATED DL SUPERCOMPUTER

DGX-1 & DGX Station

DESKTOP

Titan X Pascal

DATA CENTER

Tesla P100
Tesla V100

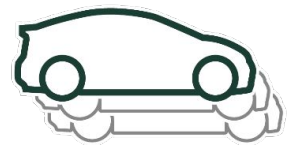
INFERENCE

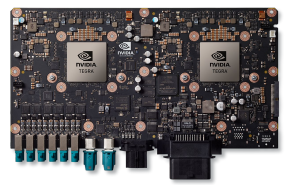
DATA CENTER

Tesla P100/V100

Tesla P4

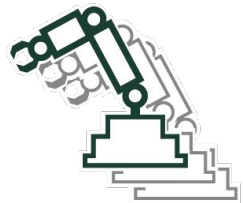
AUTOMOTIVE





Drive PX2

EMBEDDED



Jetson TX1

READY TO GET STARTED?

Project Checklist

1. What problem are you solving, what are the DL tasks?
2. What data do you have/need, and how is it labeled?
3. Which deep learning framework & tools will you use?
4. On what platform(s) will you train and deploy?

START SIMPLE, LEARN FAST

www.nvidia.com/dli