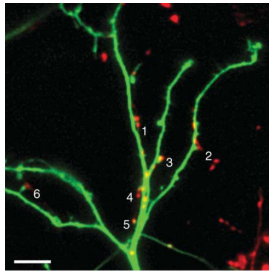
Machine Learning in Imaging

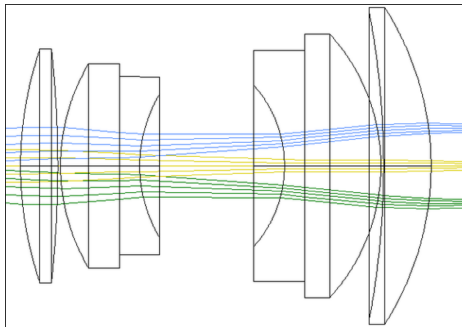
BME 590L
Roarke Horstmeyer

Lecture 19: Physical Layer Implementations and Troubleshooting

Summary of two models for image formation

- **Interpretation #1: Radiation (*Incoherent*)**
- Model: Rays



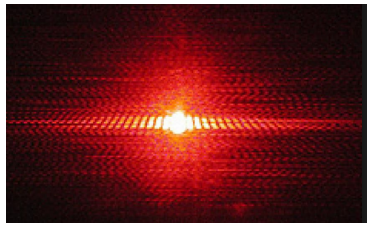


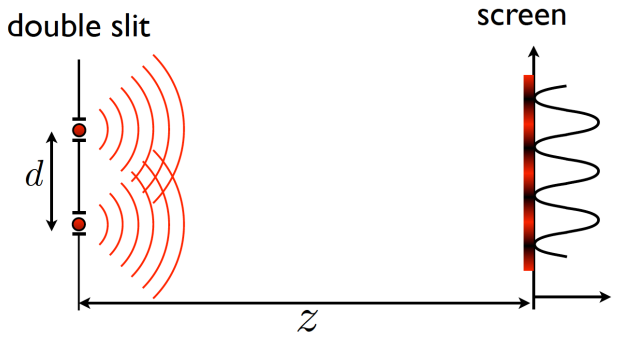
- Real, non-negative

$$I_s = H\,B\,S_0$$

- Sample absorption **S**
- Illumination brightness **B**
- Blur in **H**

- **Interpretation #2: Electromagnetic wave (*Coherent*)**
- Model: Waves





- Complex field

$$I_C = |\,H\,C\,S_C\,|^2$$

- Sample abs./phase **S**
- Illumination wave **B**
- Blur in **H**

Questions to address in this lecture

- Where and how should I implement my physical layer?
  - Simulation data
  - Experimental data
- How can I add some constraints to the physical weights that I’m optimizing?
- What are some common issues and pitfalls?

Physical Layers

Digital Layers

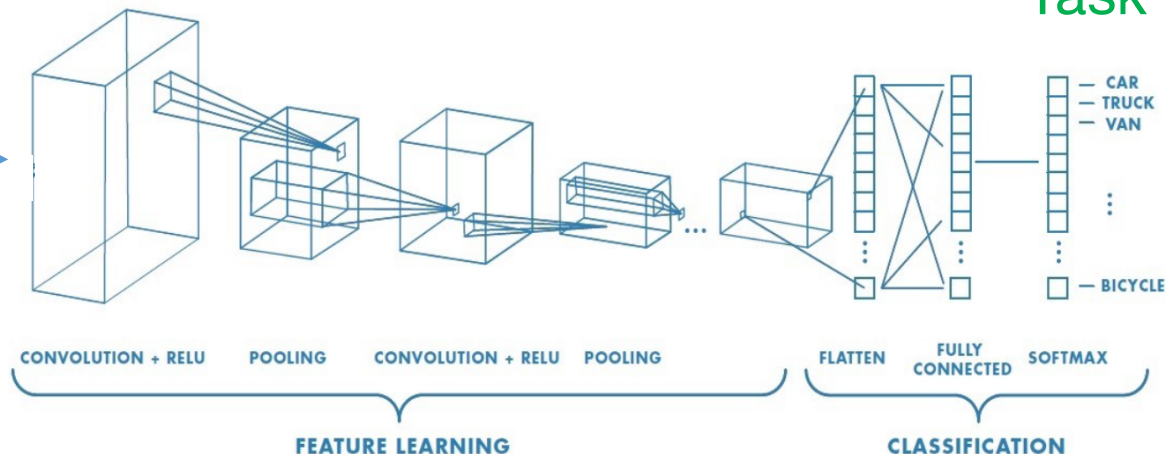
Input image data I_0

f[ ]

$I_s = f[I_0]$

Digitized I_s

Task



Some Examples:

- Optimized illumination
- Optimized sensor specifications
- Number of imagers and location
- Radiation dosage, biomarkers

Physical Layers

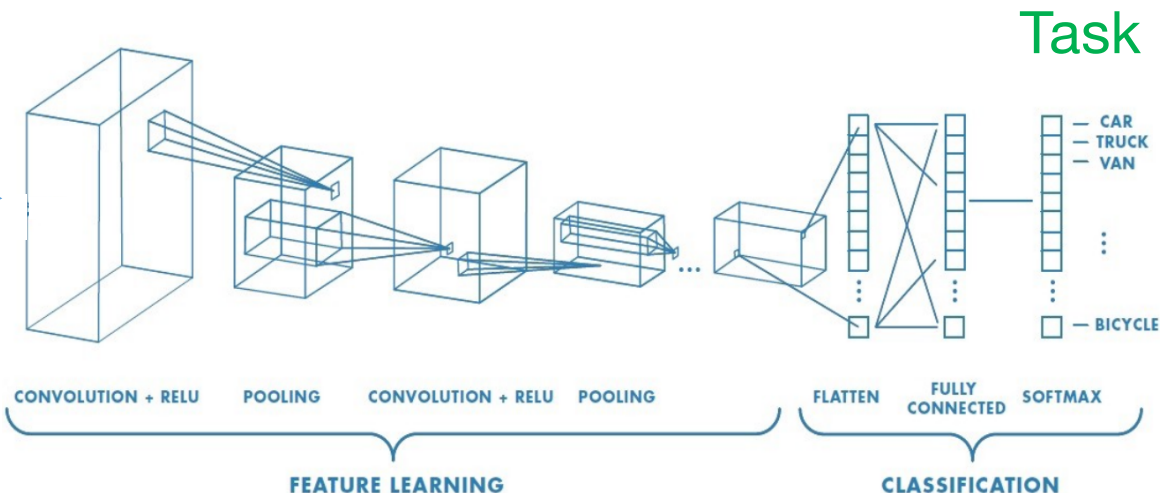
Digital Layers

Input image data I_0

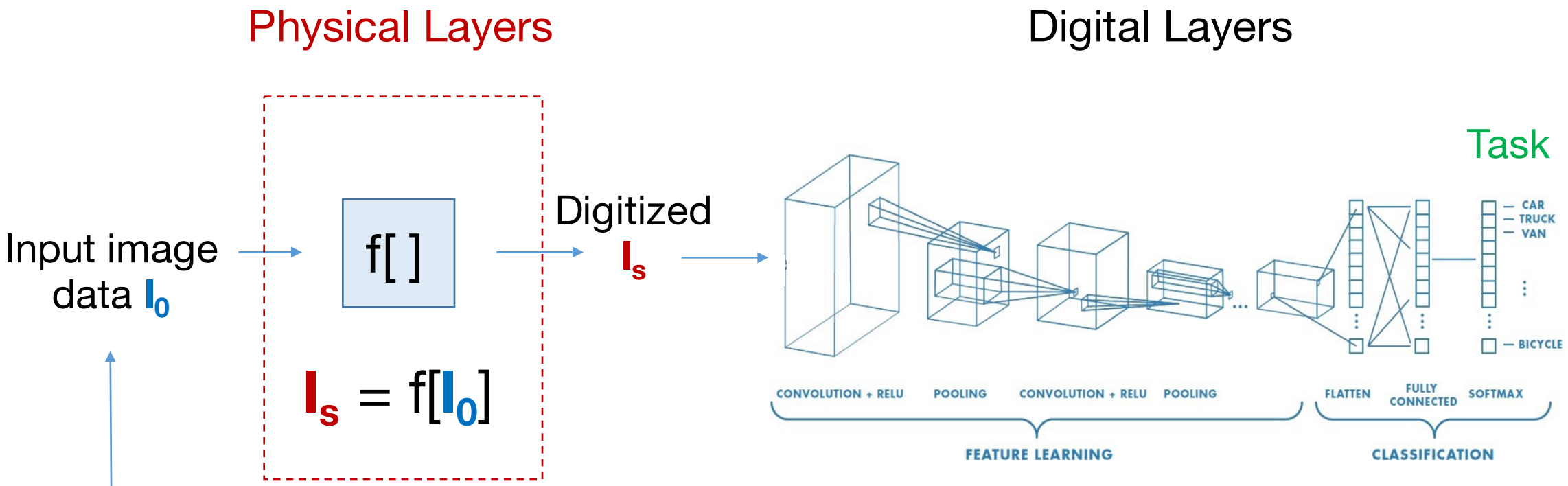
f[ ]

$I_s = f[I_0]$

Digitized I_s



Q: Where and how should I implement my physical layer?



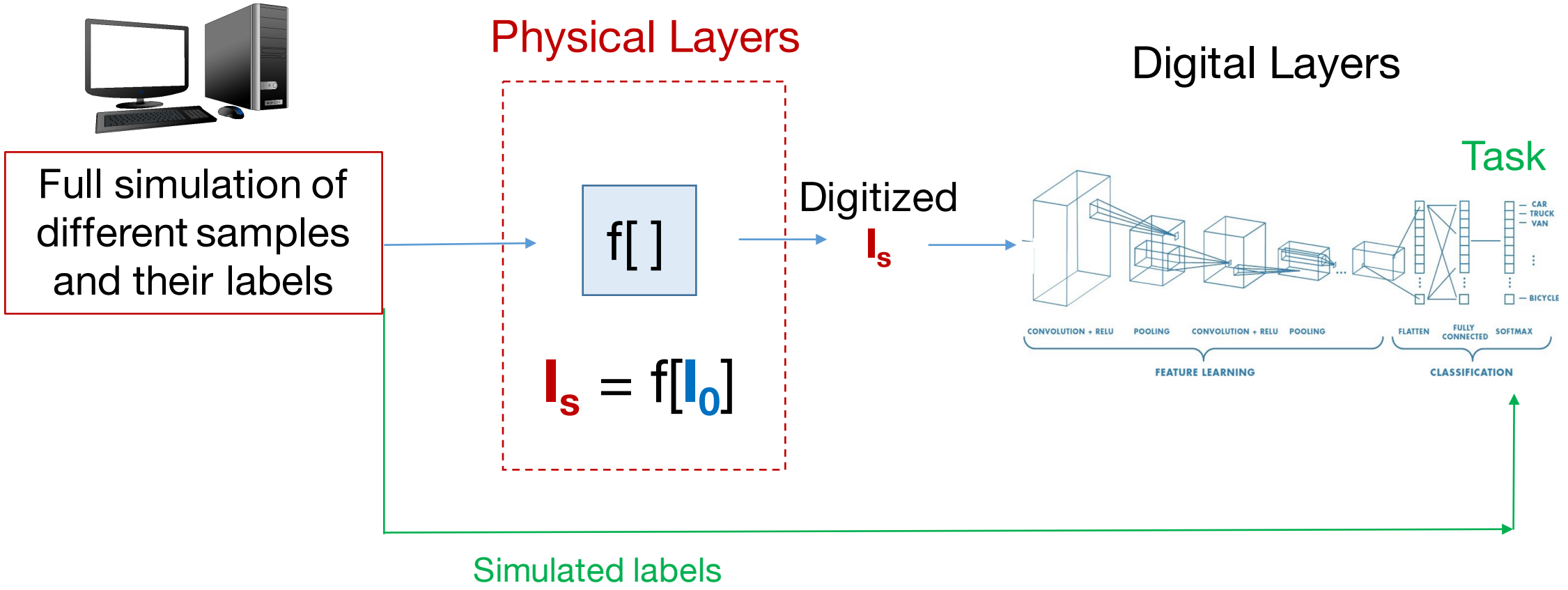
Q: Where and how should I implement my physical layer?

A: It depends on your data and implementation

- Situation #1: Fully simulated physical layers
- Situation #2: Experimentally-driven physical layers

Situation #1: Fully simulated physical layers

Option (a): Simulate the input images and the labels from scratch



Situation #1: Fully simulated physical layers

Option (a): Simulate the input images and the labels from scratch

Full simulation of different samples and their labels

Physical Layers

$f[\,]$

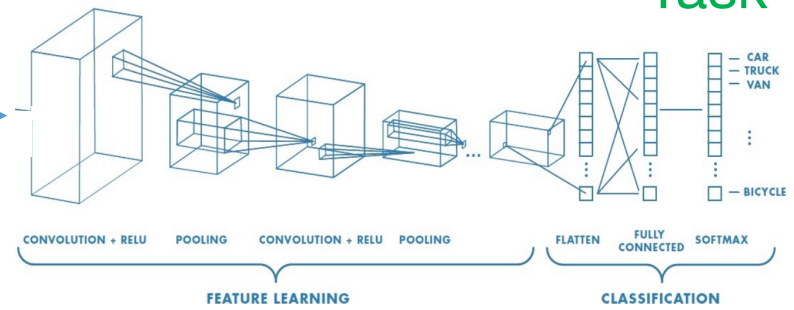
$I_s = f[I_0]$

Digitized

I_s

Digital Layers

Task



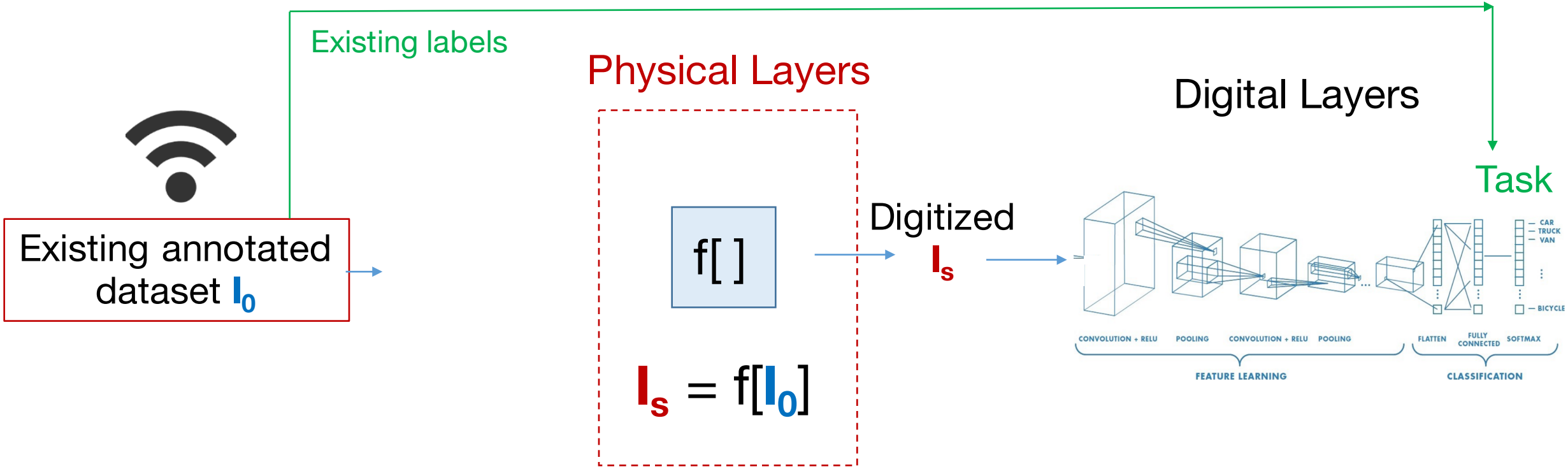
Simulated labels

Examples:

- [Ultrasound scatterers, segmentation boundary]
- [Simulated cell body types, location]
- [CT phantom, 3D mesh surfaces]

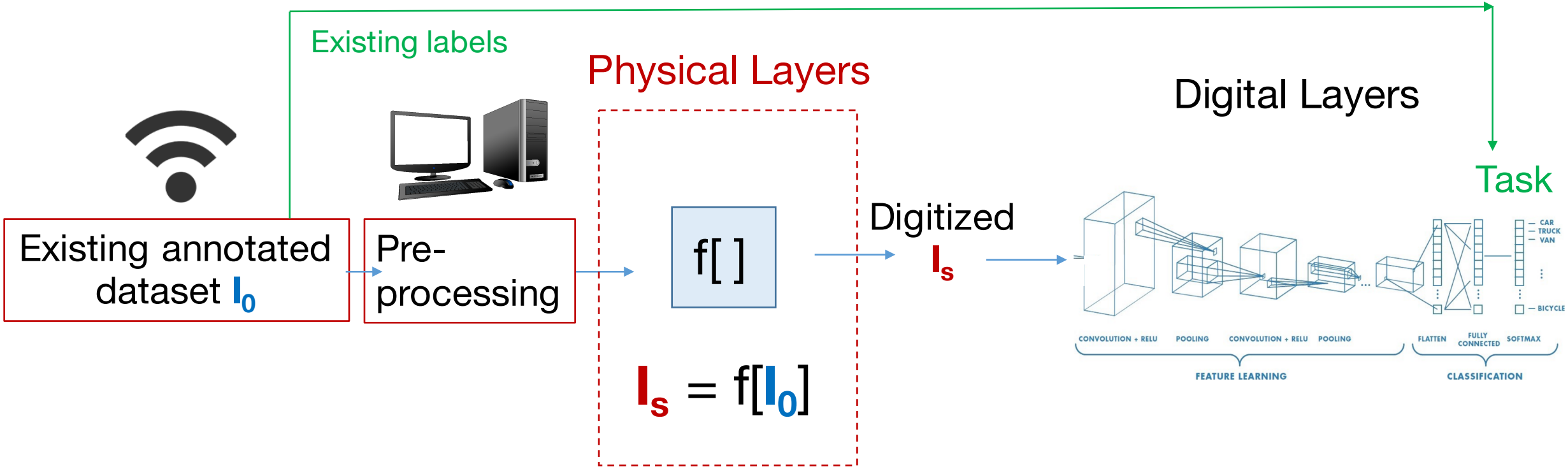
Situation #1: Fully simulated physical layers

Option (b): Augment an existing dataset that you download



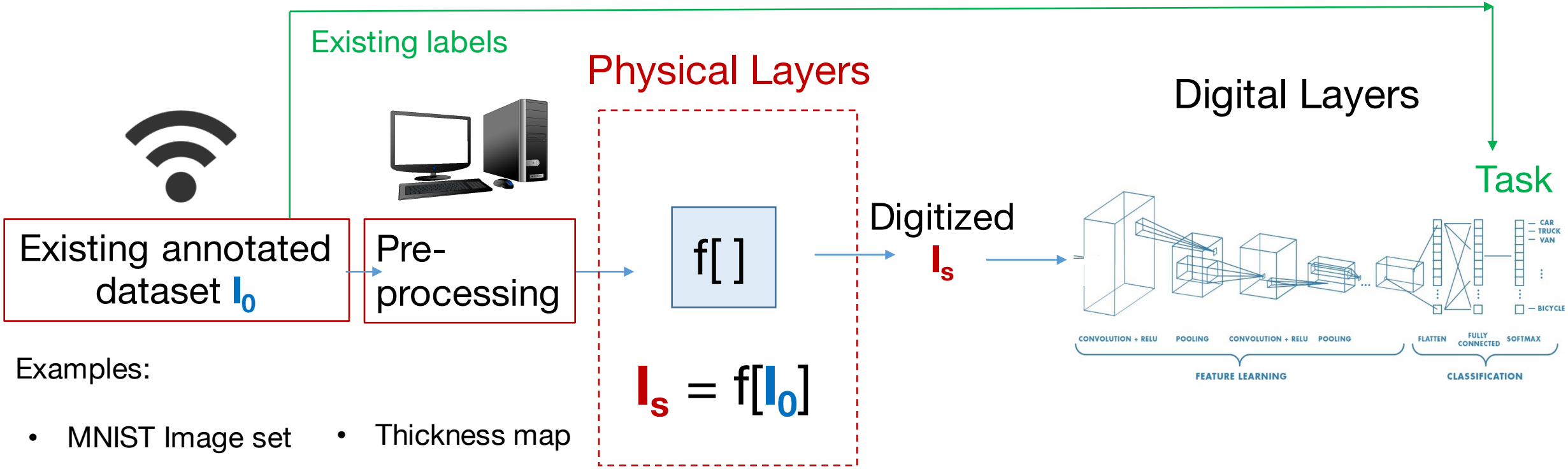
Situation #1: Fully simulated physical layers

Option (b): Augment an existing dataset that you download



Situation #1: Fully simulated physical layers

Option (b): Augment an existing dataset that you download



Examples:

- MNIST Image set
- Segmented cells from Celltracker
- Segmented CT dataset from lab
- Thickness map
- Multispectral image stack
- Stitch together in a 3D composite

Situation #1: Fully simulated physical layers

Option (a) or Option (b): Choice on where and how to simulate/pre-process

Simulation and/or pre-processing

Python/Matlab/other

Big dataset

ML Optimization

TensorFlow

TensorFlow → TensorFlow

Situation #1: Fully simulated physical layers

Option (a) or Option (b): Choice on where and how to simulate/pre-process

Simulation and/or pre-processing

Python/Matlab/other

Big dataset

ML Optimization

TensorFlow

Pros: Utilize old code, easier to archive, troubleshoot

Cons: Large datasets are slow to load, hard to fit in GPU memory, code in 2 places

---

TensorFlow → TensorFlow

Pros: batch processing, all in one place, easily incorporate additional physical layers

Cons: Harder to bug-check one part or the another if closely integrated

Aside on simulated data: Combining forward and inverse solvers

Forward problem: Start with the causes (objects in the real world) and compute the results (captured data)

Inverse problem: Start with the results (captured data) and infer about the causes (objects in the real world)

Aside on simulated data: Combining forward and inverse solvers

Forward problem: Start with the causes (objects in the real world) and compute the results (captured data)

(Typically easy)

Inverse problem: Start with the results (captured data) and infer about the causes (objects in the real world)

(Typically hard)

Aside on simulated data: Combining forward and inverse solvers

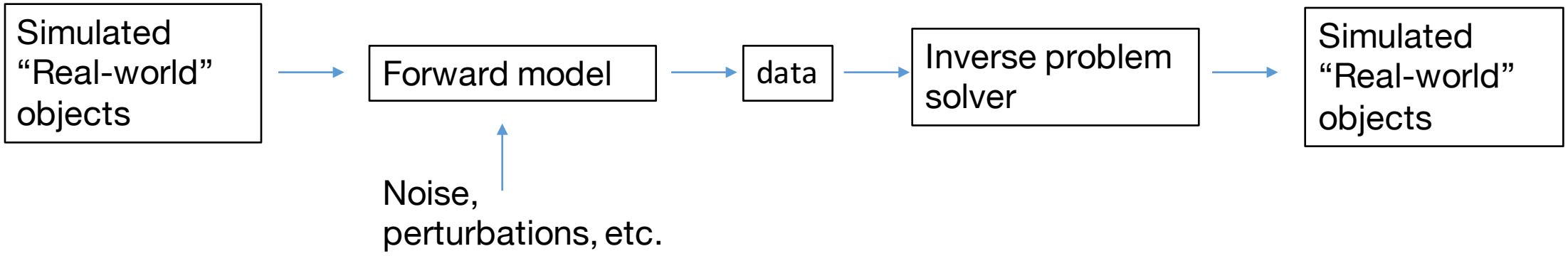
Forward problem: Start with the causes (objects in the real world) and compute the results (captured data)

(Typically easy)

Inverse problem: Start with the results (captured data) and infer about the causes (objects in the real world)

(Typically hard)

What I did in grad school to get ready for an experiment:



Aside on simulated data: Combining forward and inverse solvers

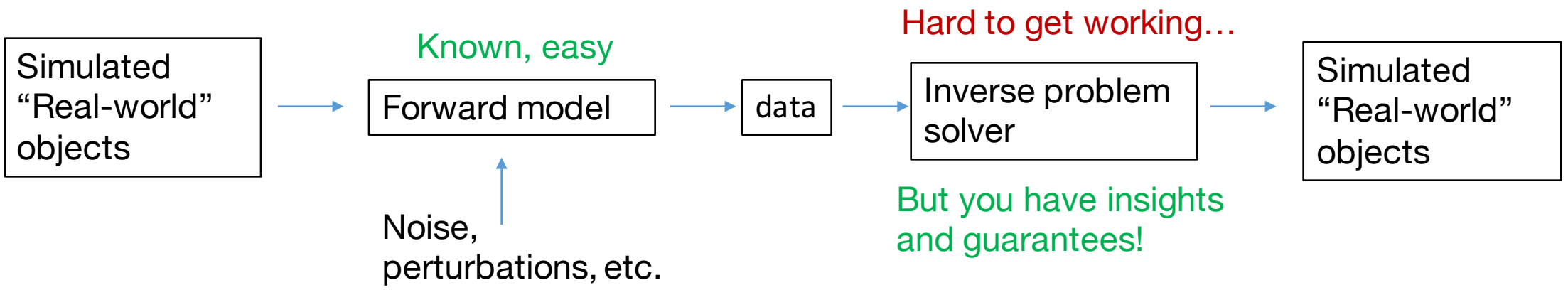
Forward problem: Start with the causes (objects in the real world) and compute the results (captured data)

(Typically easy)

Inverse problem: Start with the results (captured data) and infer about the causes (objects in the real world)

(Typically hard)

What I did in grad school to get ready for an experiment:



Aside on simulated data: Combining forward and inverse solvers

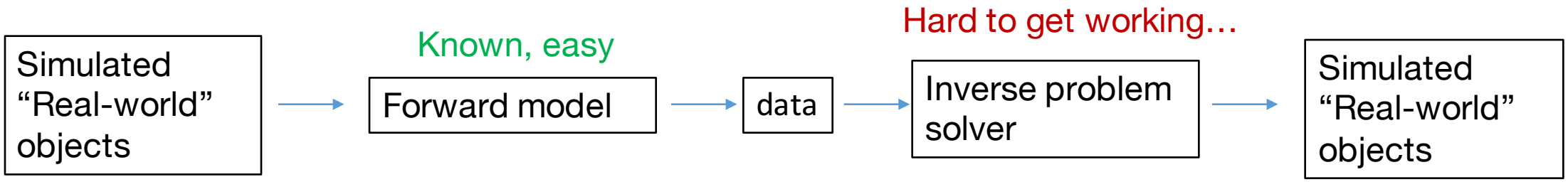
Forward problem: Start with the causes (objects in the real world) and compute the results (captured data)

(Typically easy)

Inverse problem: Start with the results (captured data) and infer about the causes (objects in the real world)

(Typically hard)

What I did in grad school to get ready for an experiment:



Classic examples: Inverse Radon Transform, US image reconstruction, image deblurring/denoising, diffraction tomography, phase retrieval, super-resolution (structured illumination, STORM/PALM), etc.

Aside on simulated data: Combining forward and inverse solvers

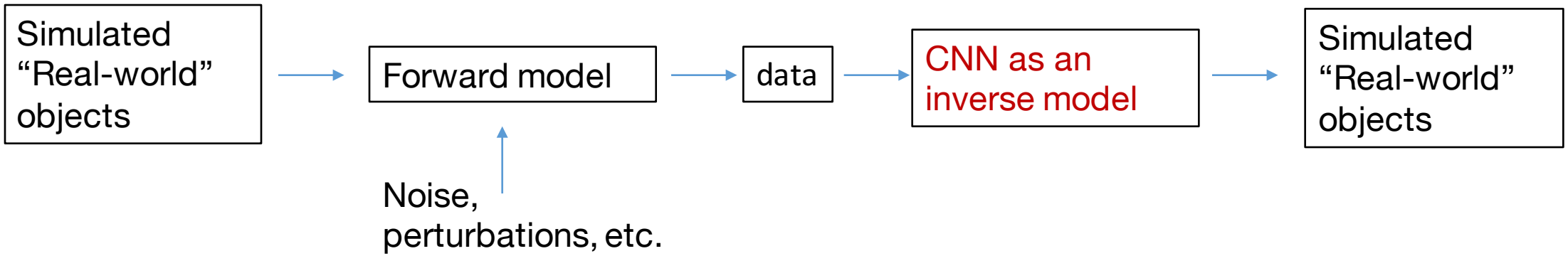
Forward problem: Start with the causes (objects in the real world) and compute the results (captured data)

(Typically easy)

Inverse problem: Start with the results (captured data) and infer about the causes (objects in the real world)

(Typically hard)

What you can do now with CNN's:



Aside on simulated data: Combining forward and inverse solvers

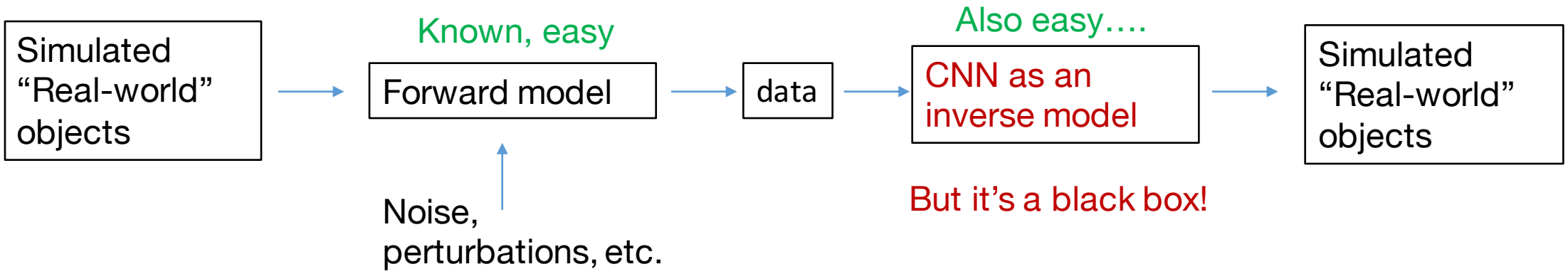
Forward problem: Start with the causes (objects in the real world) and compute the results (captured data)

(Typically easy)

Inverse problem: Start with the results (captured data) and infer about the causes (objects in the real world)

(Typically hard)

What you can do now with CNN's:



Situation #2: Fully simulated physical layers

Experimental measurements

"Expert" annotation

Generated Labels

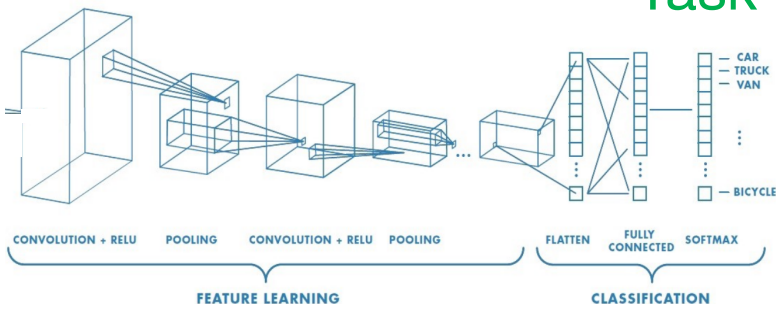
Physical Layers

f[ ]

$I_s = f[I_0]$

Digitized I_s

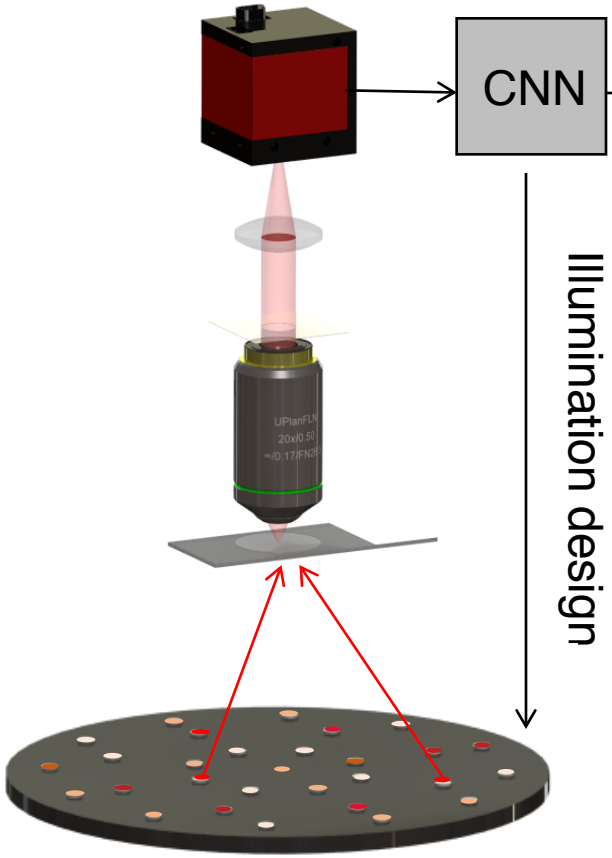
Digital Layers

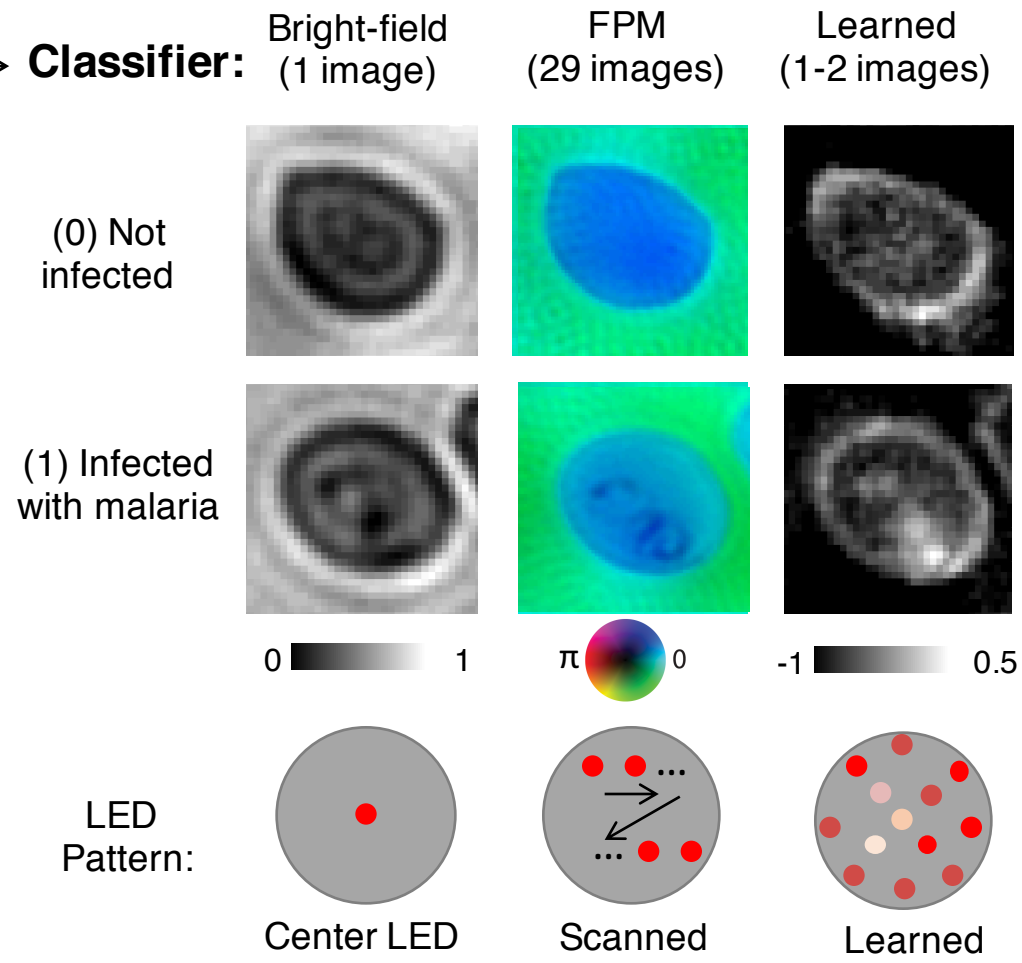


Task

Situation #2: Fully simulated physical layers

Example: CNN-Optimized illumination for classification of malaria:



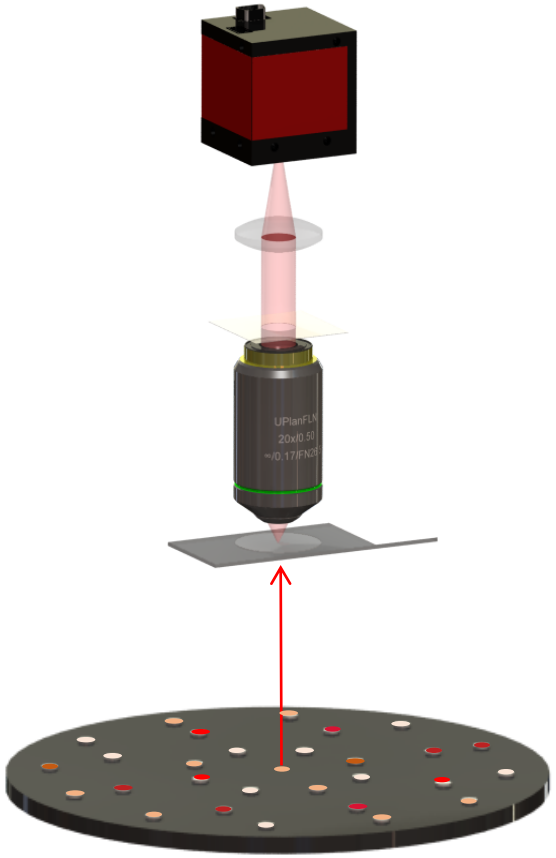


Classification Accuracy: **75%** **98.5%** **97%**

R. Horstmeyer et al., "Convolutional neural networks that teach microscopes how to image," arXiv (2017)

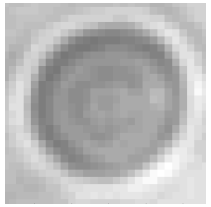
Situation #2: Fully simulated physical layers

Example: CNN-Optimized illumination for classification of malaria:



Data set for physical layer optimization:

Turn on LED 1, capture image 1:



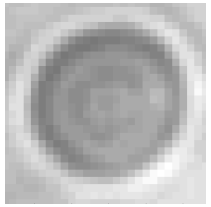
R. Horstmeyer et al., "Convolutional neural networks that teach microscopes how to image," arXiv (2017)

Situation #2: Fully simulated physical layers

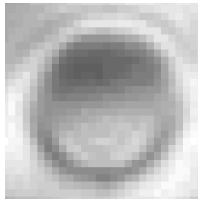
Example: CNN-Optimized illumination for classification of malaria:

Data set for physical layer optimization:

Turn on LED 1, capture image 1:



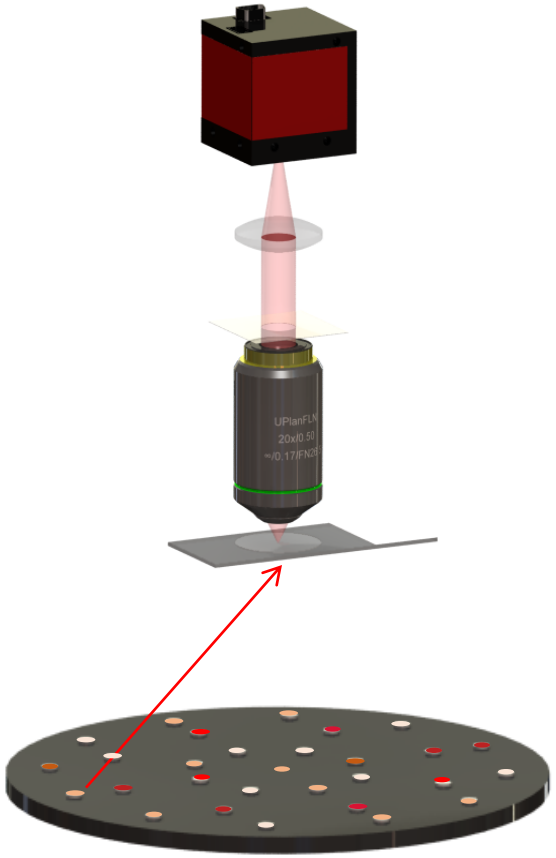
Turn on LED 1, capture image 2:



R. Horstmeyer et al., "Convolutional neural networks that teach microscopes how to image," arXiv (2017)

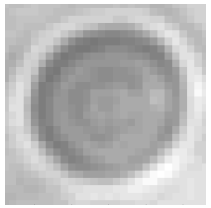
Situation #2: Fully simulated physical layers

Example: CNN-Optimized illumination for classification of malaria:

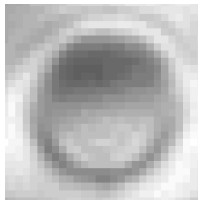


Data set for physical layer optimization:

Turn on LED 1, capture image 1:

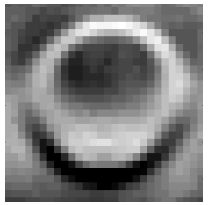


Turn on LED 1, capture image 2:



⋮

Turn on LED 32, capture image 32:

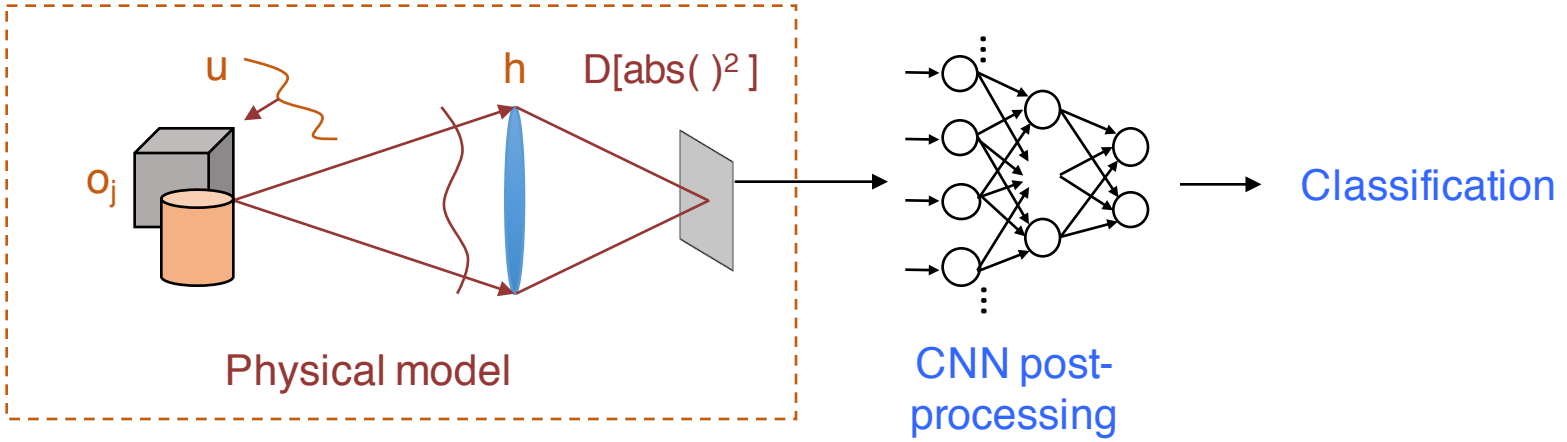


Save all 32 images (96 with 3 colors)

R. Horstmeyer et al., "Convolutional neural networks that teach microscopes how to image," arXiv (2017)

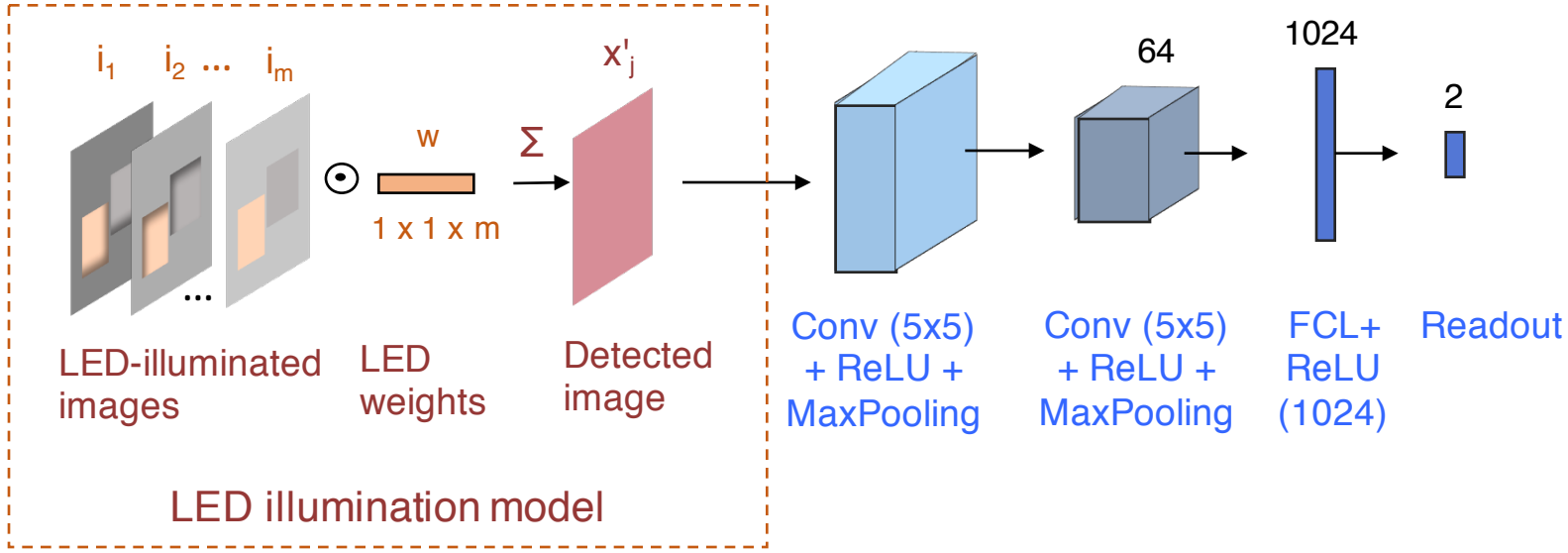
Situation #2: Fully simulated physical layers

Example: CNN-Optimized illumination for classification of malaria:



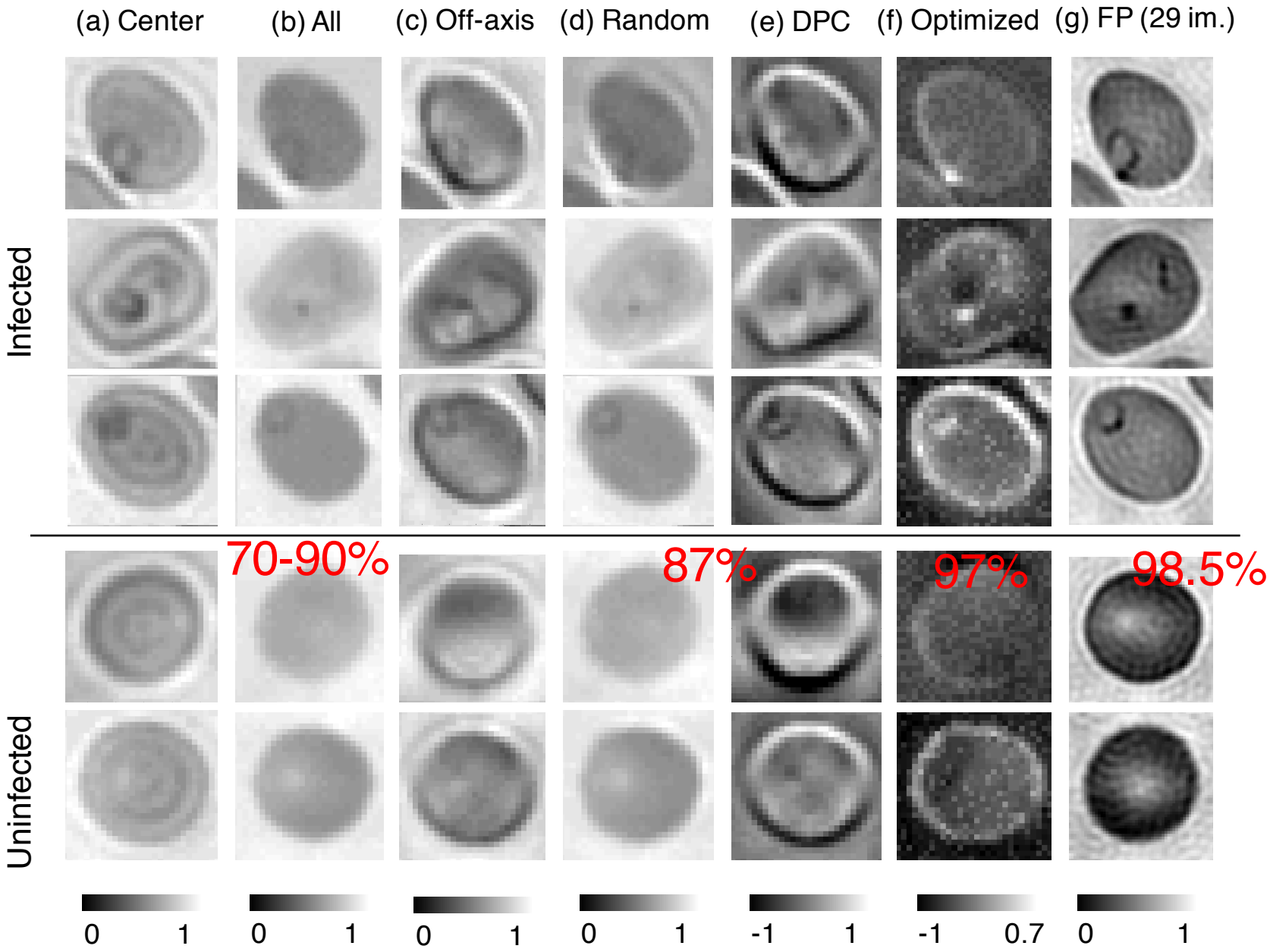
Physical layer:

$$I_s = \Sigma\, w_j I_j$$

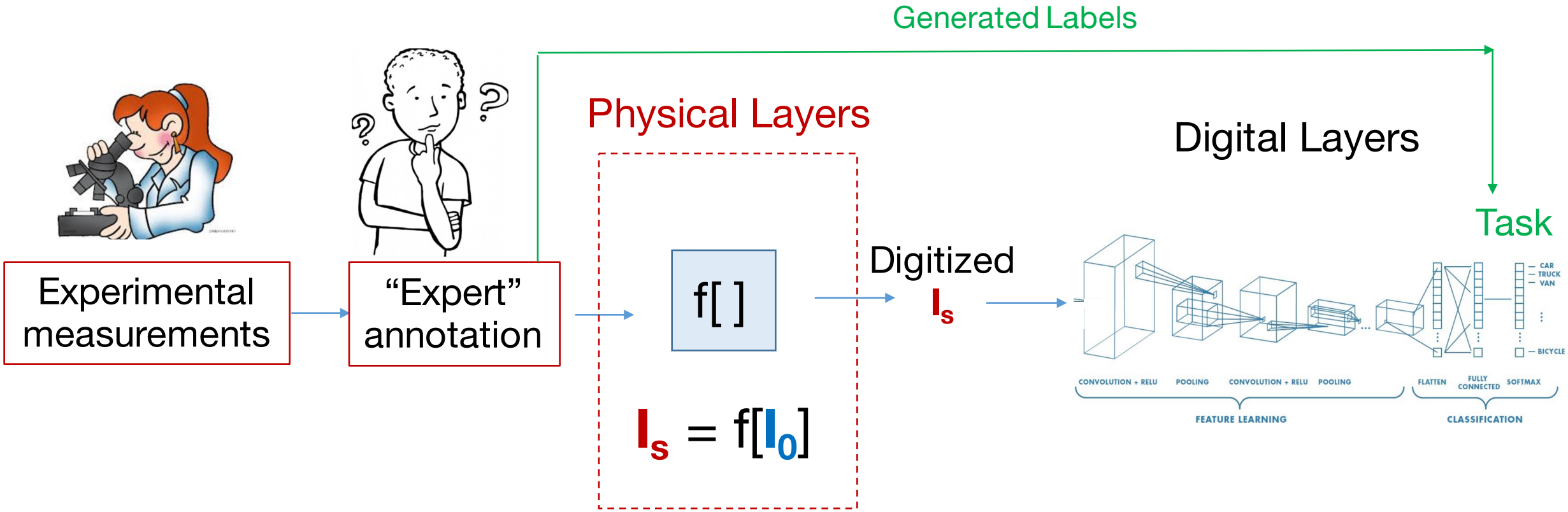


Situation #2: Fully simulated physical layers

Example: CNN-Optimized illumination for classification of malaria:



Situation #2: Fully simulated physical layers



Pro's of experimental measurements: Don't need to worry about making your simulations match the setup! (HUGE)

Con's of experimental measurements: You'll need to label them, limited access to desired sample information, often need to exploit some fundamental physical property

How can I add some constraints to my physical weights?

Without any constraints, weights can be any real (or complex) number What if you physically can't realize any real or physical number?

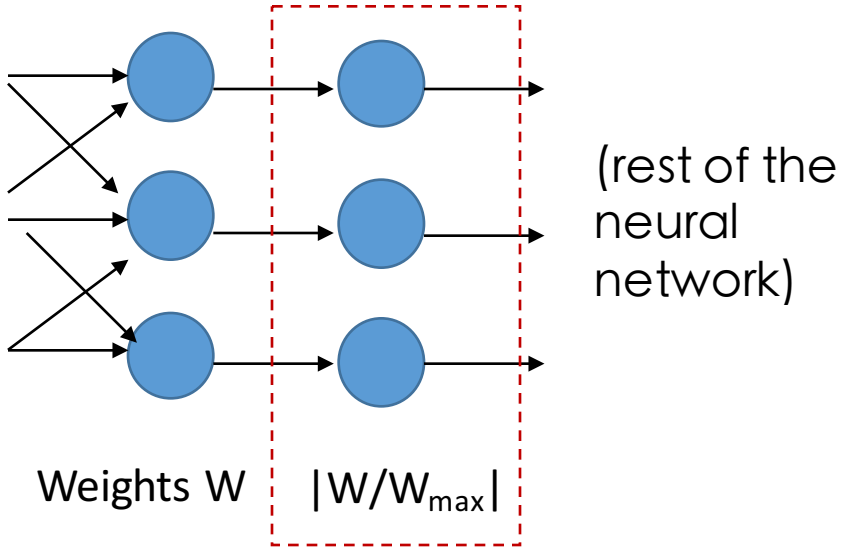
Example: Constrain weights to be non-negative values less than one

How can I add some constraints to my physical weights?

Without any constraints, weights can be any real (or complex) number What if you physically can't realize any real or physical number?

Example: Constrain weights to be non-negative values less than one

Solution: add constraint as an extra "differentiable" layer (operation)

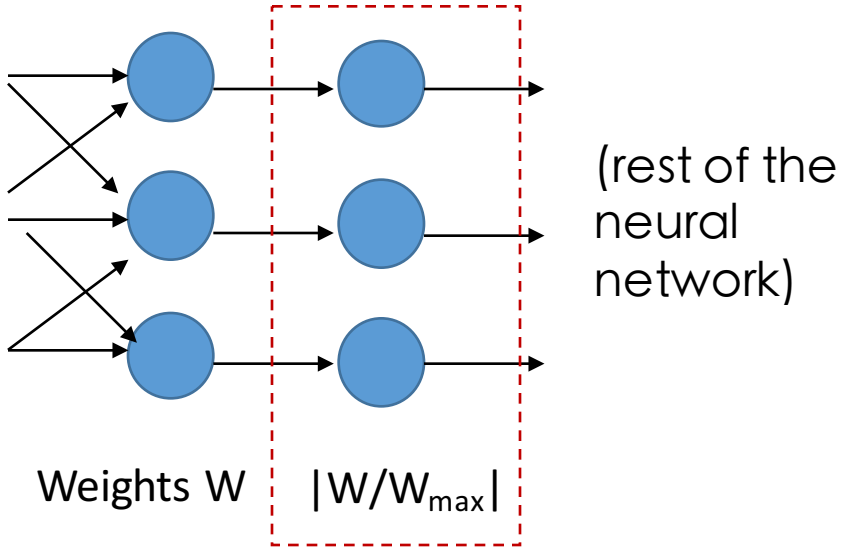


How can I add some constraints to my physical weights?

Without any constraints, weights can be any real (or complex) number What if you physically can't realize any real or physical number?

Example: Constrain weights to be non-negative values less than one

Solution: add constraint as an extra "differentiable" layer (operation)



Pros:

- Easy to implement
- Constraints are obvious

Cons:

- Not always a well-behaved derivative

How can I add some constraints to my physical weights?

Without any constraints, weights can be any real (or complex) number What if you physically can't realize any real or physical number?

Example: Constrain weights to be either 0 or 1

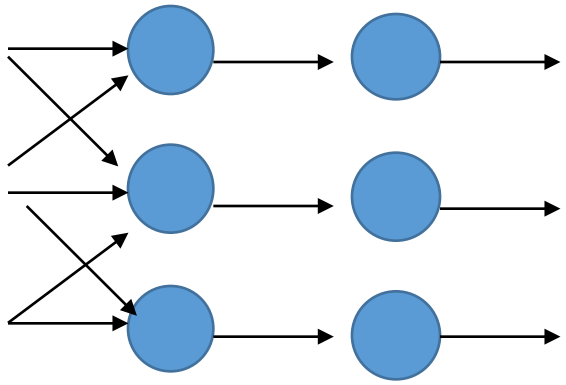
Solution: Perform annealing with a temperature parameter

How can I add some constraints to my physical weights?

Without any constraints, weights can be any real (or complex) number What if you physically can't realize any real or physical number?

Example: Constrain weights to be either 0 or 1

Solution: Perform annealing with a temperature parameter



(rest of the neural network)

Weights W $I(n) = \text{Soft-max}\left[\alpha_t w(n)\right]$

Increase α with iteration number

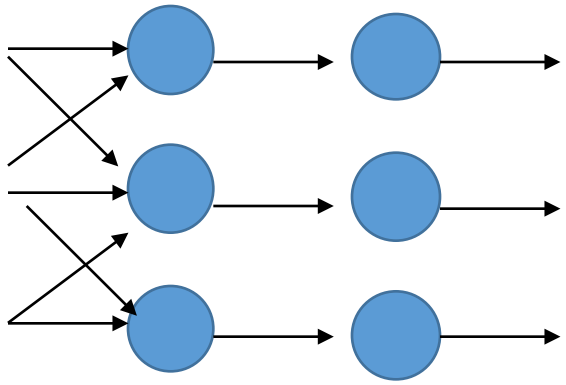
Soft-max(x) = exp(-x)/ Σ exp(-x)

How can I add some constraints to my physical weights?

Without any constraints, weights can be any real (or complex) number What if you physically can't realize any real or physical number?

Example: Constrain weights to be either 0 or 1

Solution: Perform annealing with a temperature parameter



(rest of the neural network)

Weights W

$$I(n) = \text{Soft-max}\left[\alpha_t w(n)\right]$$

Increase α with iteration number

Soft-max(x) = exp(-x)/ Σ exp(-x)

Pros:

- Works pretty well
- Flexibly address convergence issues

Cons:

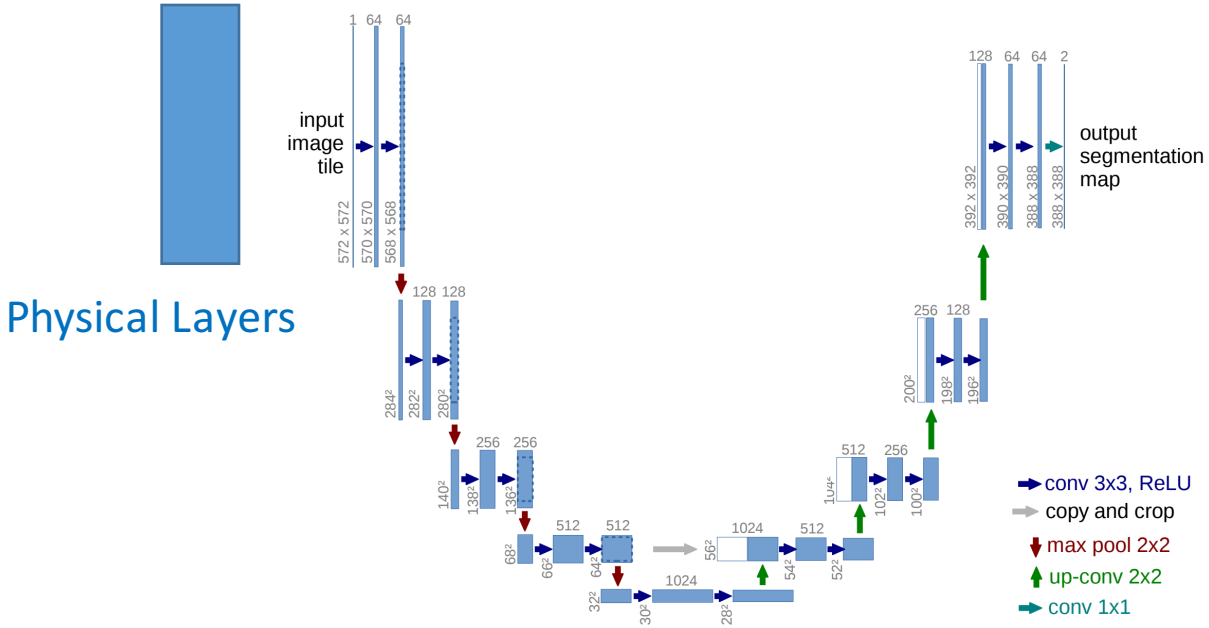
- A bit sensitive

What are some common issues and pitfalls with physical layers?

- Most common issue – you have a bug in your CNN!
    - Solution: “Disable “ physical layer (set to constant), and get network to work!
    - Good practice: always compare performance with and without physical layer
- Another common challenge - vanishing gradients

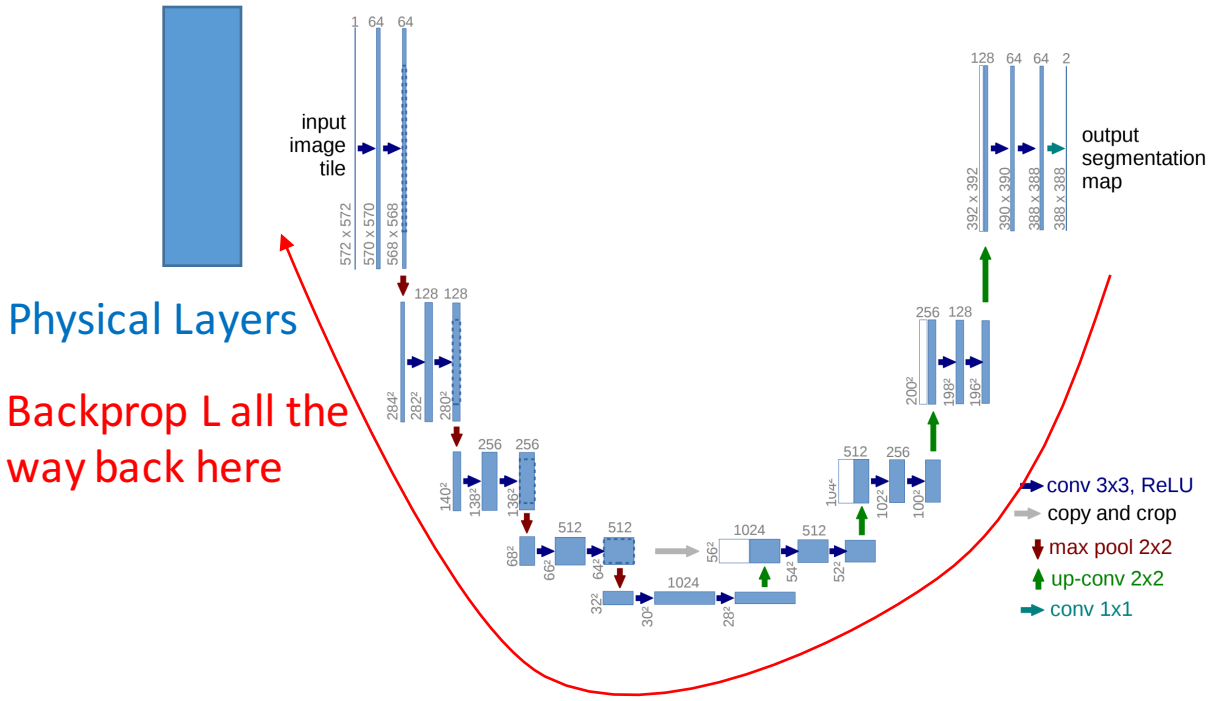
What are some common issues and pitfalls with physical layers?

- Most common issue – you have a bug in your CNN!
  - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients



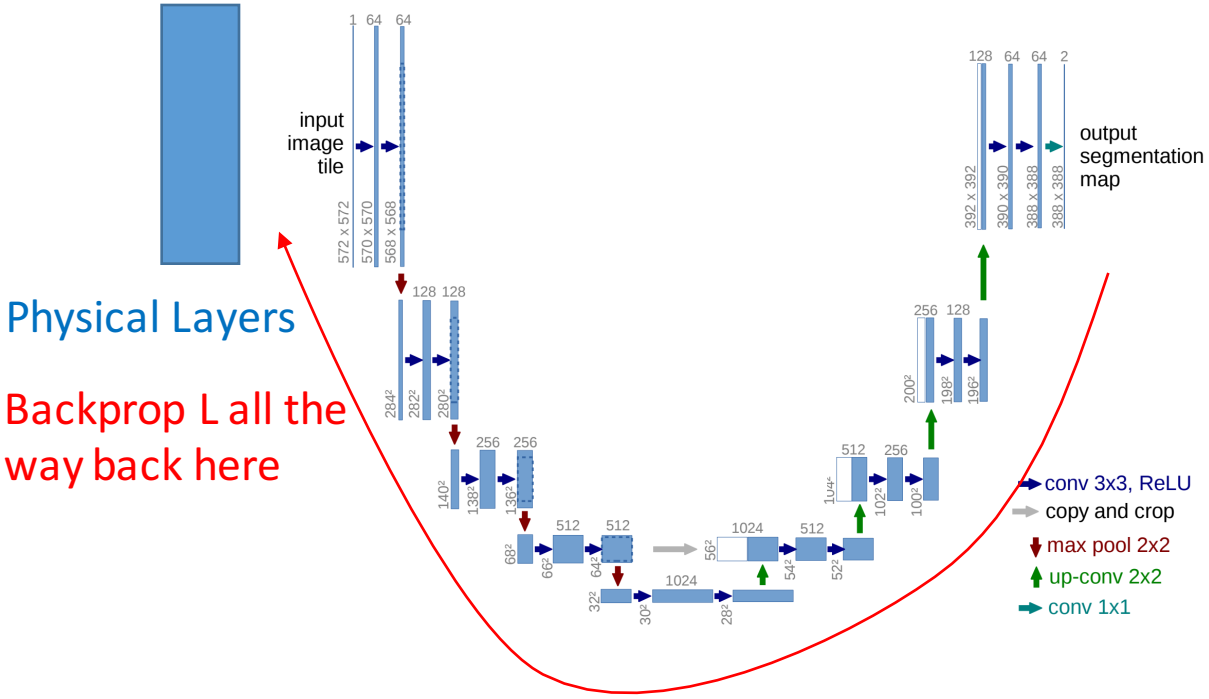
What are some common issues and pitfalls with physical layers?

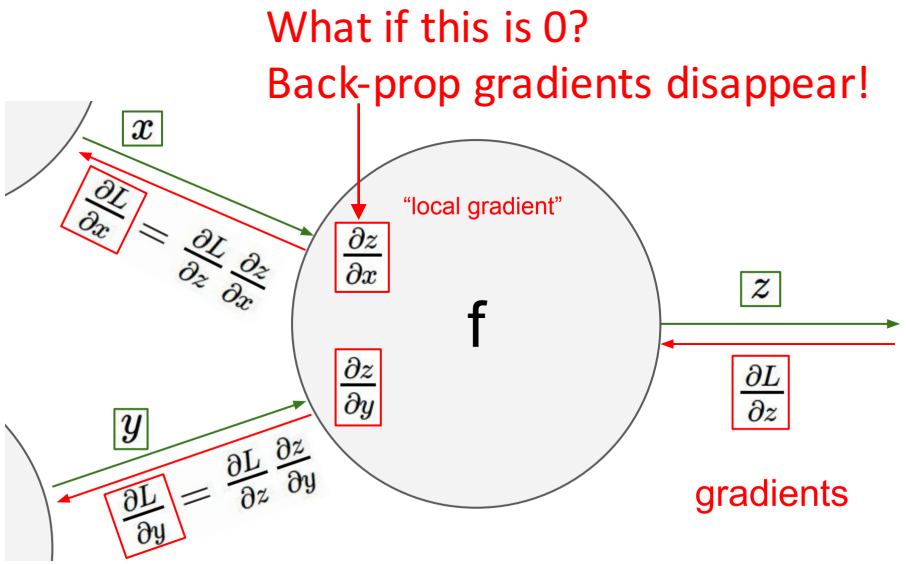
- Most common issue – you have a bug in your CNN!
  - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients



What are some common issues and pitfalls with physical layers?

- Most common issue – you have a bug in your CNN!
  - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients

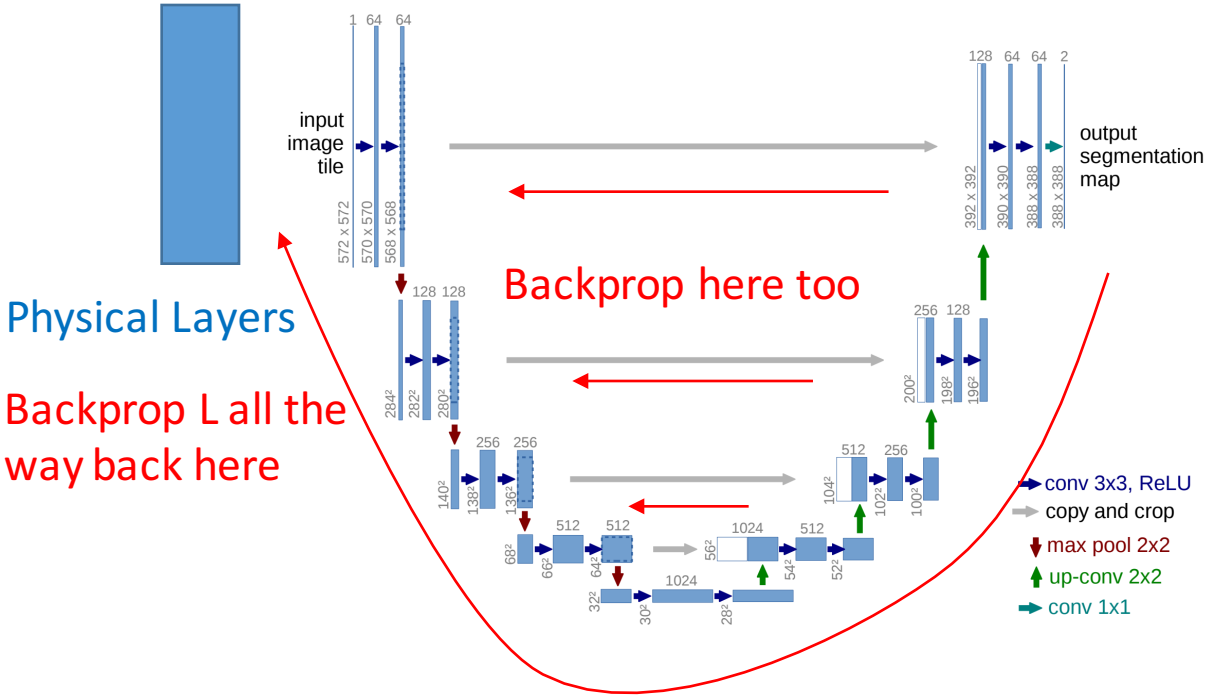




From Stanford CS231n

What are some common issues and pitfalls with physical layers?

- Most common issue – you have a bug in your CNN!
  - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients



Solution: Introduce skipped connections

Discussion group time!

1. Introduce your project (3 min)
2. Describe one (or more) potential challenges that you anticipate (2 min)
3. Discuss possible solutions to this challenge (3-5 min)
4. Switch!