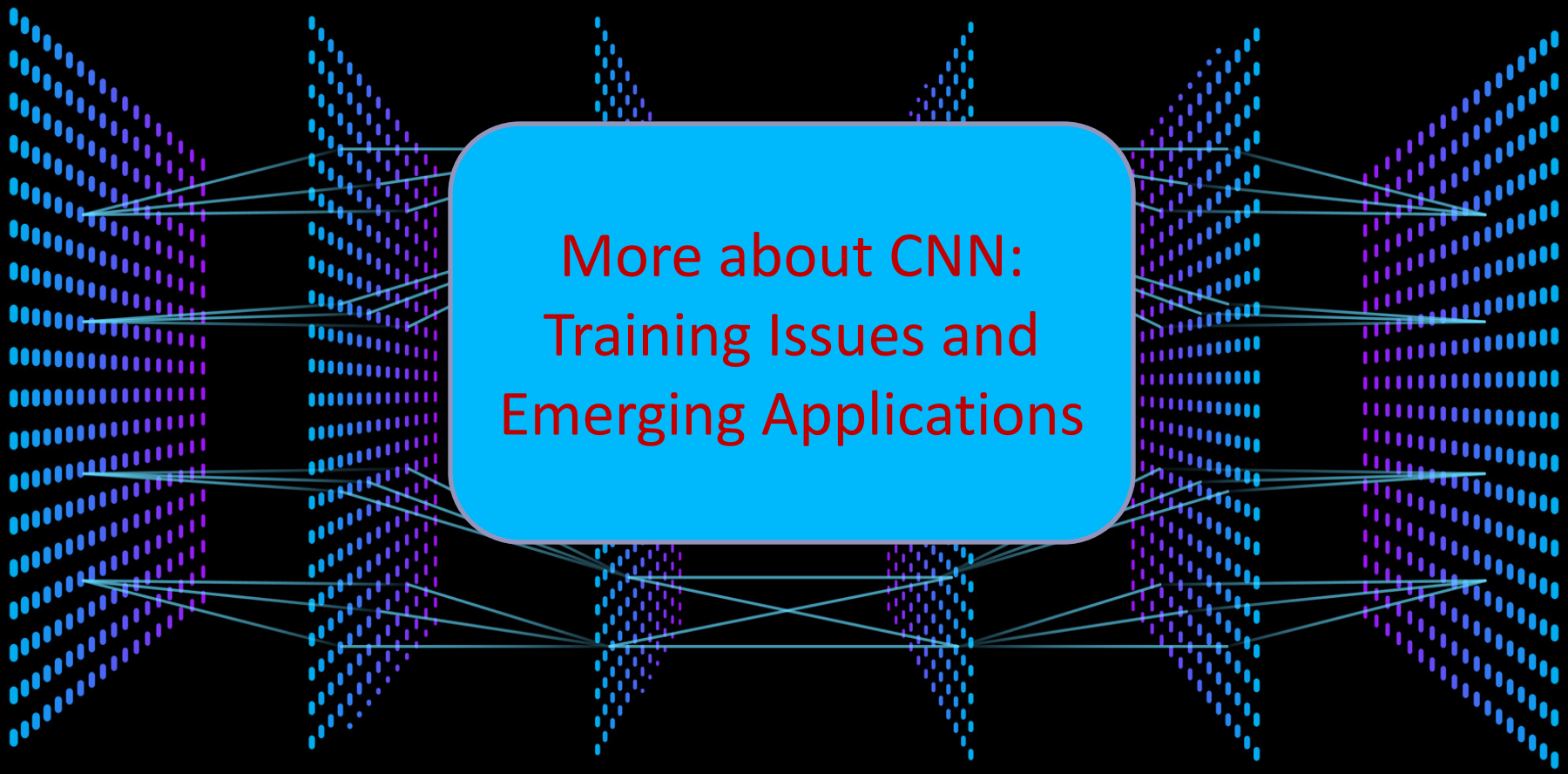
Deep Learning



Roadmap

- Training Issues
  - Why ReLU?
  - How to train CNN?
  - Derivative of max-pooling
- Emerging Applications
  - Medical
  - Image Processing
  - Self-Driving
- Take-home messages

Why ReLU? Why not Sigmoid?

[Ref.: Vanishing and Exploding Gradient Problems]

ReLU (Rectified Linear Units) have recognized as an alternative activation function to the sigmoid function in neural networks and below are some of the related advantages:

- ReLU activations used as the activation function induce sparsity in the hidden units. Inputs into the activation function of values less than or equal to 0, results in an output value of 0. Sparse representations are considered more valuable.
- ReLU activations do not face gradient vanishing problem as with sigmoid and tanh function (c.f. close to zero gradients when input's magnitude is large, see derivative of sigmoid(x)).
- ReLU activations do not require any exponential computation (such as those required in sigmoid or tanh activations). This ensures faster training than sigmoids due to less numerical computation.

Training of CNN: Deep Network Training

- **Idea #1: (Just like a shallow network)**
  1. Supervised fine-tuning only
     - Compute the supervised gradient by backpropagation.
     - Take small steps in the direction of the gradient (SGD)

- **Idea #2: (Supervised pre-training)**
  1. Supervised layer-wise pre-training
  2. Supervised fine-tuning

- **Idea #3: (Unsupervised pre-training)**
  1. Unsupervised layer-wise pre-training
  2. Supervised fine-tuning

Idea #1:
No pre-training, just like a shallow network

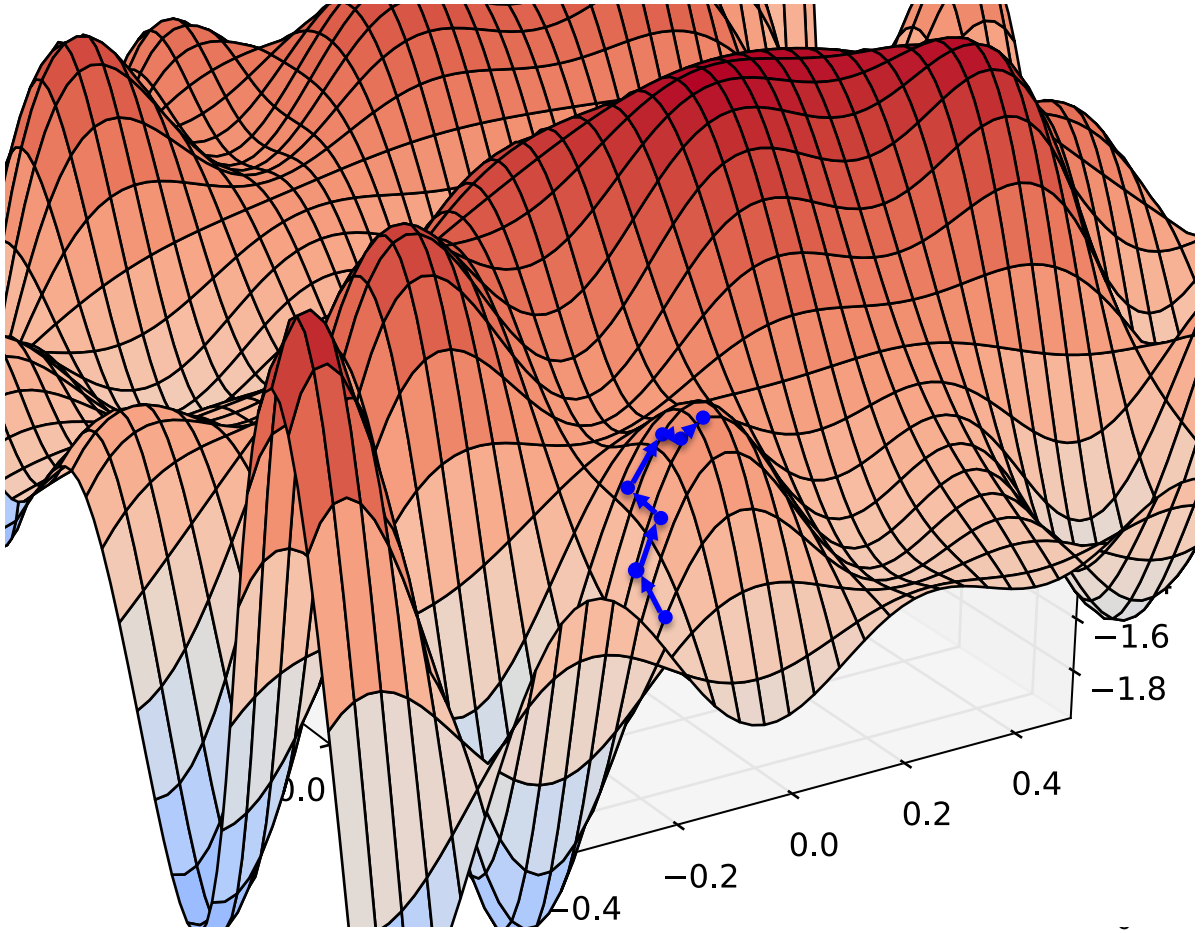
- What goes wrong?
  A. Gets stuck in local optima (minima/maxima)
     - Nonconvex objective function
     - Usually start at a random (bad) point in parameter space
  B. Gradient is progressively getting more dilute
     - “Vanishing gradients”

Problem A: *Nonconvexity*

Stochastic Gradient Descent (Ascent)…

…climbs to the top of the nearest hill…

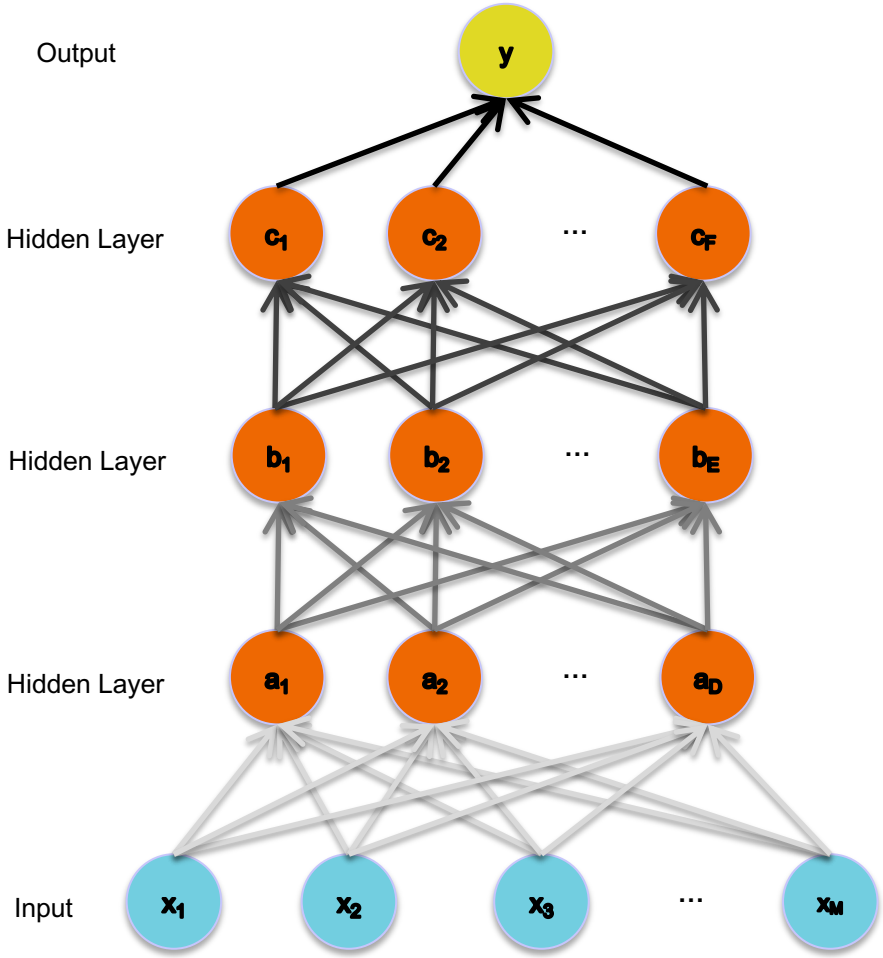
…which might not lead to the top of the mountain



Problem B: *Vanishing Gradients*

The gradient for an edge (weight) at the base of the network depends on the gradients of many edges above it.

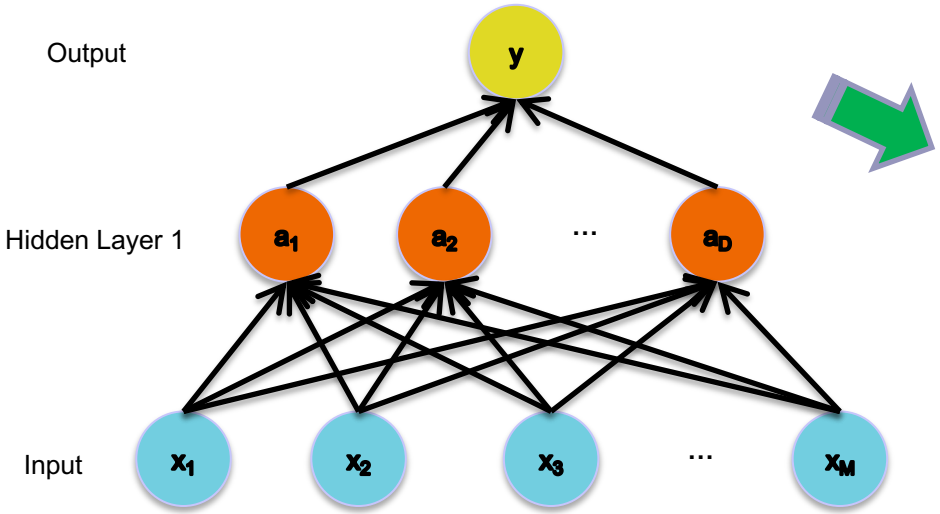
The chain rule multiplies many of these partial derivatives (which are typically pretty close to zero) together.

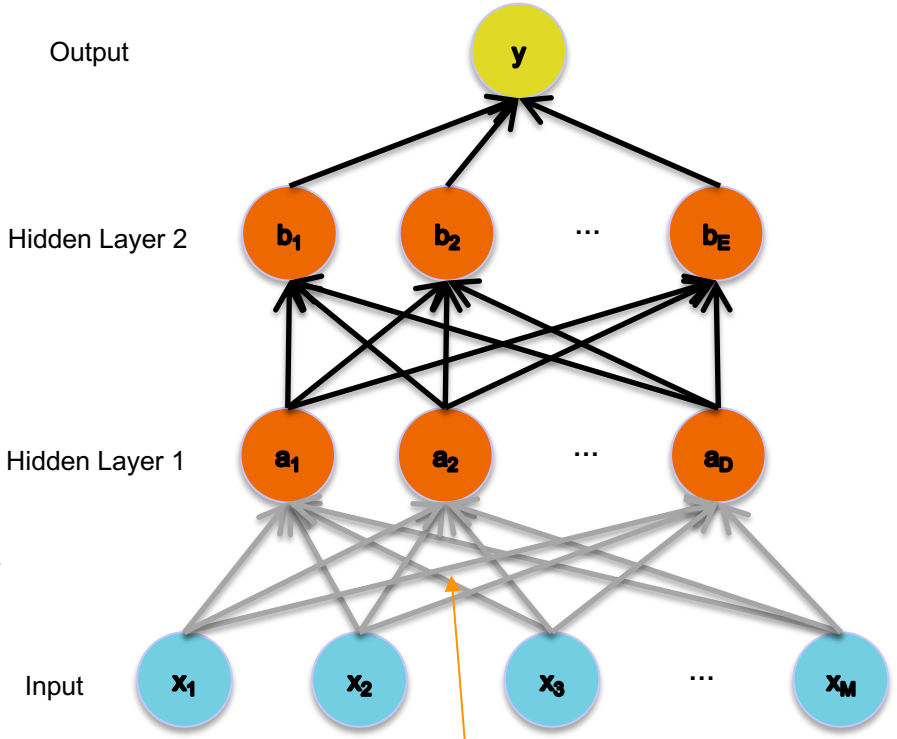


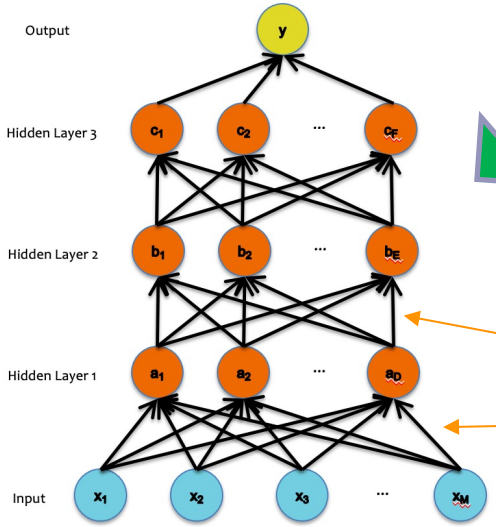
Idea #2:
Supervised pre-training (with two steps)

1. Supervised Pre-training
   - Use **labeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following “Idea #1”
   - Refine the features by backpropagation so that they become tuned to the end-task

Supervised Pre-training







Finally, finetune ALL parameters!

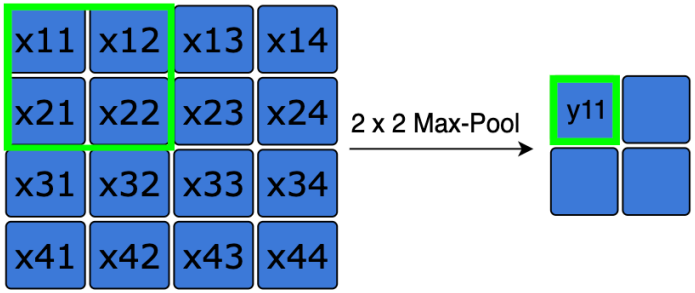
Parameters fixed!

Idea #3: Unsupervised pre-training

1. Unsupervised Pre-training
   - Use **unlabeled** data
   - Work bottom-up
     - Train hidden layer 1. Then fix its parameters.
     - Train hidden layer 2. Then fix its parameters.
     - …
     - Train hidden layer n. Then fix its parameters.
2. Supervised Fine-tuning
   - Use **labeled** data to train following “Idea #1”
   - Refine the features by backpropagation so that they become tuned to the end-task

Derivative of max-pooling -

How to backpropagate through max-pooling layers? [Ref:Reference 1 and Reference 2]



Since the max-pooling layer doesn't have any weights, we need to find the gradient of the error with respect to the input matrix only for backpropagation.

$$\begin{bmatrix} \frac{\partial E}{\partial x_{11}} & \frac{\partial E}{\partial x_{12}} & \frac{\partial E}{\partial x_{13}} & \frac{\partial E}{\partial x_{14}} \\ \frac{\partial E}{\partial x_{21}} & \frac{\partial E}{\partial x_{22}} & \frac{\partial E}{\partial x_{23}} & \frac{\partial E}{\partial x_{24}} \\ \frac{\partial E}{\partial x_{31}} & \frac{\partial E}{\partial x_{32}} & \frac{\partial E}{\partial x_{33}} & \frac{\partial E}{\partial x_{34}} \\ \frac{\partial E}{\partial x_{41}} & \frac{\partial E}{\partial x_{42}} & \frac{\partial E}{\partial x_{43}} & \frac{\partial E}{\partial x_{44}} \end{bmatrix}$$

y11 = max (x11, x12, x21, x22)

By applying the chain rule, we have

$$\frac{\partial E}{\partial x_{11}} = \frac{\partial E}{\partial y_{11}} \cdot \frac{\partial y_{11}}{\partial x_{11}} + \frac{\partial E}{\partial y_{12}} \cdot \overbrace{\frac{\partial y_{12}}{\partial x_{11}}}^{0} + \frac{\partial E}{\partial y_{21}} \cdot \overbrace{\frac{\partial y_{21}}{\partial x_{11}}}^{0} + \frac{\partial E}{\partial y_{22}} \cdot \overbrace{\frac{\partial y_{22}}{\partial x_{11}}}^{0}$$

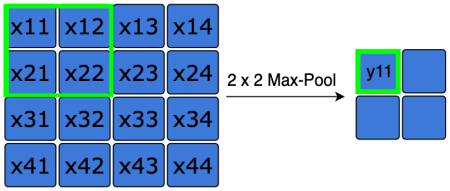
Since $y_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$, $\frac{\partial y_{11}}{\partial x_{11}}$ is nonzero only if x_{11} is the maximum valued feature in the kernel, let's assume that $x_{11} \neq \max(x_{11}, x_{12}, x_{21}, x_{22})$, and then

$$\frac{\partial E}{\partial x_{11}} = 0$$

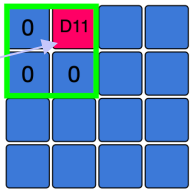
If $x_{12} = \max(x_{11}, x_{12}, x_{21}, x_{22})$, then $\frac{\partial y_{11}}{\partial x_{12}} = 1$ and

$$\frac{\partial E}{\partial x_{12}} = \frac{\partial E}{\partial y_{11}}$$

defined $D_{11} = \frac{\partial E}{\partial y_{11}}$



y11 = max (x11, x12, x21, x22) = x12



Backward Pass

No weight updates! Just propagate the error to the input of max pooling layer.

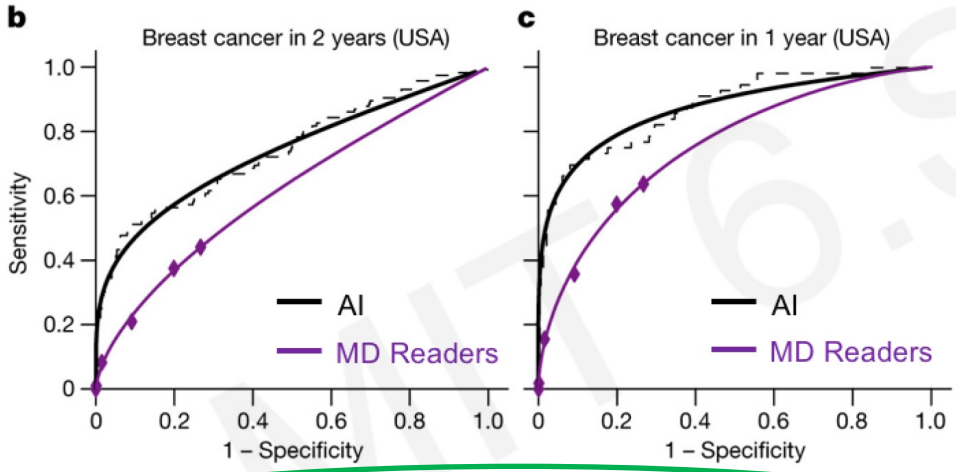
Emerging Applications of CNN

(extracted from http://introtodeeplearning.com/)

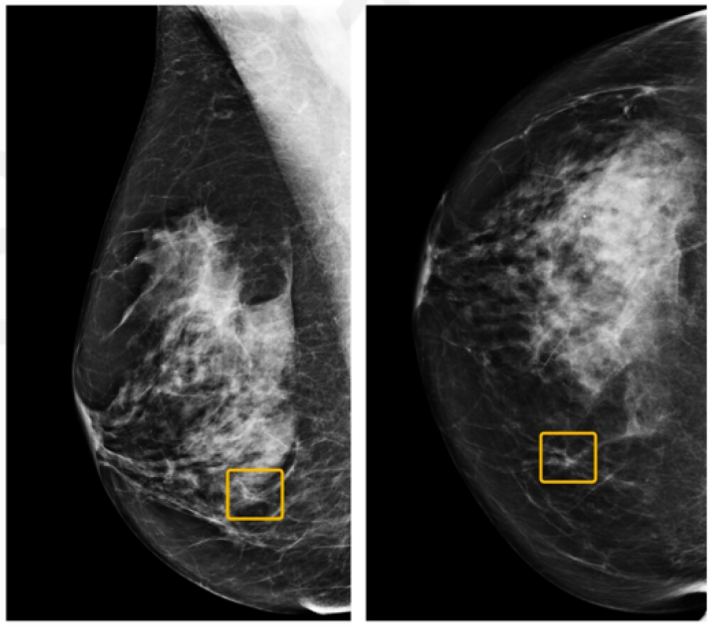
Medical Applications:
International evaluation of an AI system for breast cancer screening

Detection: Breast Cancer Screening

International evaluation of an AI system for breast cancer screening nature



CNN-based system outperformed expert radiologists at detecting breast cancer from mammograms



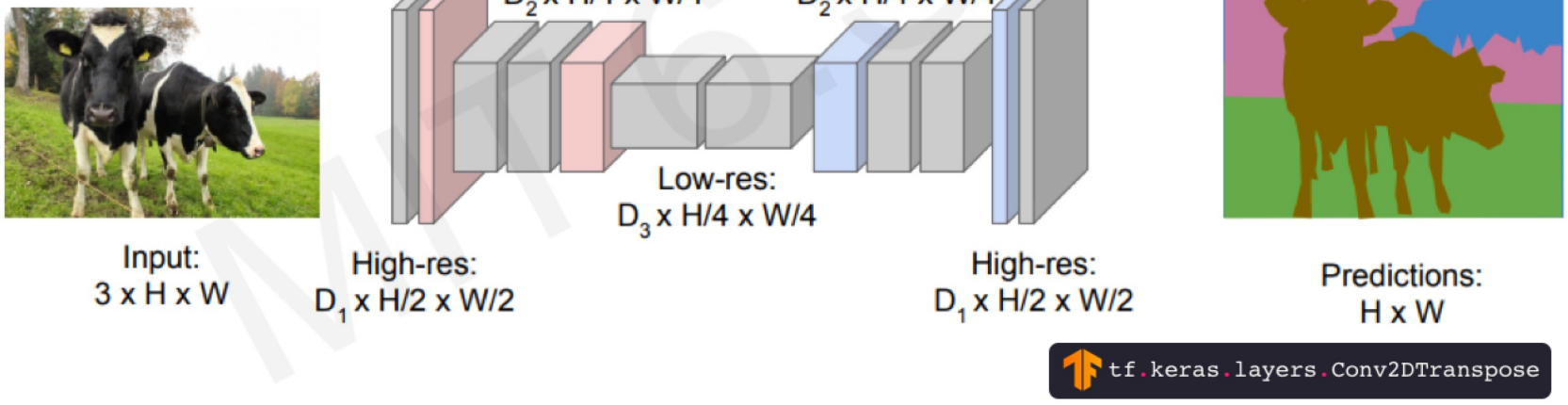
Breast cancer case missed by radiologist but detected by AI

Image Processing Applications: Semantic Segmentation

Original paper: Click here
Github link: Click here

Semantic Segmentation: Fully Convolutional Networks

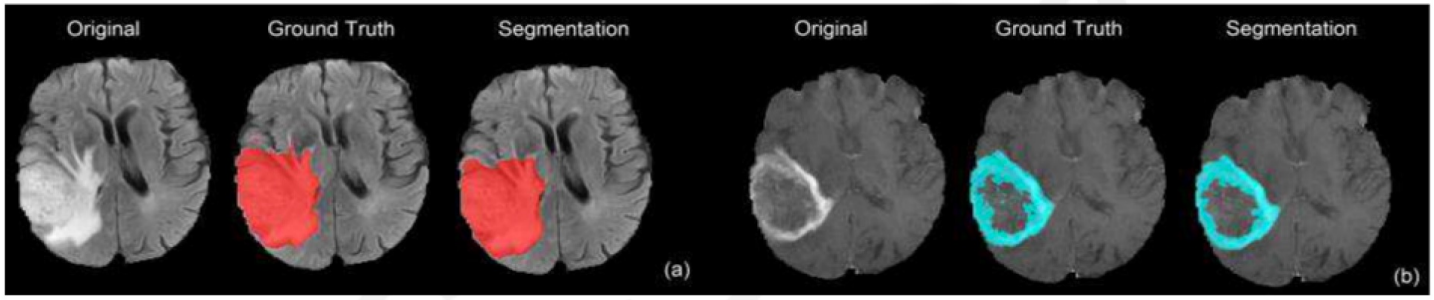
FCN: Fully Convolutional Network.
Network designed with all convolutional layers,
with **downsampling** and **upsampling** operations



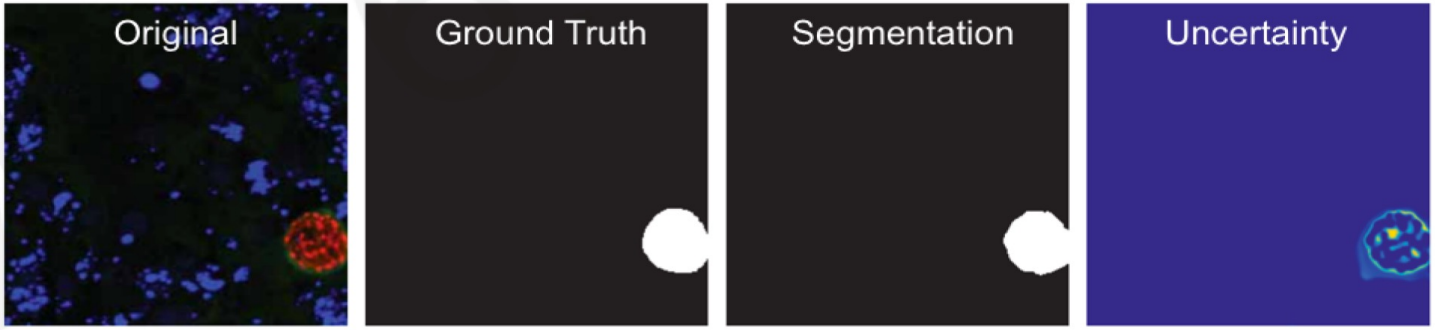
Medical Application Again: Biomedical Image Analysis

Semantic Segmentation: Biomedical Image Analysis

Brain Tumors
Dong+ *MIUA* 2017.

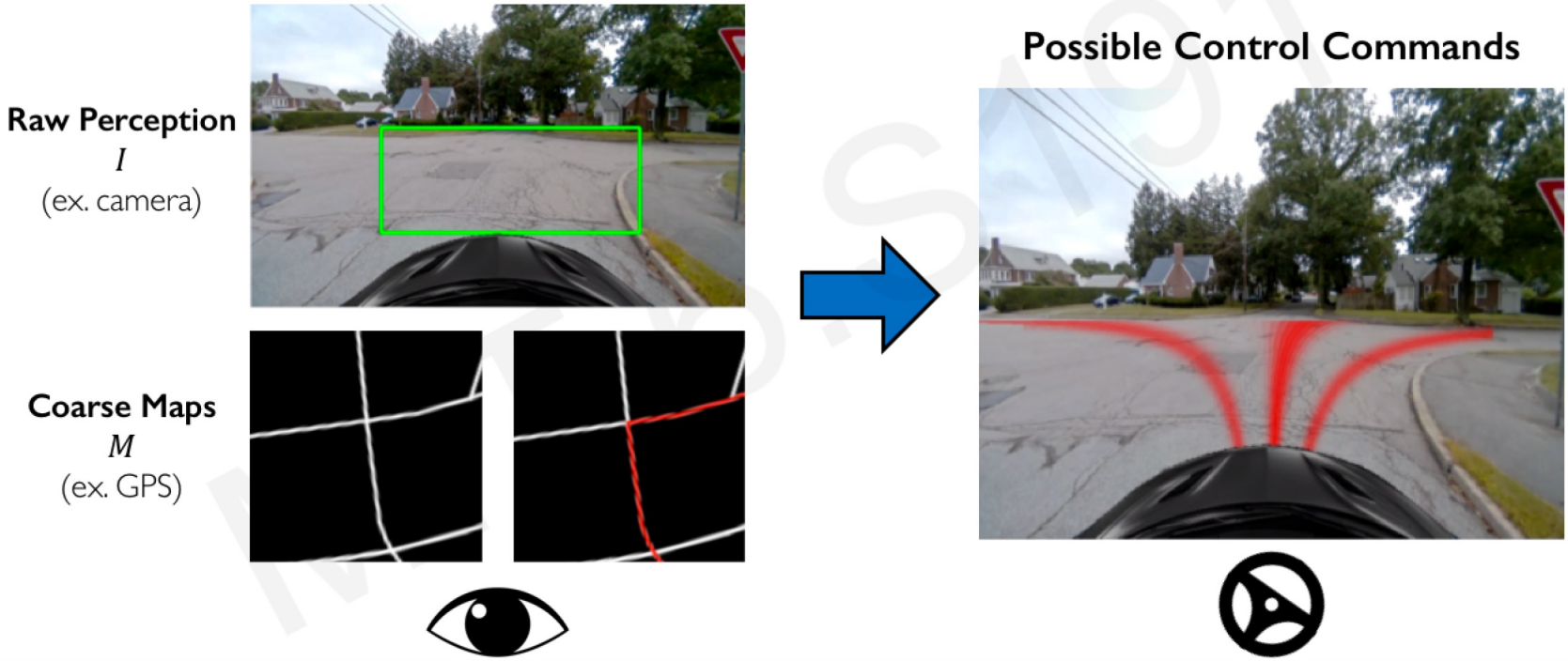


Malaria Infection
Soleimany+ *arXiv* 2019.



Self-Driving Applications

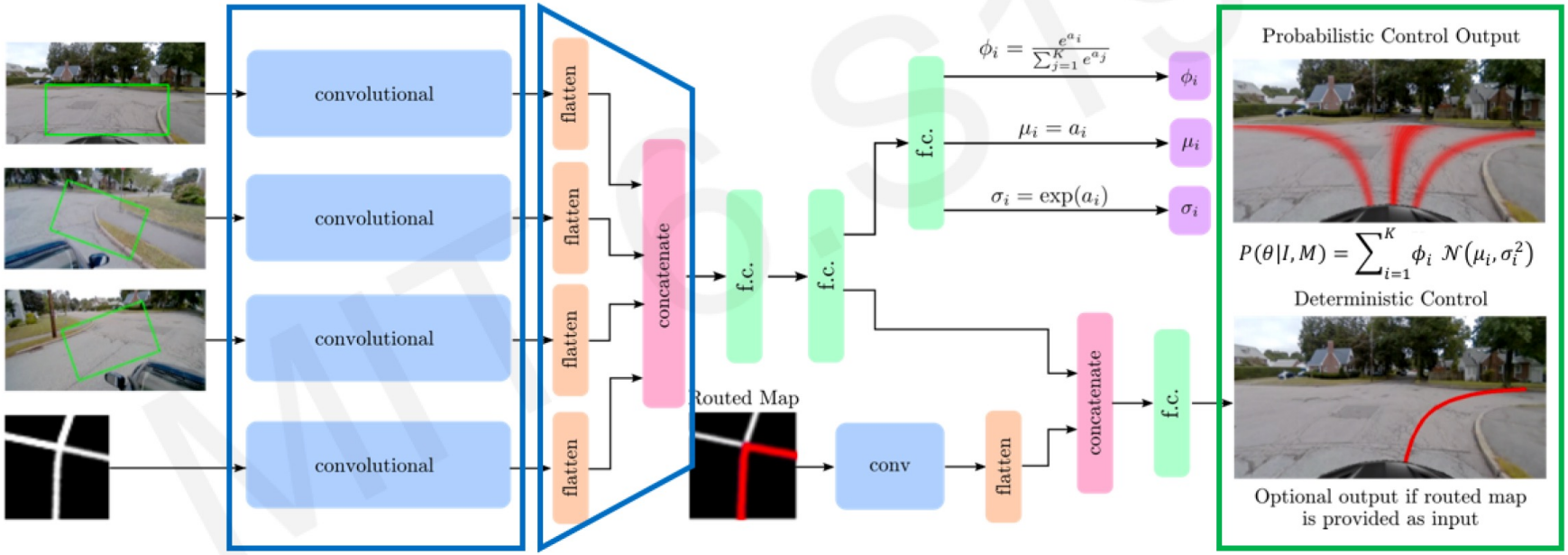
Self-Driving Cars: Navigation from Visual Perception



Self-Driving Applications

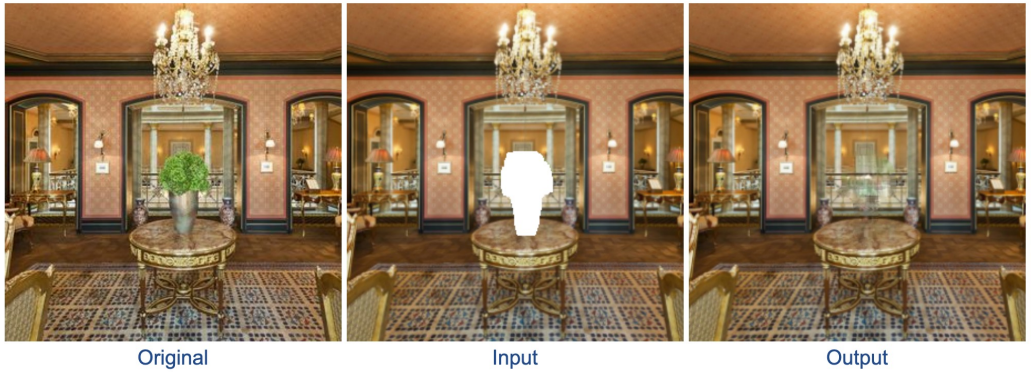
End-to-End Framework for Autonomous Navigation

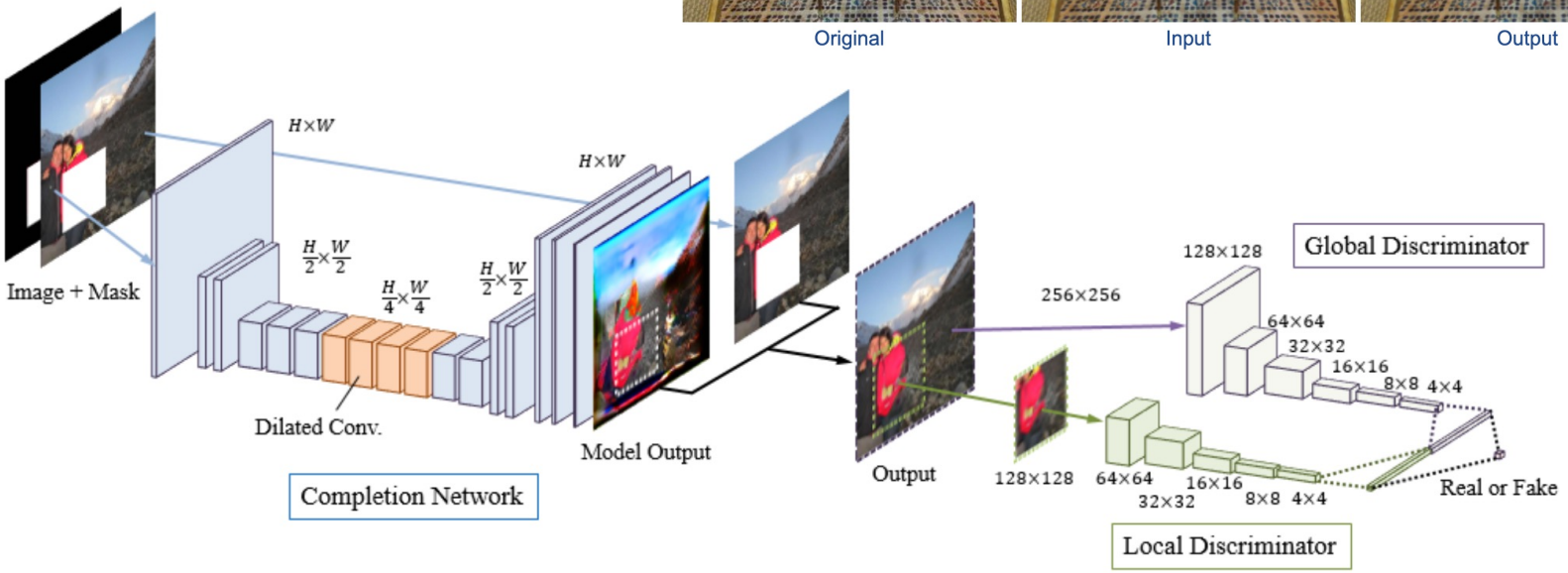
Entire model is trained end-to-end **without any human labelling or annotations**



Yet another type of interesting applications

- Remember week 1's Image completion work
  - http://iizuka.cs.tsukuba.ac.jp/projects/completion/en/





Take-home Messages

- Training from scratch of a CNN is tough because it involves so many weight parameters (e.g. 138M in VGGNet). Some tricks are needed, i.e., pre-training in a layer-by-layer manner.
- Yes, effectiveness of training is a big issue in deep learning!
- There exist a lot of applications of CNN!

Acknowledgement

- Slides from following sources:
  - https://cs231n.github.io/convolutional-networks
  - https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/
  - http://introtodeeplearning.com/
  - Ali Harakeh, U of Waterloo
- Images from Google search of Internet