Lecture 18: Coherent physical layers and general guidelines

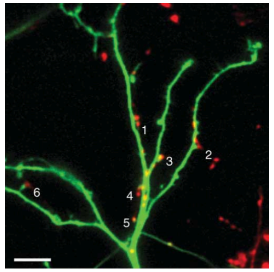
Machine Learning and Imaging

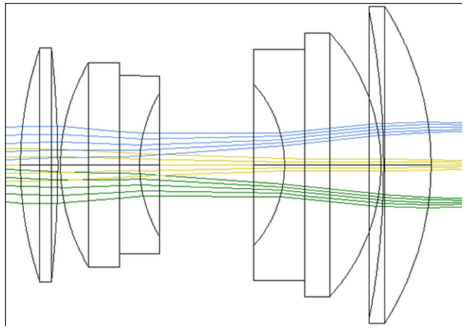
BME 548L

Roarke Horstmeyer

Summary of two models for image formation

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



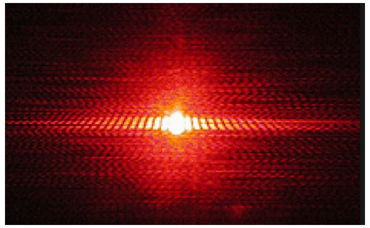


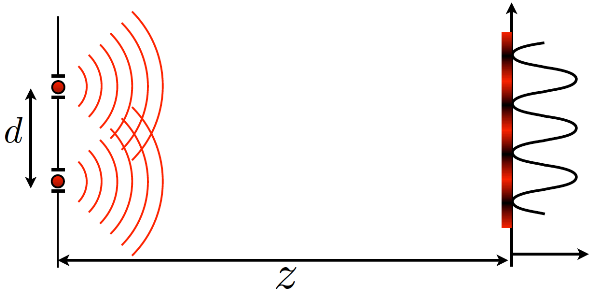
- Real, non-negative
- Models absorption and brightness

$$I_{tot} = I_1 + I_2$$

$$\mathbf{I_s = H\ B\ S_0}$$

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



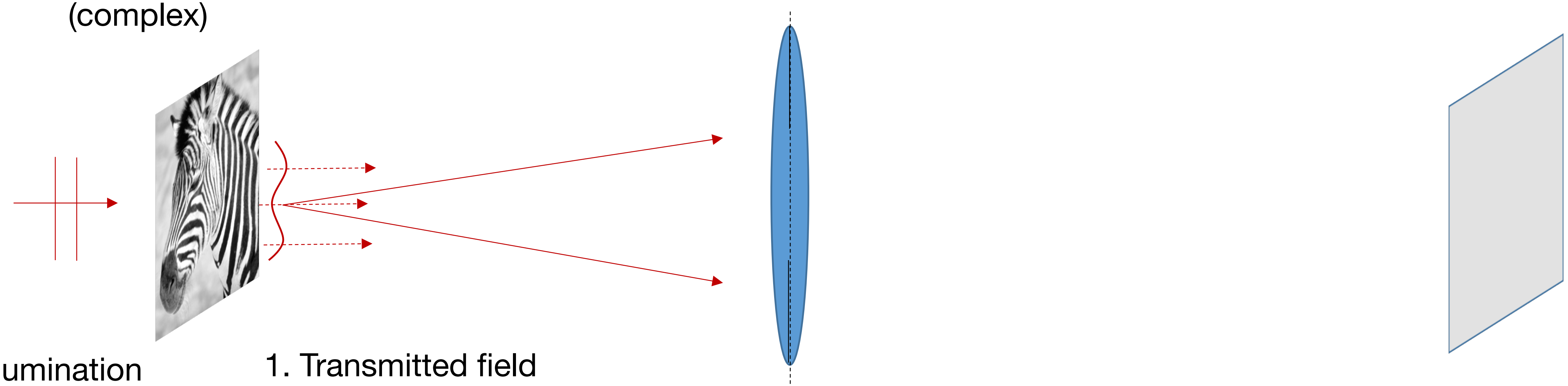


- Complex field
- Models Interference

$$E_{tot} = E_1 + E_2$$

Model of image formation for wave optics (coherent light):

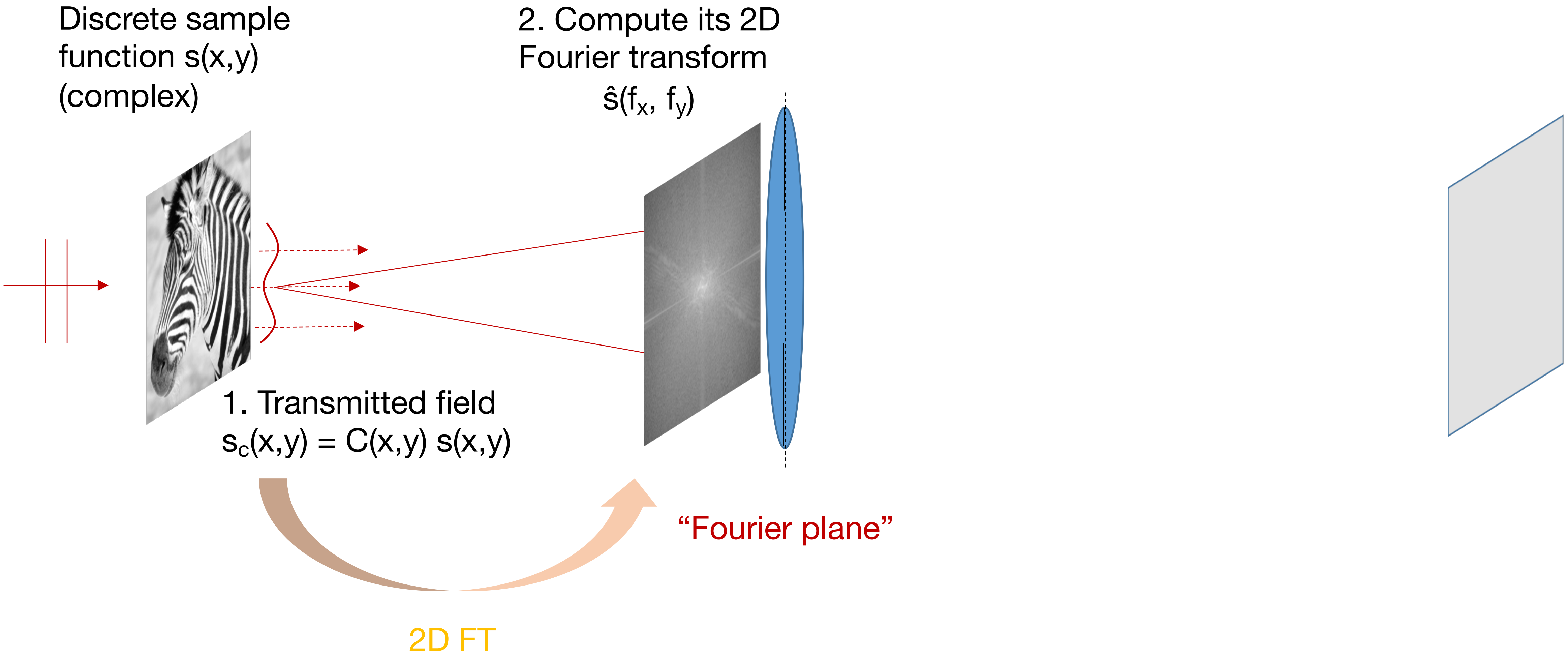
Discrete sample function s(x,y) (complex)



Illumination field C(x,y)

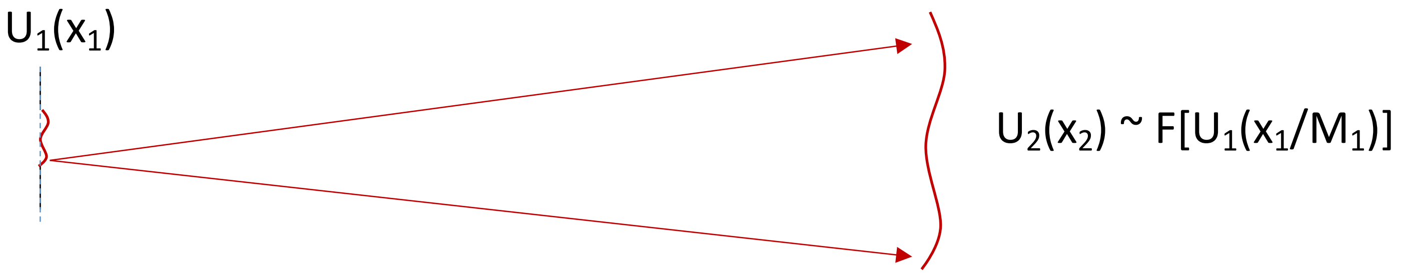
1. Transmitted field
$s_c(x,y) = C(x,y)\ s(x,y)$

Model of image formation for wave optics (coherent light):

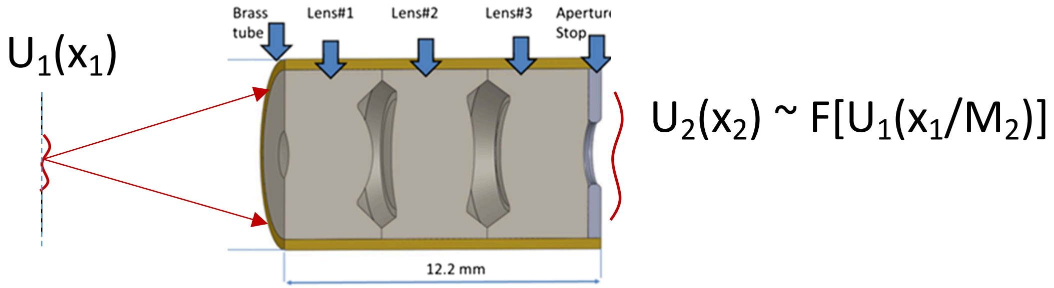


General rules for applying the Fourier transform in optics

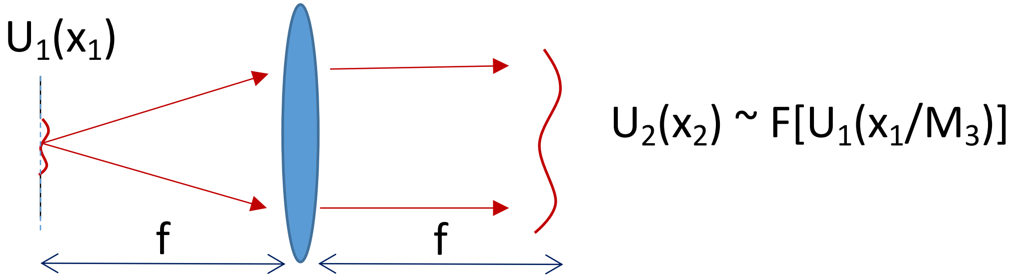
Situation 1: From an object to a plane "really far away"



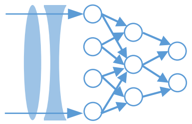
Situation 2: From an object to the back focal plane of the microscope objective lens

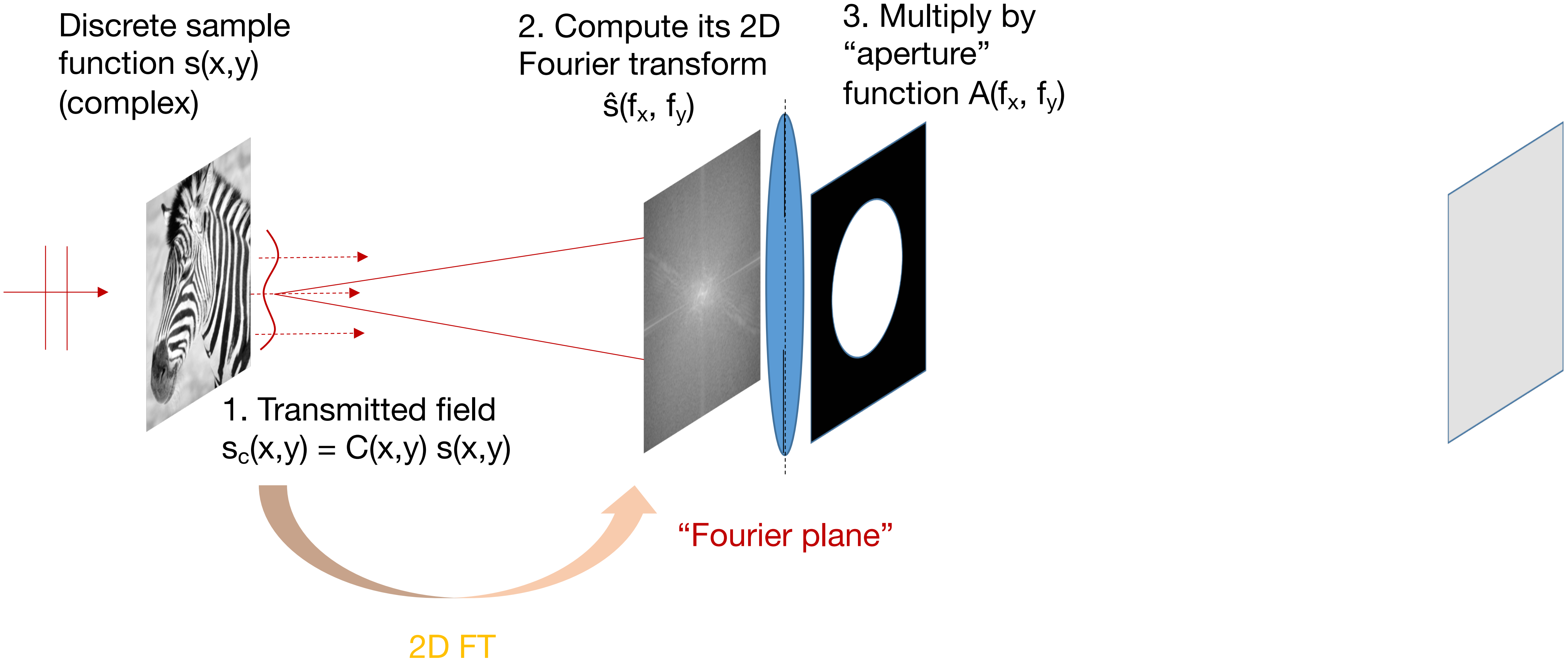


Situation 3: From an object to a plane 1 focal length away from a lens (1f-1f system)



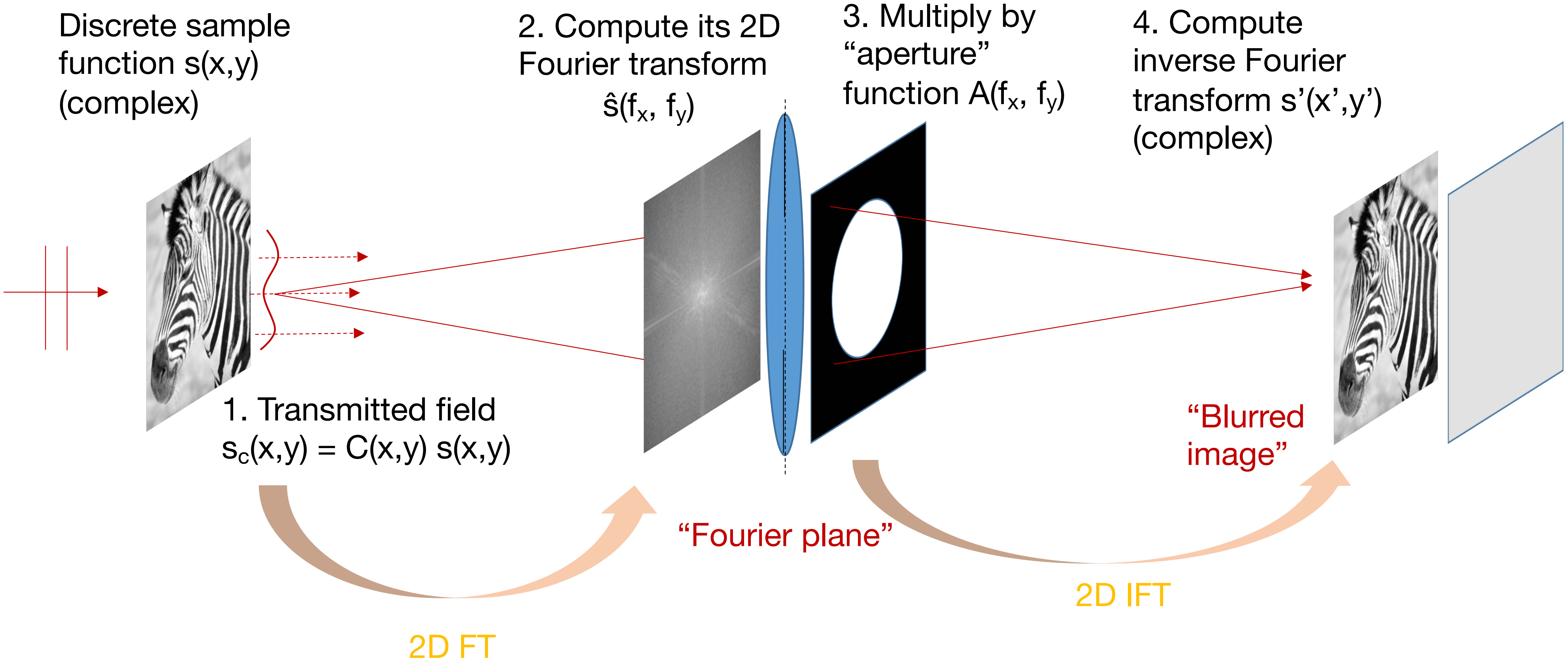
Model of image formation for wave optics (coherent light):



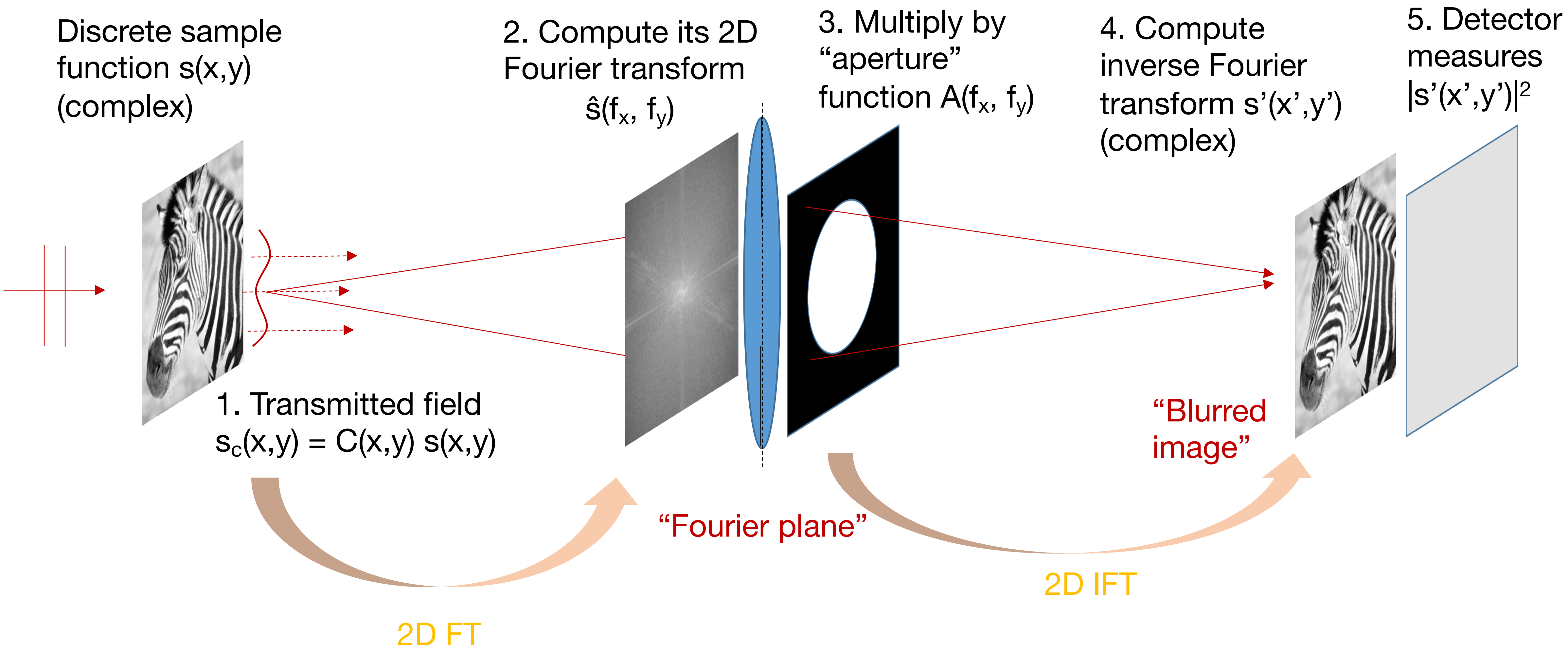


deep imaging

Model of image formation for wave optics (coherent light):

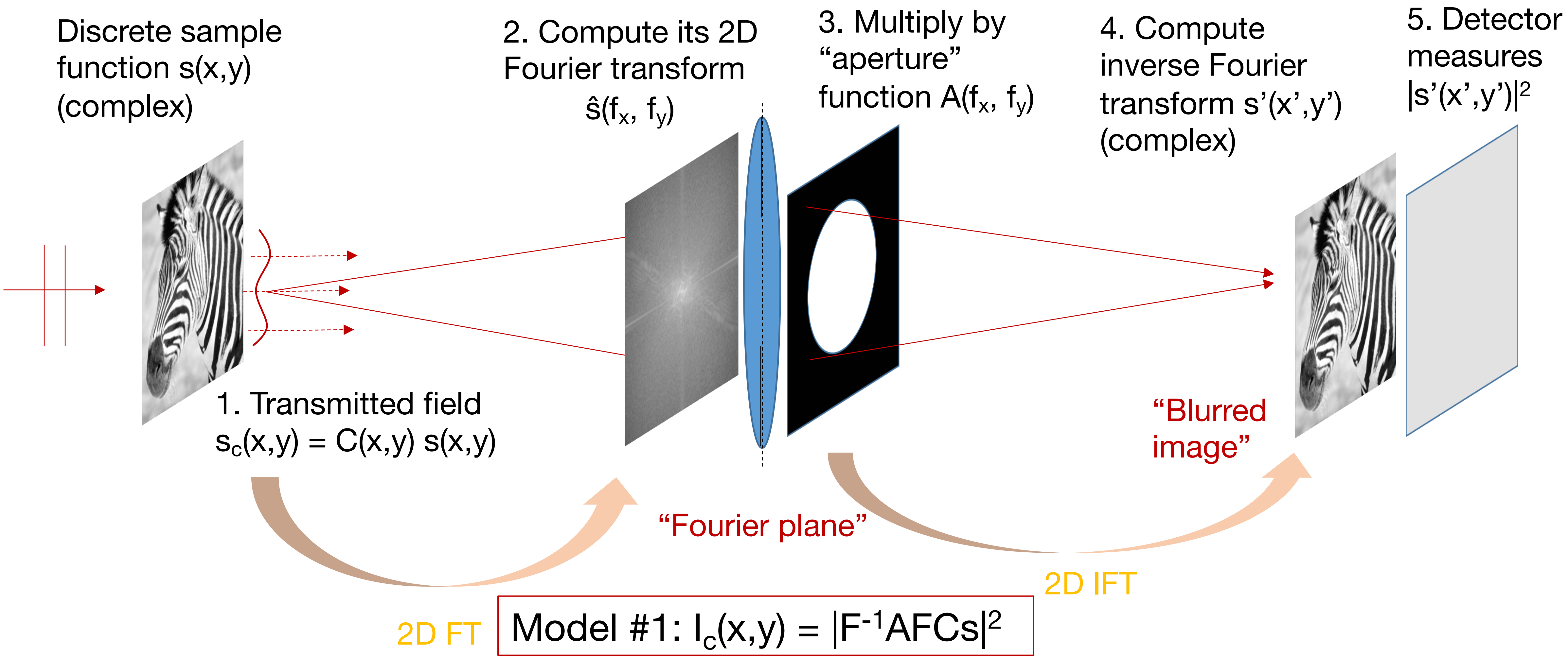


Model of image formation for wave optics (coherent light):



deep imaging

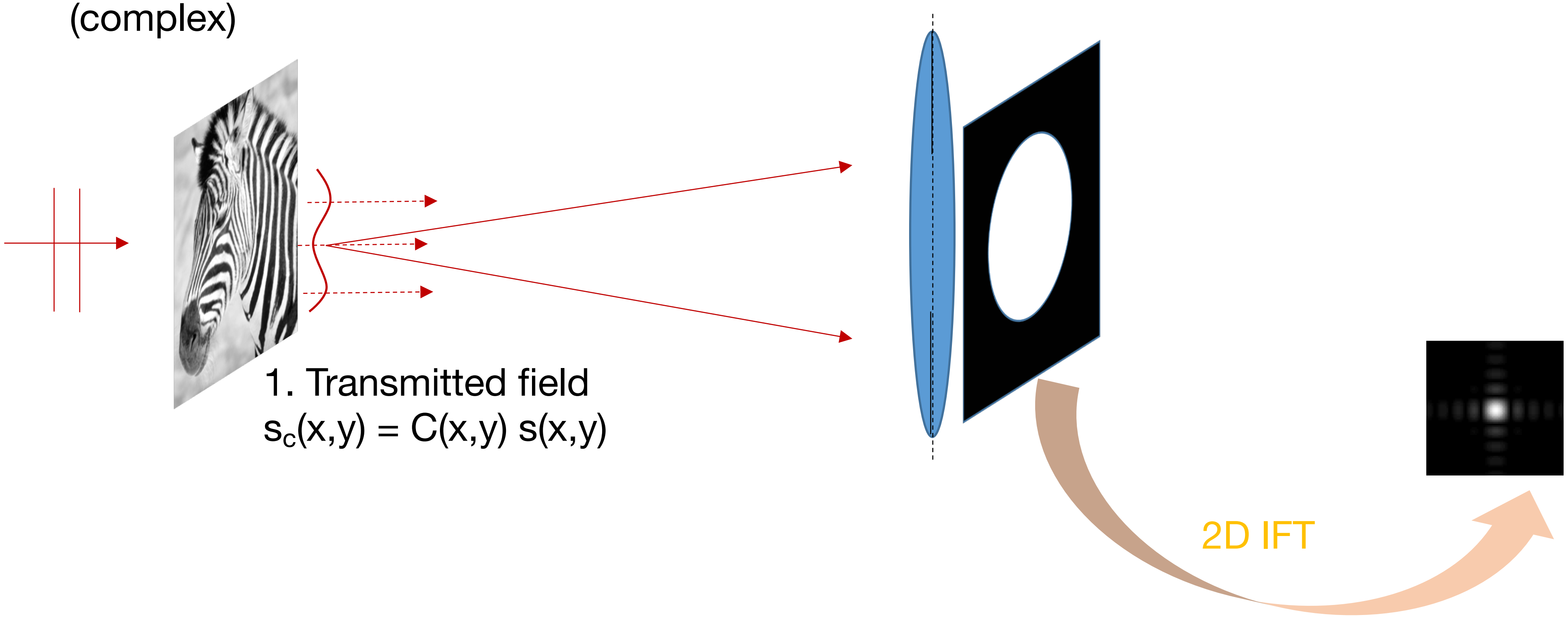
Model of image formation for wave optics (coherent light):



Model #1: $I_c(x,y) = |F^{-1}AFCs|^2$

deep imaging

Model of image formation for wave optics (coherent light):

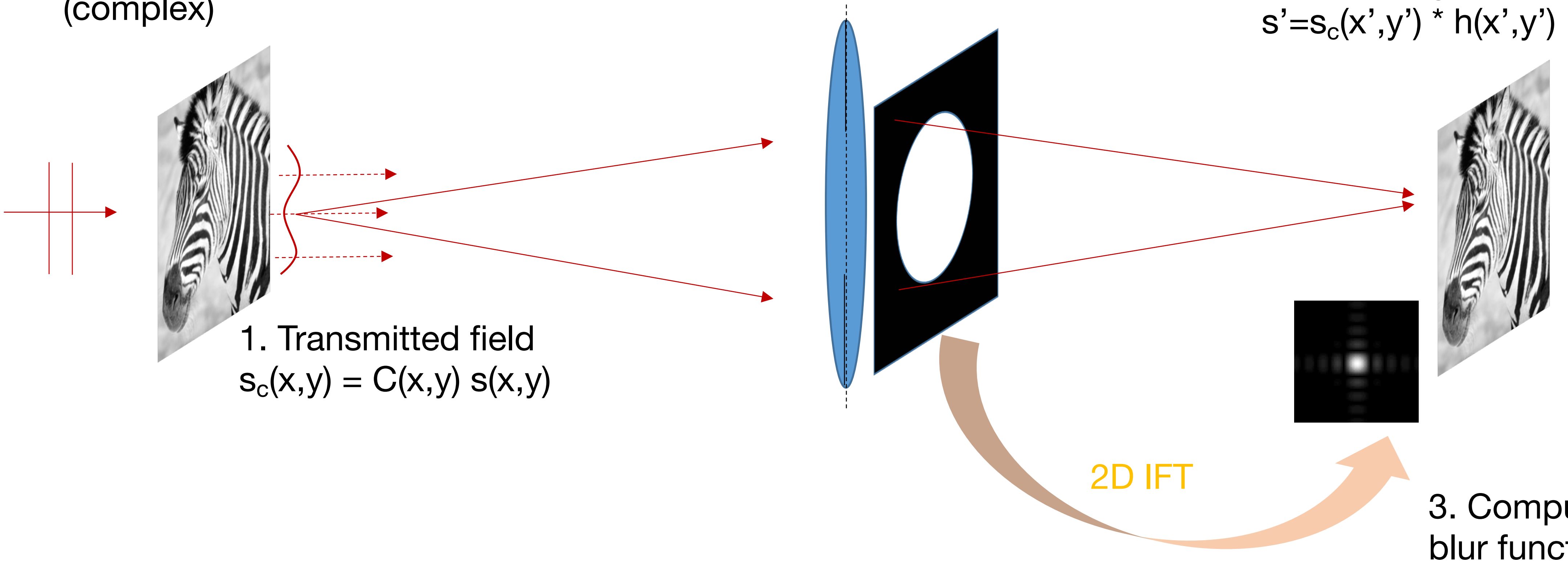


Model of image formation for wave optics (coherent light):

Discrete sample function $s(x,y)$ (complex)

2. "aperture" function $A(f_x, f_y)$

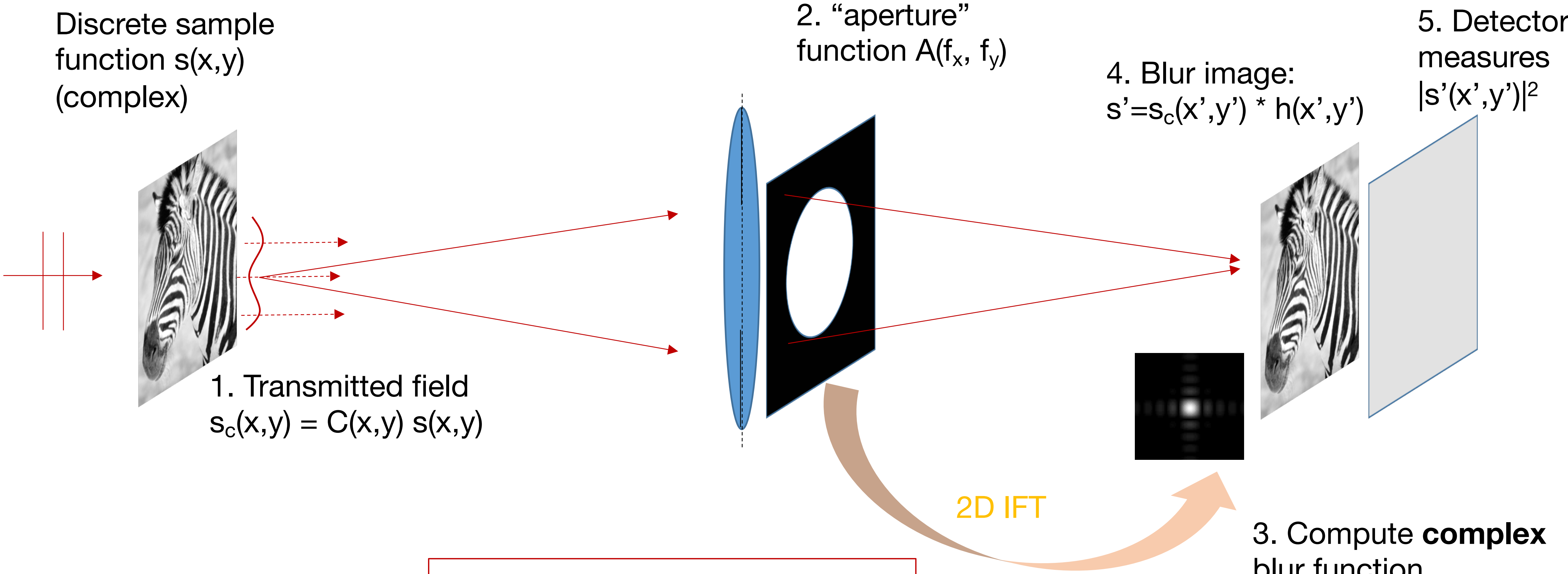
4. Blur image: $s' = s_c(x',y') * h(x',y')$



1. Transmitted field $s_c(x,y) = C(x,y)\, s(x,y)$

3. Compute **complex** blur function $h(x,y) = F[A(f_x, f_y)]$

Model of image formation for wave optics (coherent light):



$$\boxed{\text{Model \#2: } I_c(x,y) = |h \ast Cs|^2}$$

deep imaging

You typically go between 4 functions to describe one imaging system:

<u>Coherent Light</u> | **<u>Incoherent Light</u>**

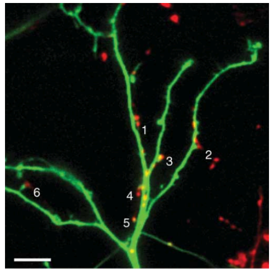
$$
\begin{tikzcd}
\text{Coherent point-spread function } h_c(x) \arrow[r, "|.|^2"] \arrow[d, "F[.]"] & \text{Incoherent point-spread function } h_i(x) \arrow[d, "F[.]"] \\
\text{Coherent transfer function } H_c(f_x) \arrow[r, "H*H"] & \text{Incoherent transfer function } H_i(f_x)
\end{tikzcd}
$$

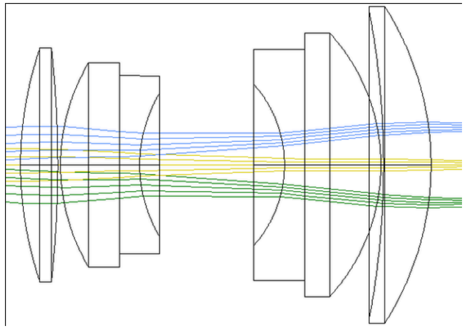
Incoherent PSF = Coherent PSF squared:

$h_i(x) = |h_c(x)|^2$

Summary of two models for image formation

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



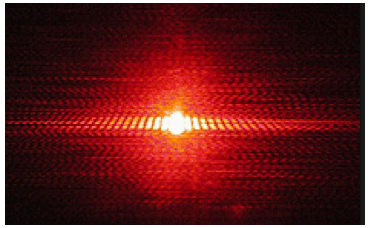


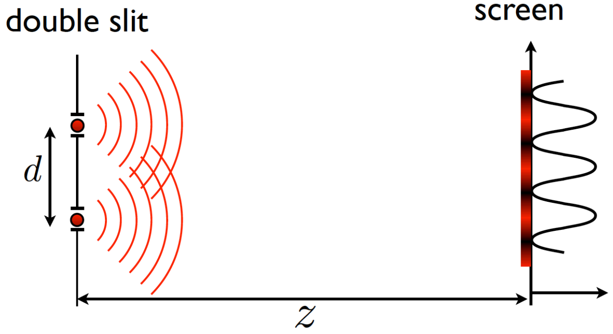
- Real, non-negative

$$\boxed{I_s = h_i * B\, S_0}$$

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

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





- Complex-valued

$$\boxed{I_C = |\, h_c * C\, S_C\,|^2}$$

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

Coherent image formation equation as CNN operations

deep imaging

$$\boxed{I_C = D\,|\,h_c * C\,S_C\,|^2}$$

CNN layer

Step 1: Multiply with weights	(Step 1: Normalization)
Step 2: Convolution	Step 2: Convolution
Step 3: Absolute value square (non-linearity)	Step 3: Non-linearity
Step 4: Down-sampling by detector	Step 4: Down-sampling by max pooling

Example #1: Optimizing coherent illumination pattern for improved classification

Example future situation: Hacking has brought online banking to a halt. We now rely on a special form of physical check that is made of visibly transparent plastic. To write the amount in, you press down with a pen-like instrument, and then the check is read out by shining a particular pattern of laser light onto it, and then imaging it with a lens.

Question: What type of illumination should you use to maximize the classification accuracy of the numbers on the check?

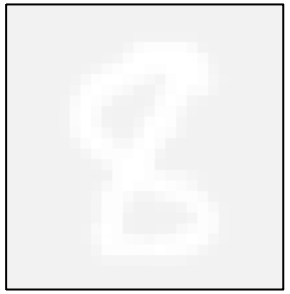
Step 1: Transform MNIST image data set into transparent plastic sheets with varying thickness

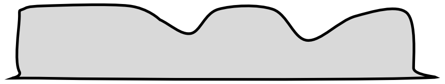
Values [0, 255]

I(x,y)

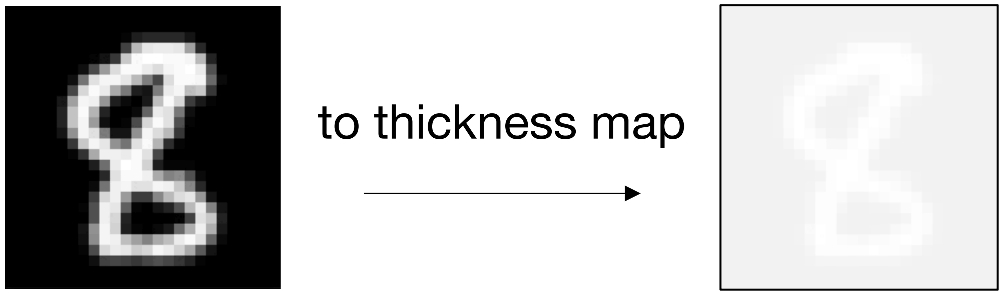
to thickness map

∂t(x,y)

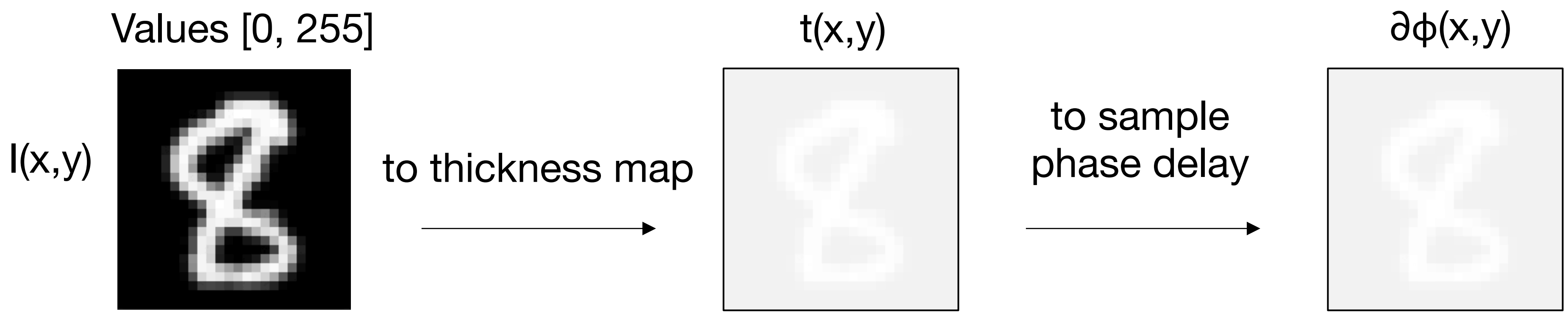




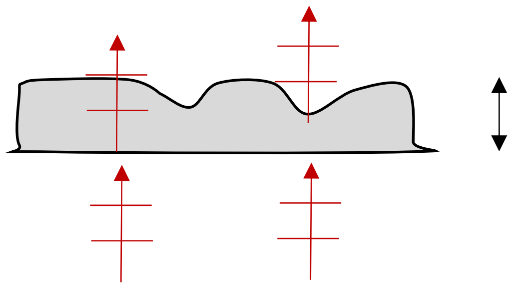
∂t(x,y)



1. Normalize intensity map to 1
2. Define thickness map at some reasonable amount (100 μm max change)

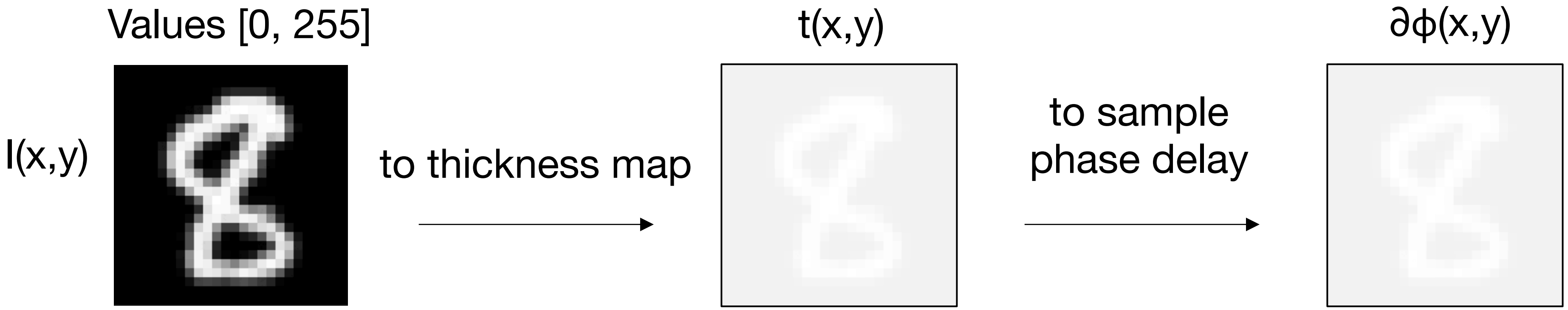


1. Normalize intensity map to 1
2. Define thickness map at some reasonable amount (100 μm max change)
3. Convert thickness map into optical phase delay:



$$\boxed{\partial\phi(x,y) = \exp[\, j\, n\, t(x,y) / \lambda\,]}$$

nt = Optical path length



1. Normalize intensity map to 1

2. Define thickness map at some reasonable amount (100 μm max change)

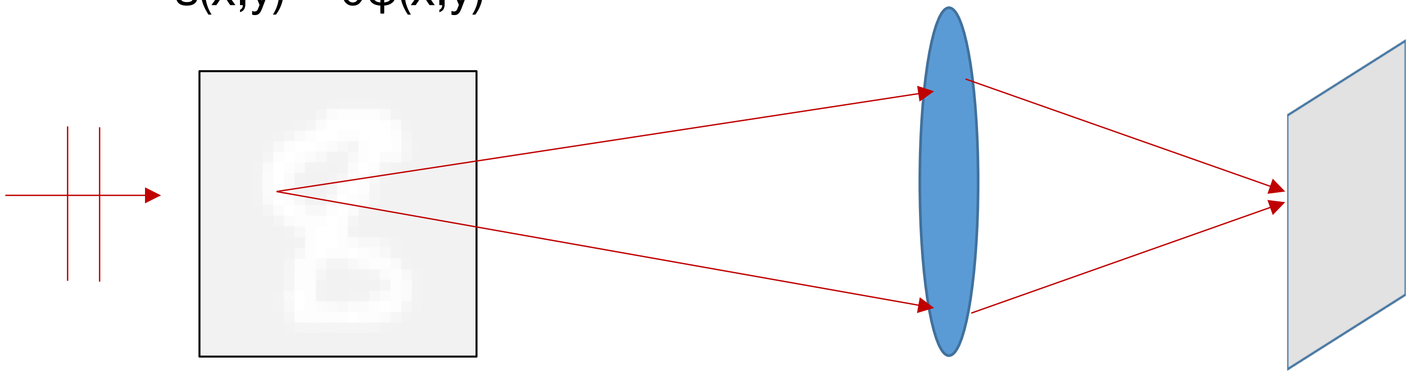
3. Convert thickness map into optical phase delay:

```
n = 1
wavelength = 0.5e-3
mnist_raw_images = tf.placeholder(tf.float32, [image_size, None])
thickness_map = mnist_raw_images/np.amax(mnist_raw_images)
mnist_phase_delay_real = cos(thickness_map * n/wavelength)
mnist_phase_delay_imag = sin(thickness_map * n/wavelength)
mnist_phase_delay = tf.complex(mnist_phase_delay_real,mnist_phase_delay_imag)
```

Example #1: Optimizing coherent illumination pattern for improved classification

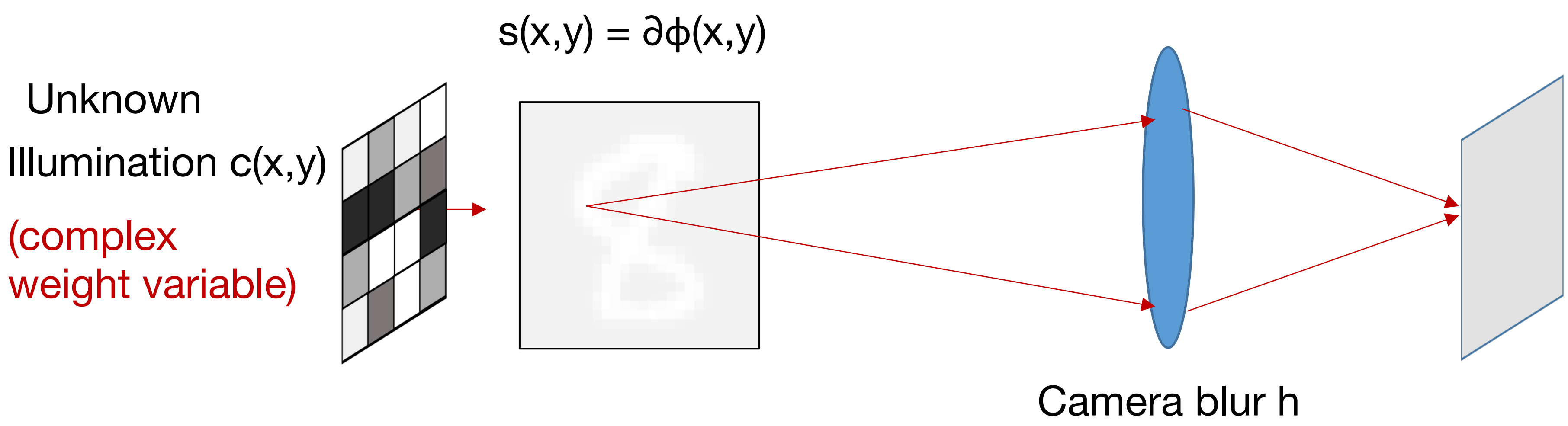
Coherent image Model: $I_c(x,y) = |h * Cs|^2$

$s(x,y) = \partial\phi(x,y)$



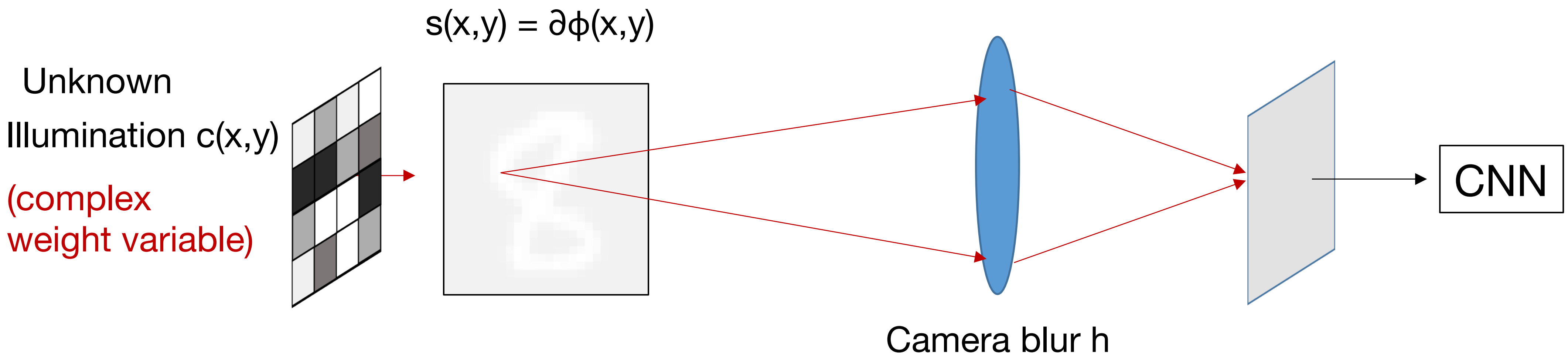
Example #1: Optimizing coherent illumination pattern for improved classification

Coherent image Model: $I_c(x,y) = |h * Cs|^2$



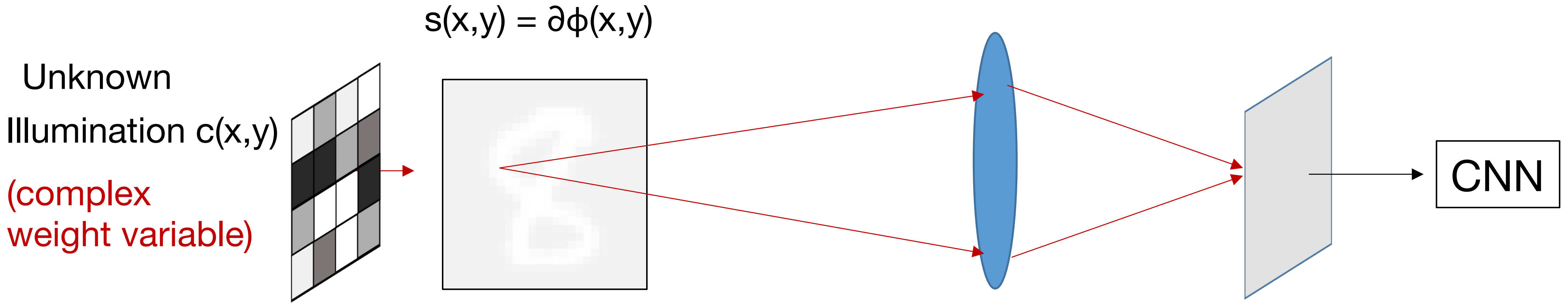
Example #1: Optimizing coherent illumination pattern for improved classification

Coherent image Model: $I_c(x,y) = |h * Cs|^2$



Example #1: Optimizing coherent illumination pattern for improved classification

Coherent image Model: $I_c(x,y) = |h * Cs|^2$




```
mnist_phase_delay = tf.reshape(mnist_phase_delay, [-1, image_size, image_size])
C0_real = tf.Variable([image_size, image_size])
C0_imag = tf.Variable([image_size, image_size])
C0_complex = tf.complex(C0_real, C0_imag)
x_C_complex = tf.mul(mnist_phase_delay, C0_complex)
image_complex = conv2d(x_C_complex, camera_blur)
detected_image = tf.complex_abs(image_complex)
```

detected_image then enters standard CNN classification pipeline

Example #2: Optimizing aperture shape for improved digit classification

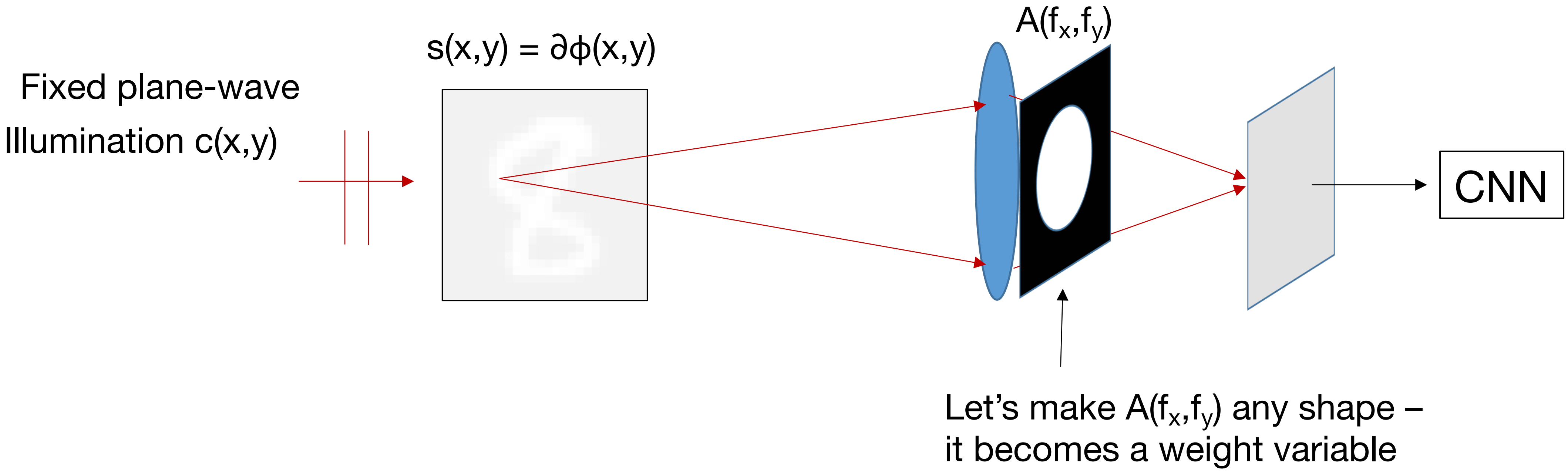
Example future situation: Hacking has brought online banking to a halt. We now rely on a special form of physical check that is made of visibly transparent plastic. To write the amount in, you press down with a pen-like instrument, and then the check is read out by shining a particular pattern of laser light onto it, and then imaging it with a lens.

Question #2: What type of aperture shape should you use to maximize classification accuracy?

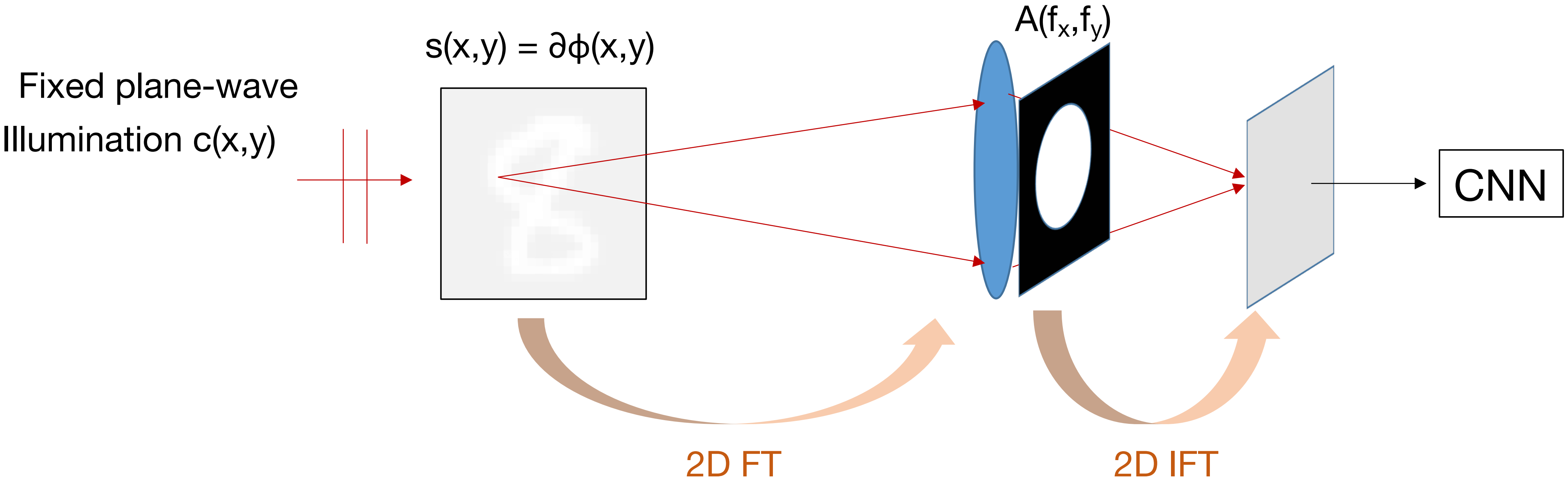
Example #2: Optimizing aperture shape for improved digit classification

Example future situation: Hacking has brought online banking to a halt. We now rely on a special form of physical check that is made of visibly transparent plastic. To write the amount in, you press down with a pen-like instrument, and then the check is read out by shining a particular pattern of laser light onto it, and then imaging it with a lens.

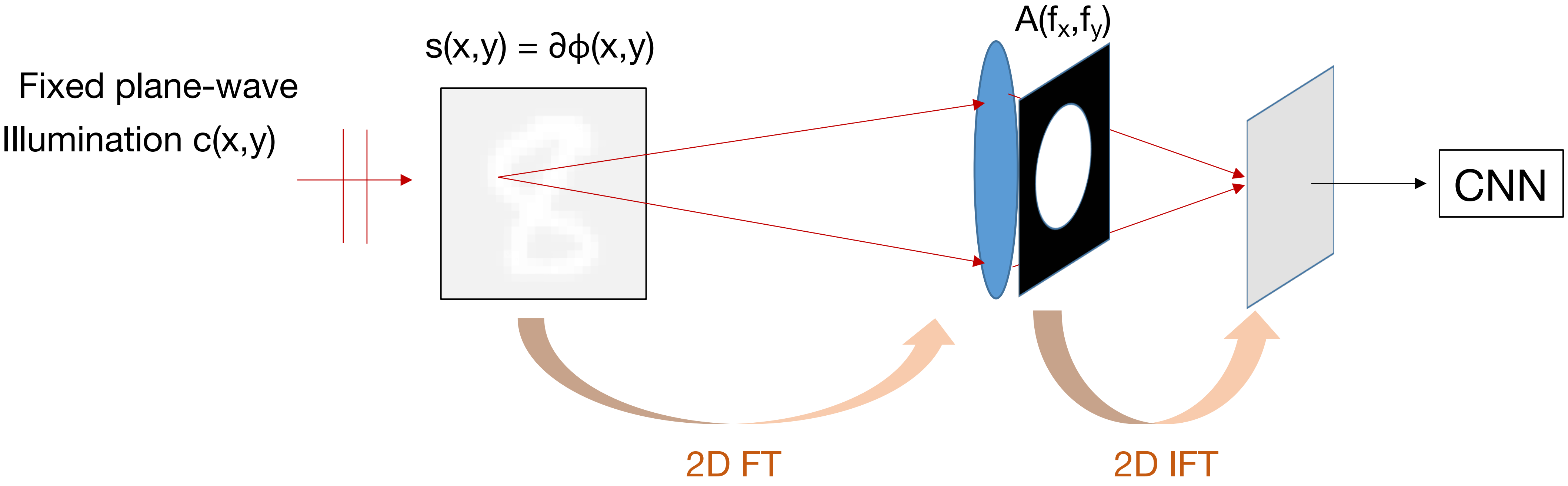
Question #2: What type of aperture shape should you use to maximize classification accuracy?



Example #2: Optimizing aperture shape for improved digit classification



Example #2: Optimizing aperture shape for improved digit classification




```
mnist_phase_delay = tf.reshape(mnist_phase_delay, [-1, image_size, image_size])
C0 = np.ones(image_size, image_size)
C0 = tf.constant(C0)
x_C_complex = tf.mul(mnist_phase_delay, C0)
fx_C_complex = tf.fft2d(x_C_complex)
ap_filter = tf.Variable([image_size, image_size])
filtered_x_C =  tf.mul(fx_C_complex, ap_filter)
image_complex = tf.ifft2d(filtered_x_C)
detected_image = tf.complex_abs(image_complex)
```

Remaining questions to address about physical layers:

- 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?

deep imaging

Physical Layers

Digital Layers

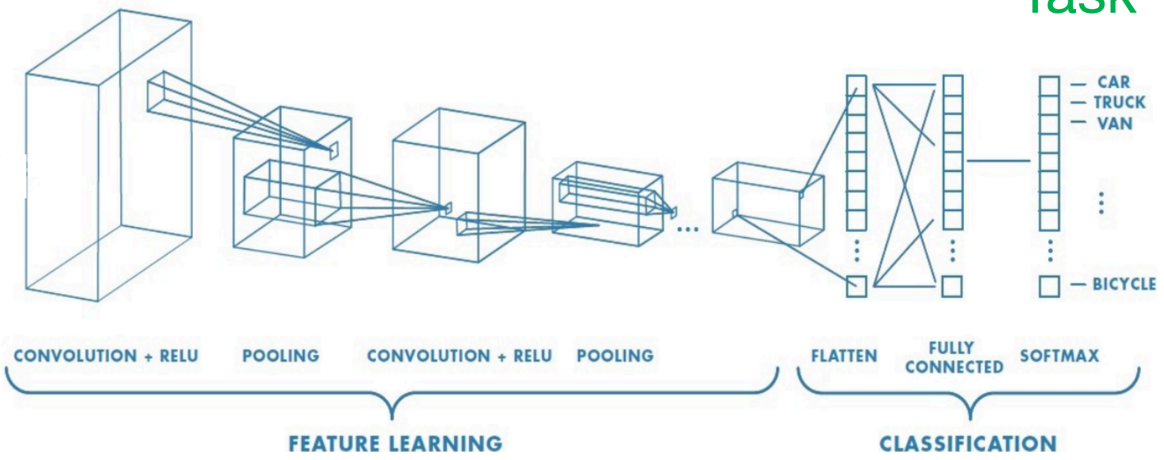
Input image data $\mathbf{I}_0$

$f[\ ]$

$\mathbf{I}_s = f[\mathbf{I}_0]$

Digitized $\mathbf{I}_s$

Task



Some Examples:

- Optimized illumination
- Optimized sensor specifications
- Number of measurements and locations
- Radiation dosage, biomarkers

deep imaging

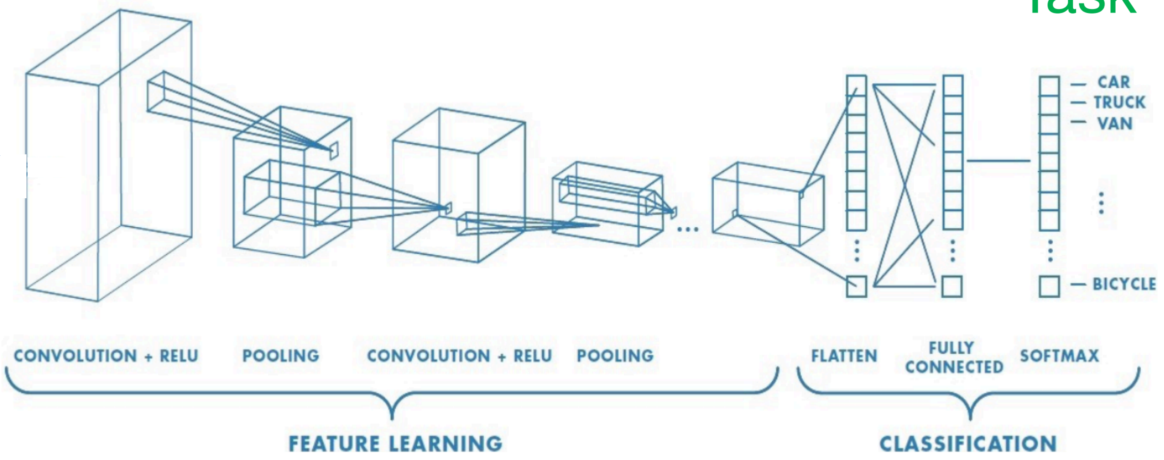
Physical Layers

Digital Layers

Input image data $\mathbf{I}_0$ → f[ ] → Digitized $\mathbf{I}_s$ →

$$\mathbf{I}_s = f[\mathbf{I}_0]$$

Task



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

Physical Layers

Digital Layers

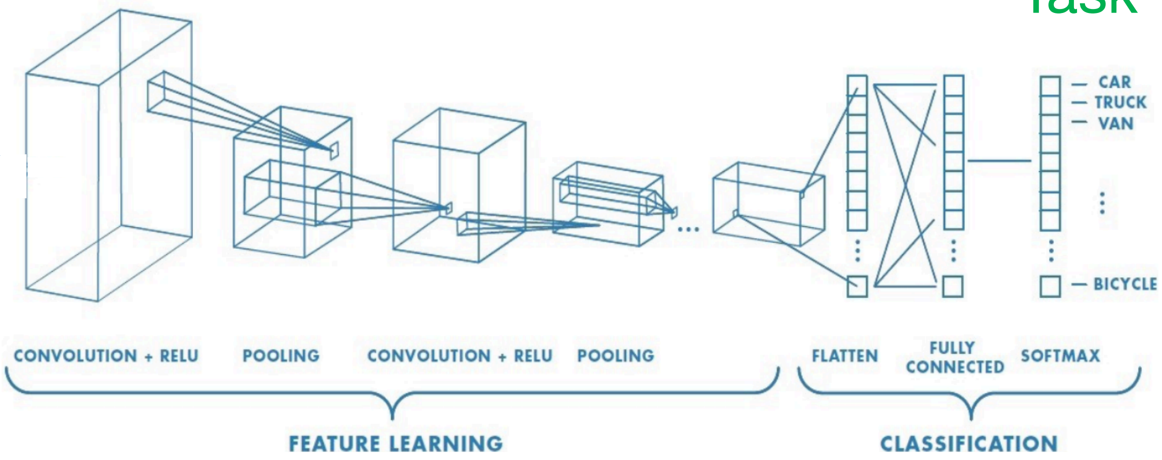
Input image data $\mathbf{I}_0$

f[ ]

Digitized $\mathbf{I}_s$

$$\mathbf{I}_s = f[\mathbf{I}_0]$$

Task



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

Full simulation of different samples and their labels

Physical Layers

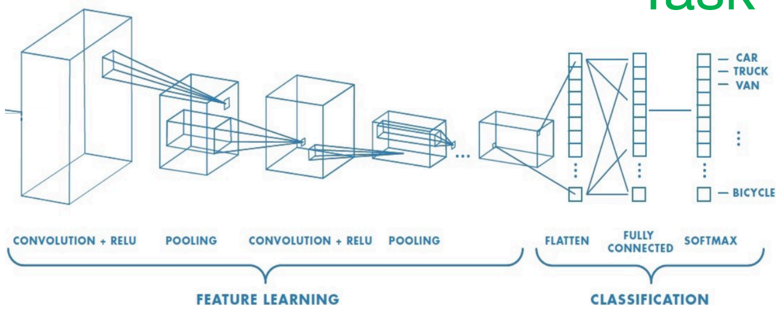
f[ ]

$$\mathbf{I_s} = f[\mathbf{I_0}]$$

Digitized $\mathbf{I_s}$

Digital Layers

Task



Simulated labels

Situation #1: Fully simulated physical layers

deep imaging

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

Full simulation of different samples and their labels

Physical Layers

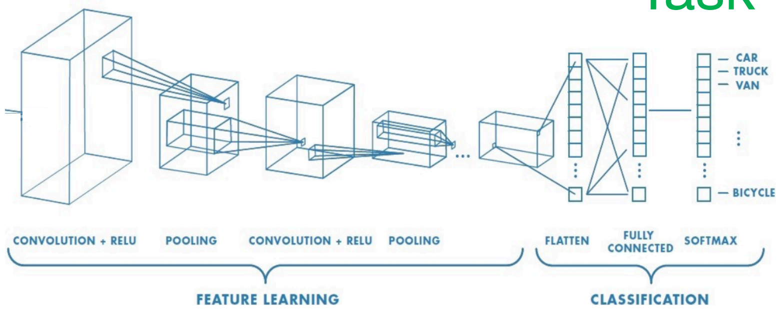
f[ ]

$$I_s = f[I_0]$$

Digitized I_s

Digital Layers

Task



Simulated labels

Examples:

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

deep imaging

Situation #1: Fully simulated physical layers

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

Existing labels

Physical Layers

Digital Layers

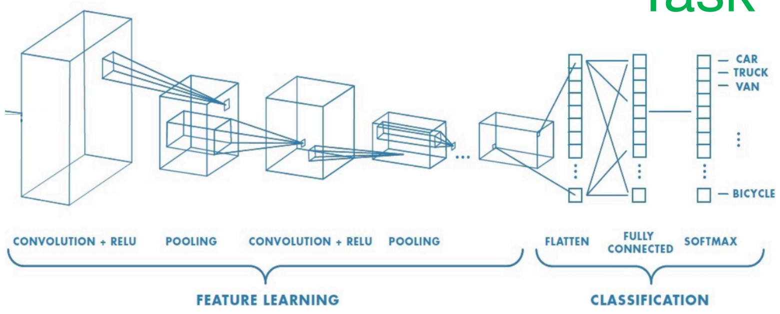
Existing annotated dataset $\mathbf{I_0}$

f[ ]

$$\mathbf{I_s} = f[\mathbf{I_0}]$$

Digitized $\mathbf{I_s}$

Task



Situation #1: Fully simulated physical layers

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

Existing labels

Existing annotated dataset $\mathbf{I_0}$

Pre-processing

Physical Layers

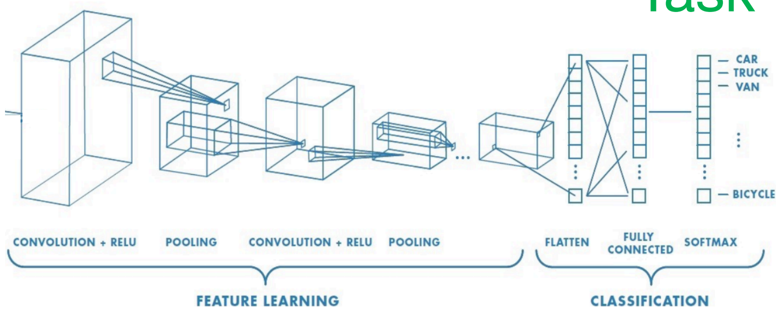
f[ ]

$$\mathbf{I_s} = f[\mathbf{I_0}]$$

Digitized $\mathbf{I_s}$

Digital Layers

Task



Situation #1: Fully simulated physical layers

deep imaging

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

Existing labels

Physical Layers

Digital Layers

Existing annotated dataset $\mathbf{I}_0$

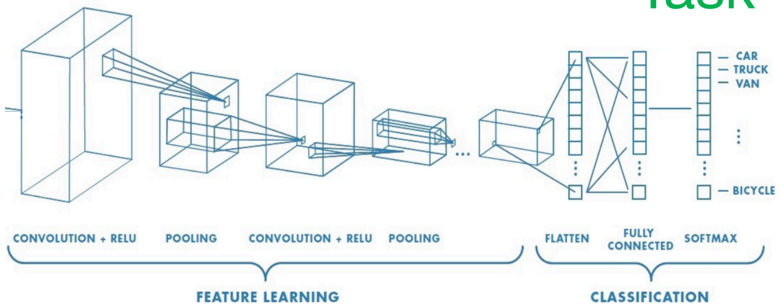
Pre-processing

f[ ]

$$\mathbf{I}_s = f[\mathbf{I}_0]$$

Digitized $\mathbf{I}_s$

Task



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

ML Optimization

Python/Matlab/other

→ Big dataset → TensorFlow

TensorFlow → TensorFlow

Situation #1: Fully simulated physical layers

deep imaging

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

Simulation and/or pre-processing

ML Optimization

Python/Matlab/other → Big dataset → 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

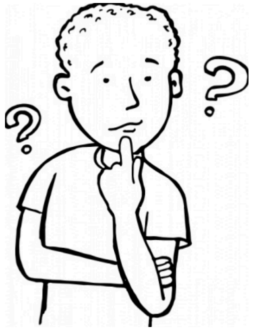
Cons: Harder to bug-check /compare to prior work if closely integrated

Situation #2: Experimentally-driven physical layers

Generated Labels

Physical Layers

Digital Layers



Experimental measurements

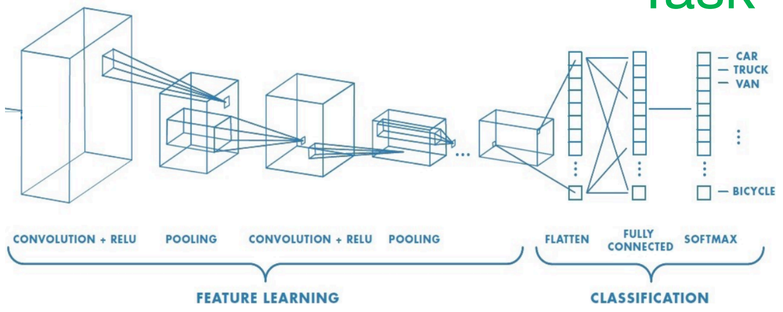
"Expert" annotation

f[ ]

$$\mathbf{I}_s = f[\mathbf{I}_0]$$

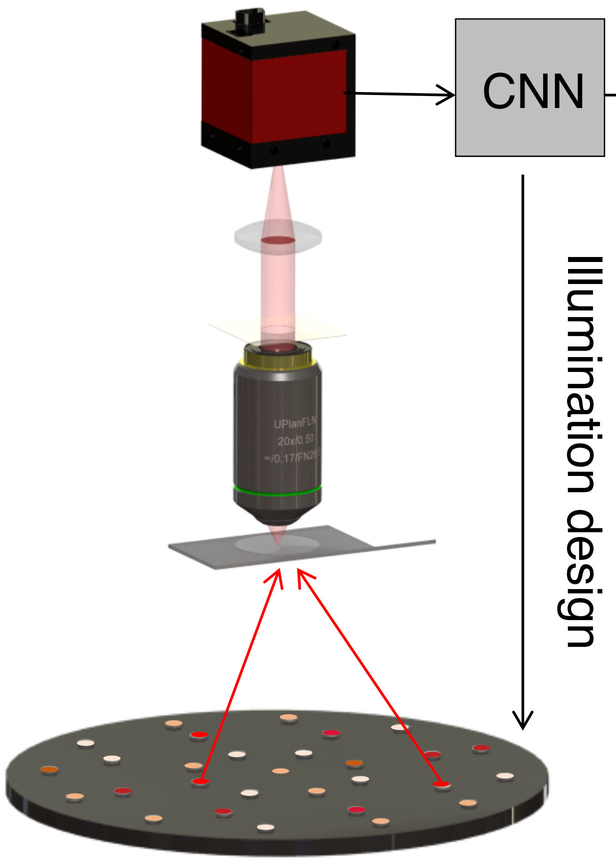
Digitized $\mathbf{I}_s$

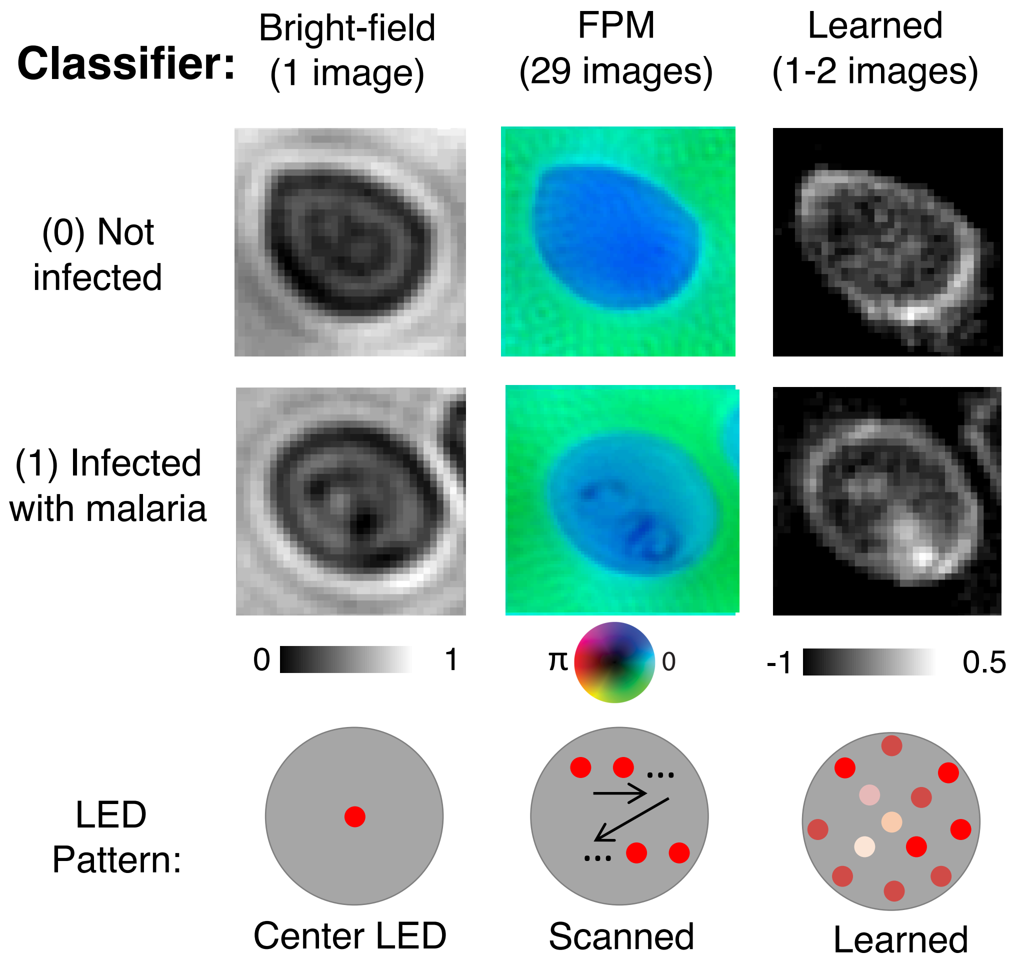
Task



Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:



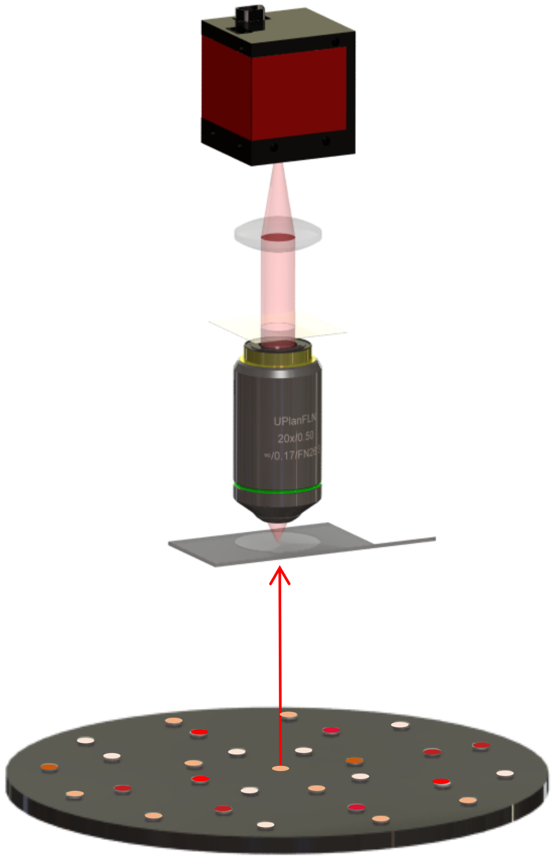


Classification Accuracy: **75%** **98.5%** **97%**

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

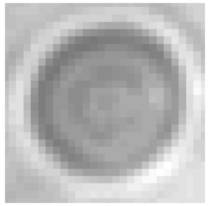
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:



Data set for physical layer optimization:

Turn on LED 1, capture image 1:

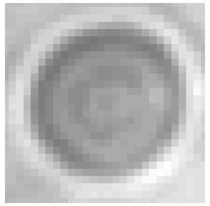


Situation #2: Experimentally-driven physical layers

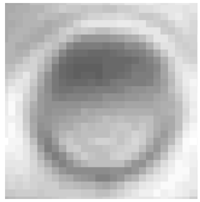
Example: CNN-Optimized illumination for classification of malaria:

Data set for physical layer optimization:

Turn on LED 1, capture image 1:



Turn on LED 1, capture image 2:

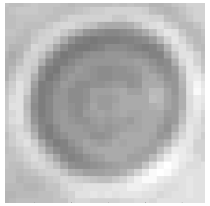


Situation #2: Experimentally-driven physical layers

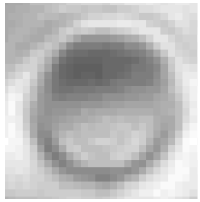
Example: CNN-Optimized illumination for classification of malaria:

Data set for physical layer optimization:

Turn on LED 1, capture image 1:

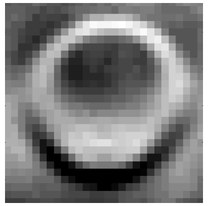


Turn on LED 1, capture image 2:



⋮

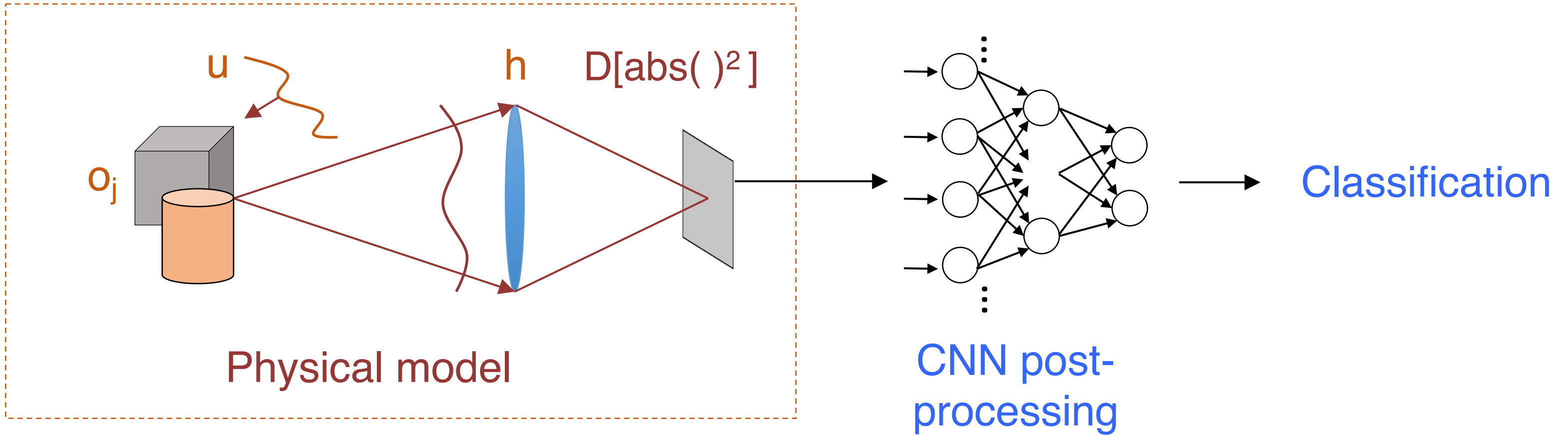
Turn on LED 32, capture image 32:



Save all 32 images (96 with 3 colors)

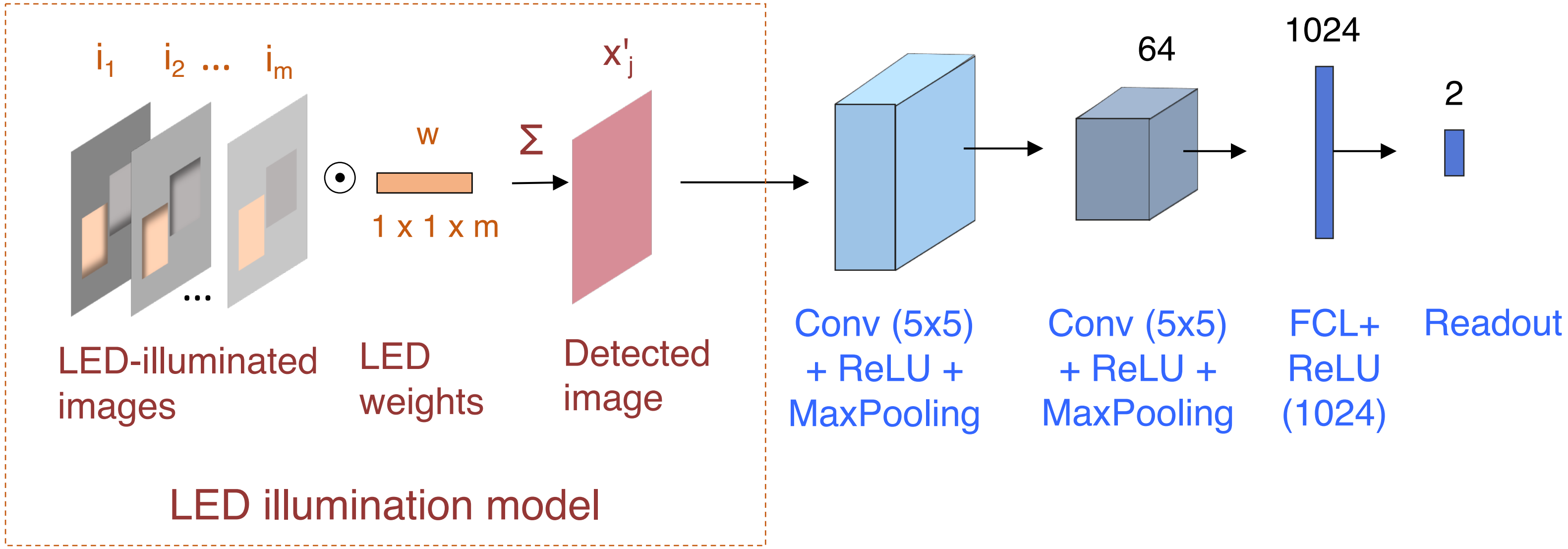
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:



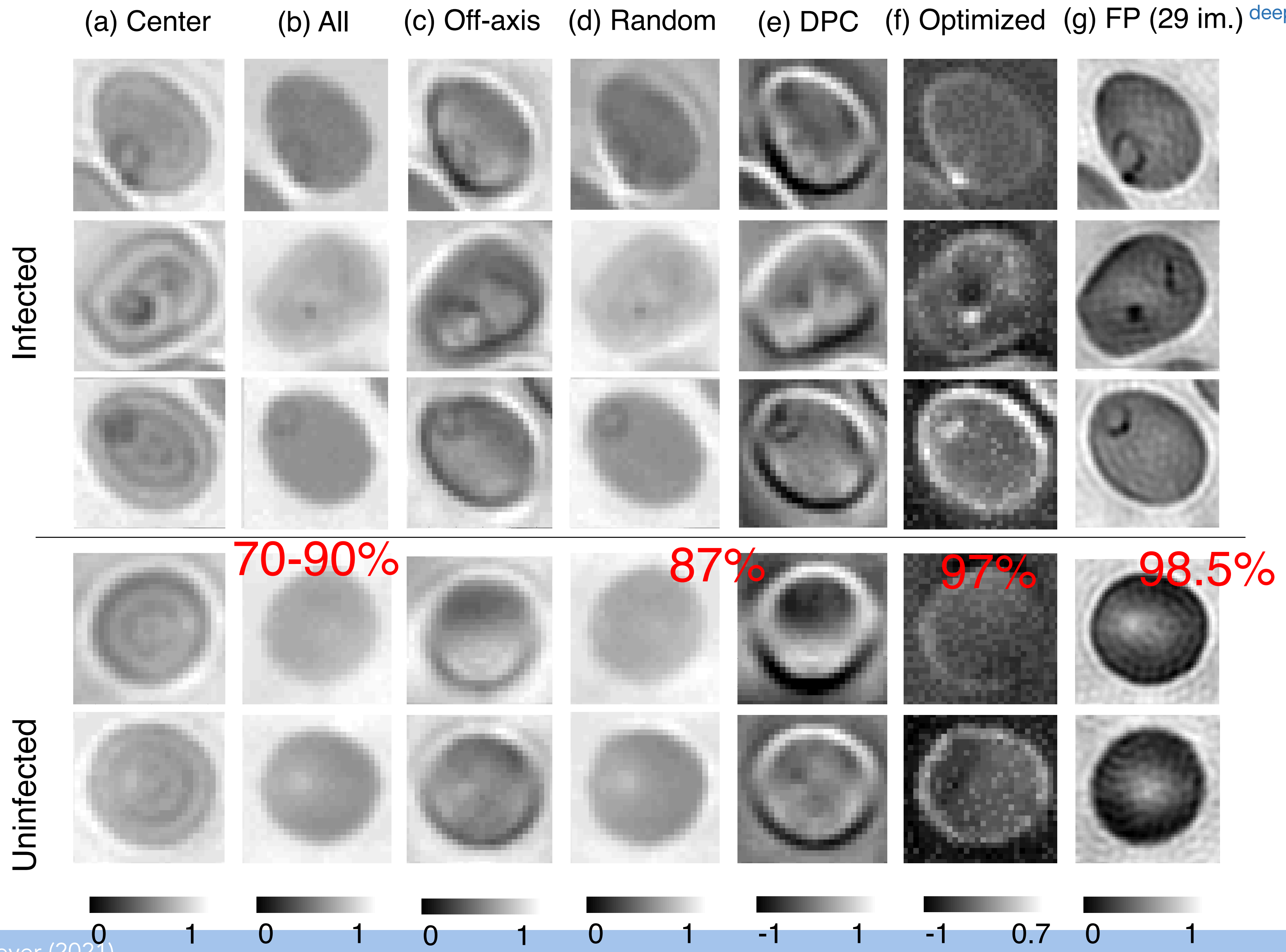
Physical layer:

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



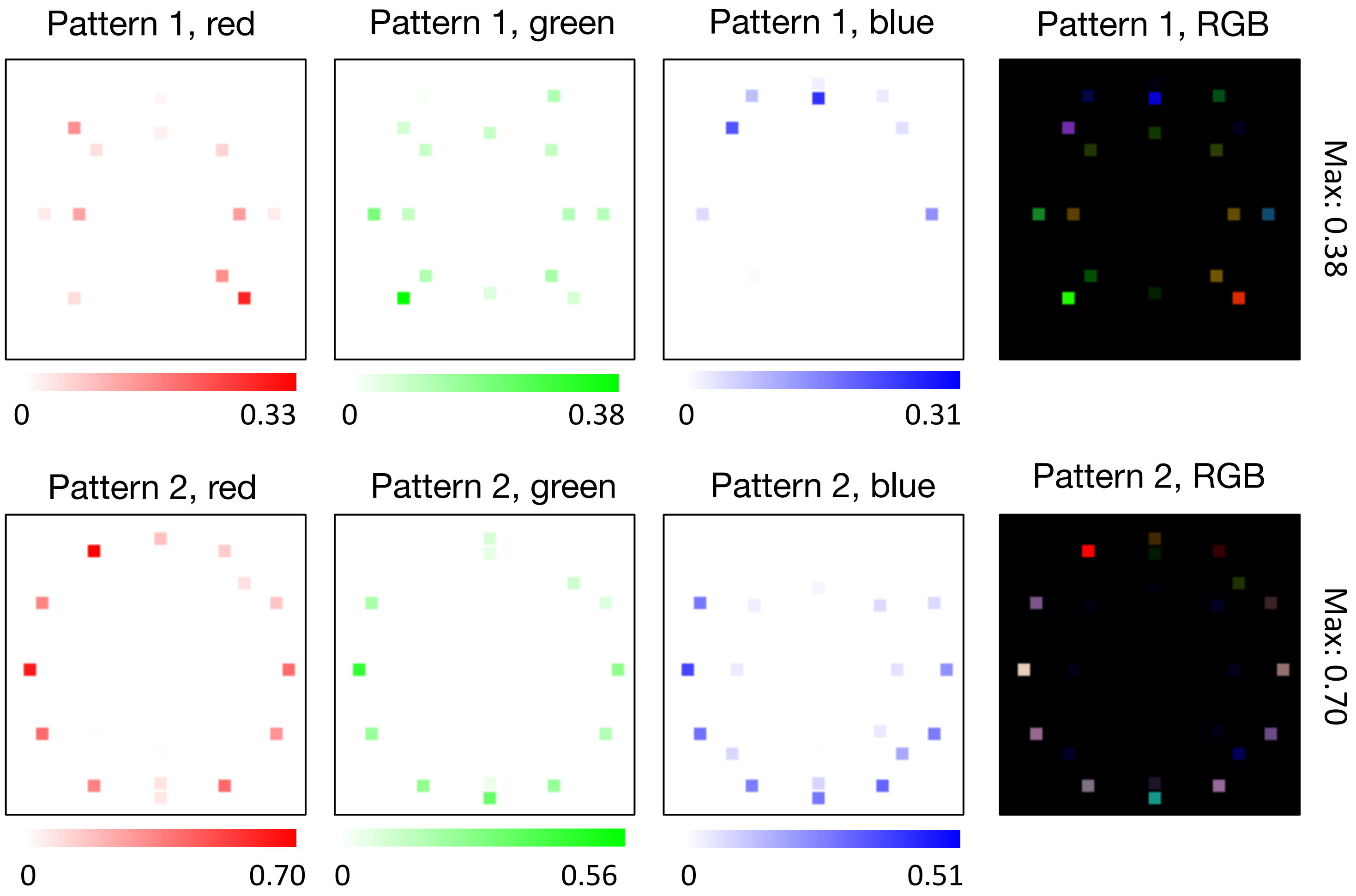
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:



Situation #2: Experimentally-driven physical layers

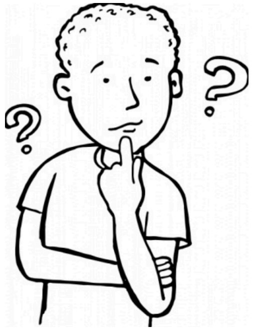
Optimized color LED patterns to classify malaria



Situation #2: Experimentally-driven physical layers

deep imaging

Generated Labels



Physical Layers

Digital Layers

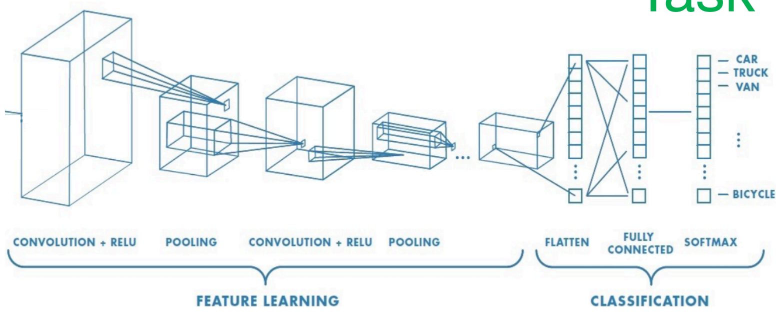
Task

Experimental measurements

"Expert" annotation

f[ ]

Digitized $\mathbf{I}_s$



$$\mathbf{I}_s = f[\mathbf{I}_0]$$

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

<u>Con's of experimental measurements:</u> 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?

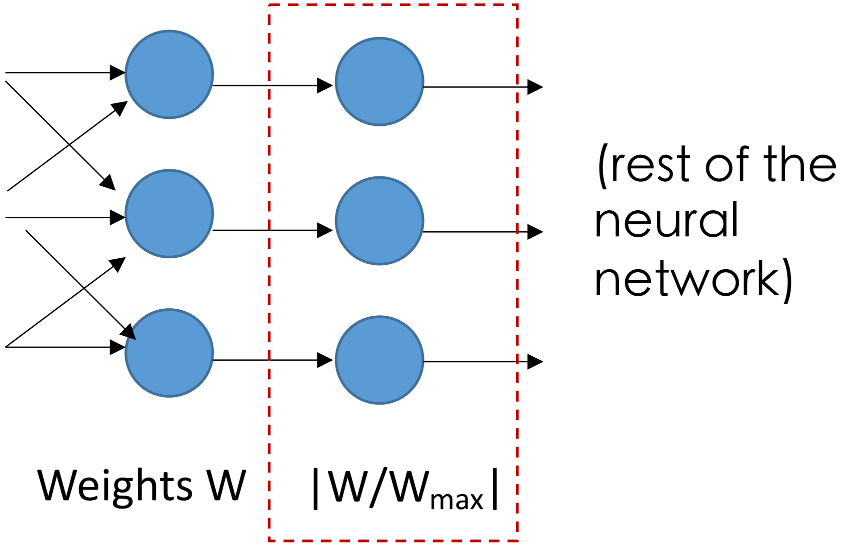
Example: Constrain weights to be non-negative values less than one

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)

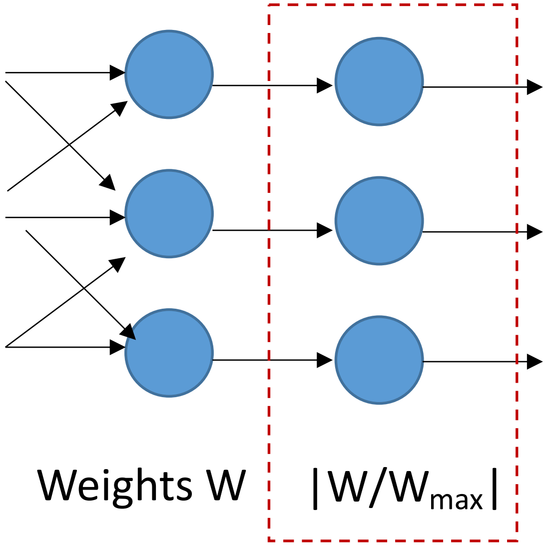


How can I add some constraints to my physical weights?

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)



(rest of the neural network)

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

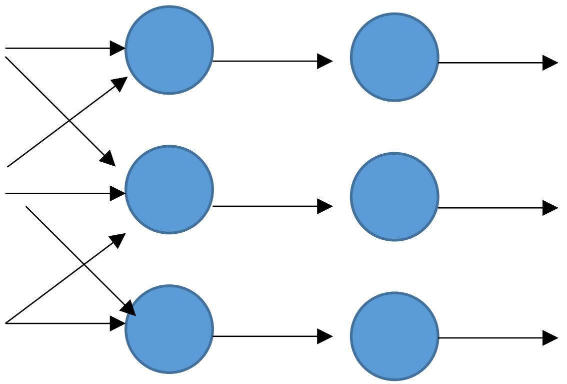
Solution: Perform annealing with a temperature parameter

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

Increase α with iteration number

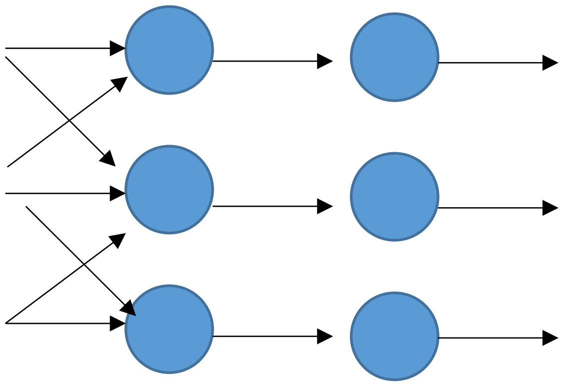
Soft-max(x) = exp(-x)/ Σ exp(-x)

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?

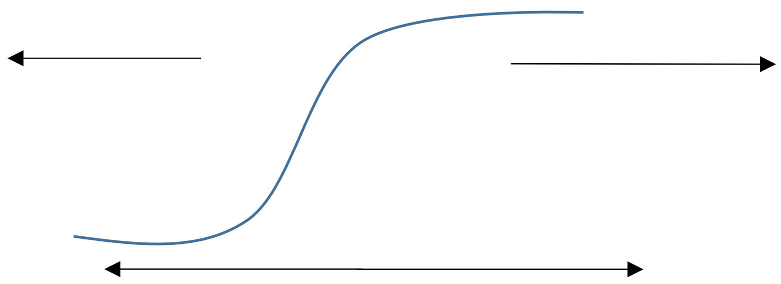
Example: Constrain weights to be either 0 or 1

Solution: Perform annealing with a temperature parameter



(rest of the neural network)

Drive w to be large, so softmax(w) -> 0 or 1



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

Increase α with iteration number

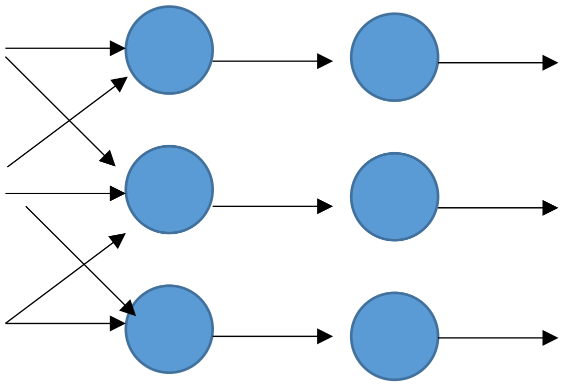
Soft-max(x) = exp(-x)/ Σ exp(-x)

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)

Pros:

- Works pretty well
- Flexibly address convergence issues

Cons:

- A bit sensitive

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

Increase α with iteration number

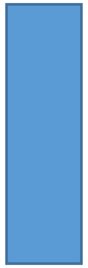
Soft-max(x) = exp(-x)/ Σ exp(-x)

What are some common issues and pitfalls with physical layers?

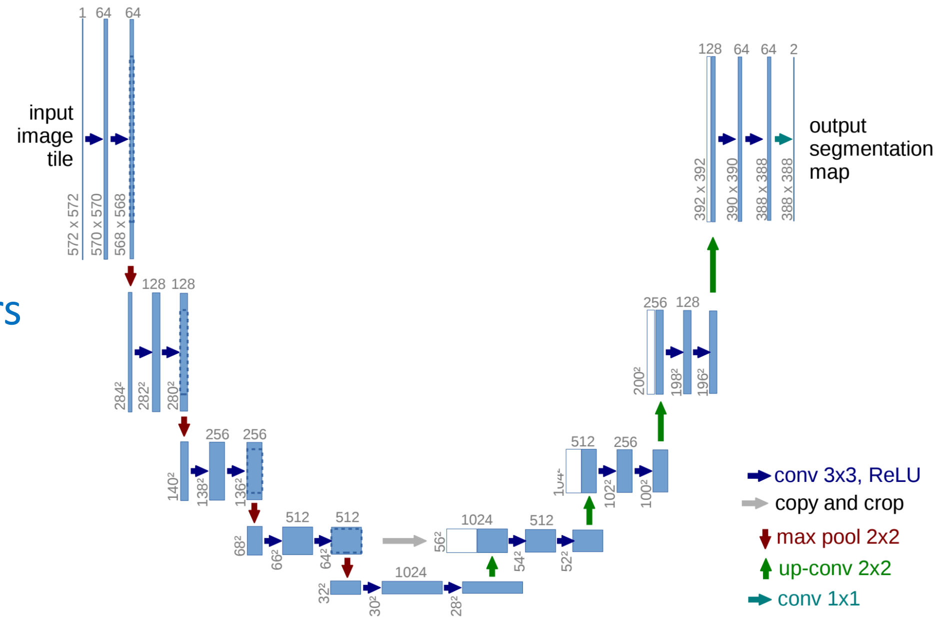
- Most common issue – you have a bug in your CNN!
    - Solution: “Disable “ physical layer (set to constant), and get network to work!
    - Good practice: always compare performance with and without physical layer
- Another common challenge - vanishing gradients

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

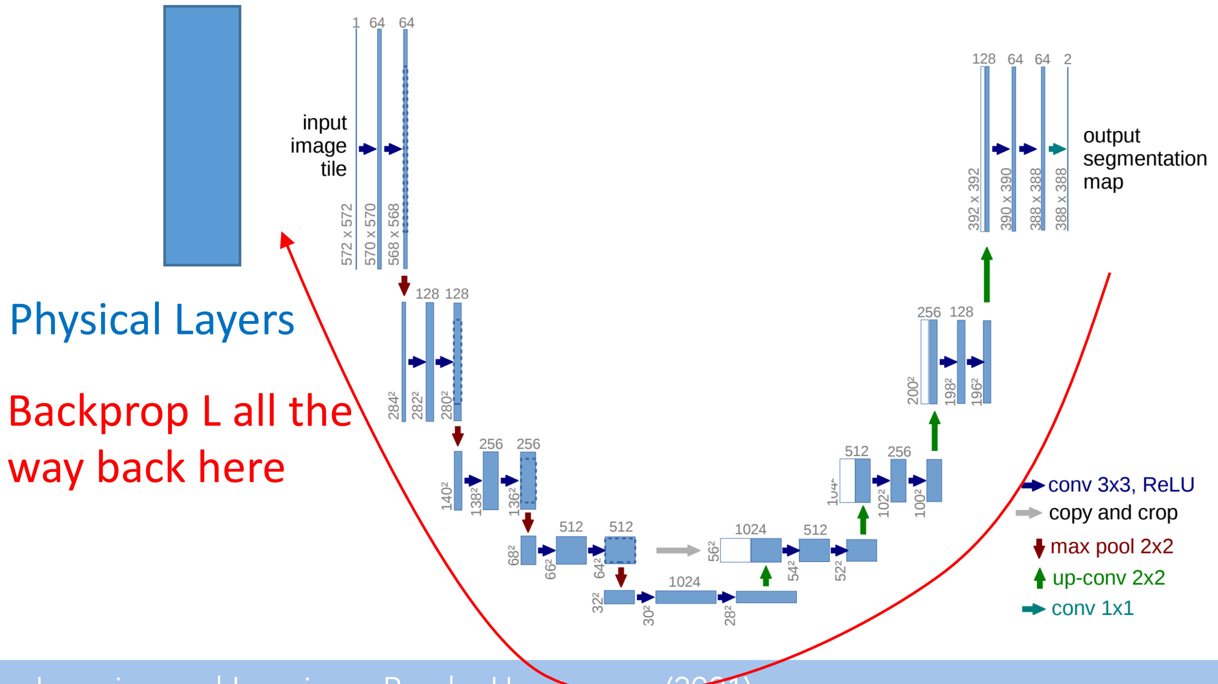


Physical Layers



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



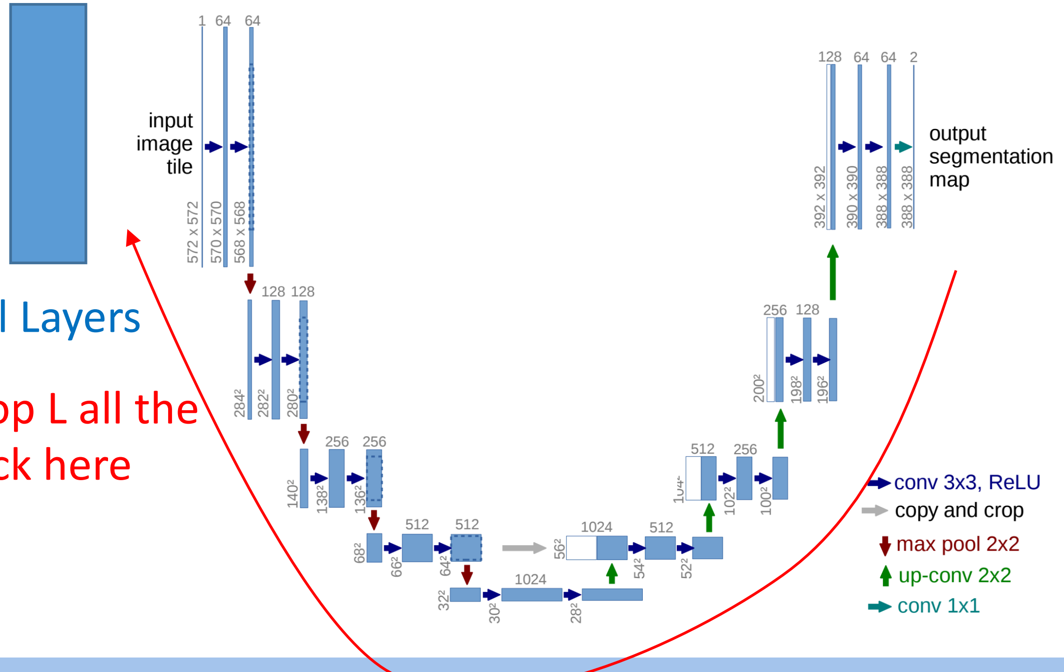
deep imaging

What are some common issues and pitfalls with physical layers?

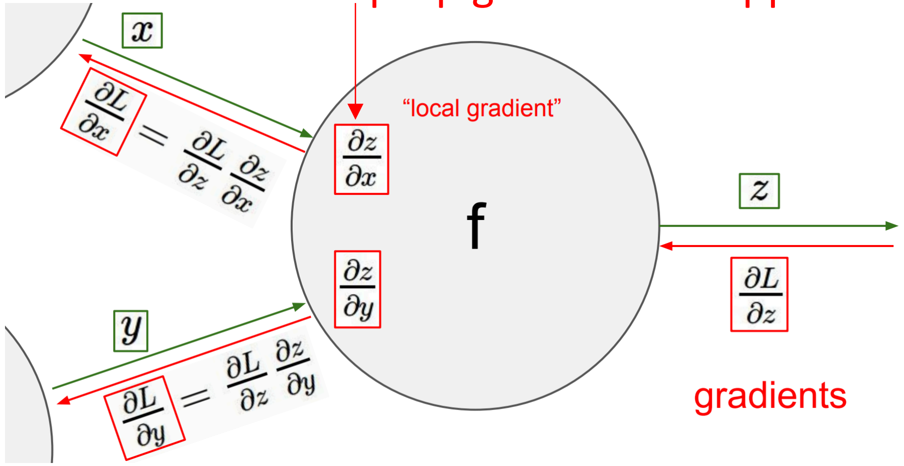
- Most common issue – you have a bug in your CNN!
  - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients

Physical Layers

Backprop L all the way back here



What if this is 0?
Back-prop gradients disappear!

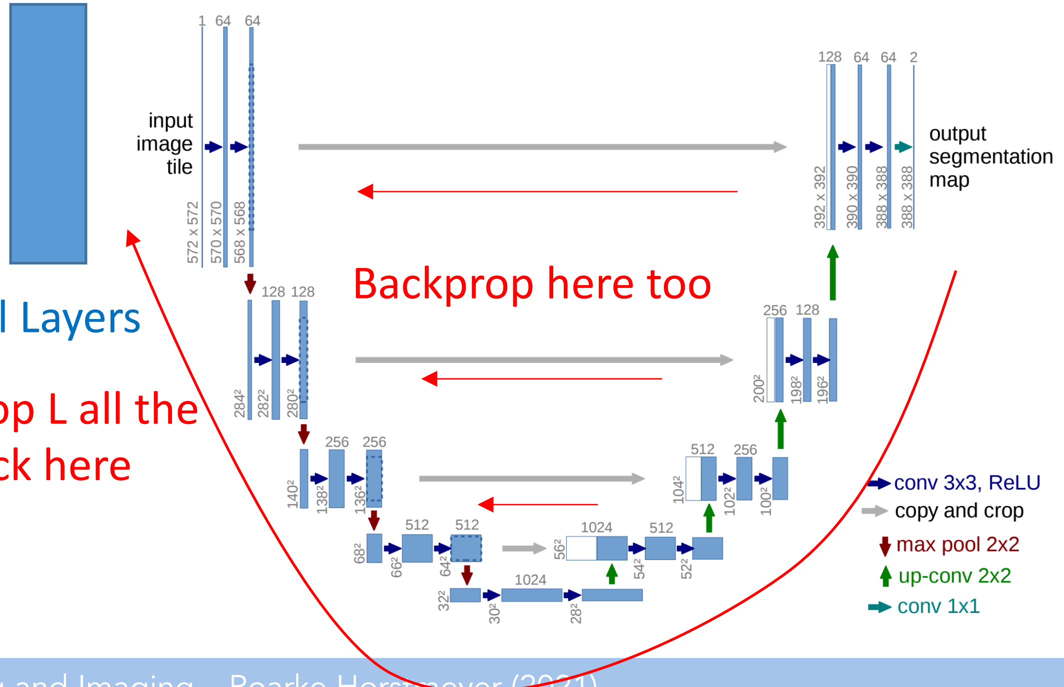


From Stanford CS231n

deep imaging

What are some common issues and pitfalls with physical layers?

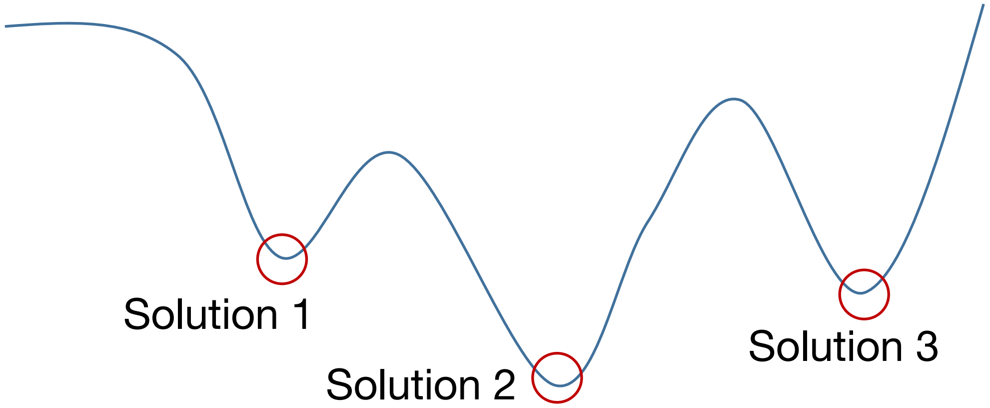
- Most common issue – you have a bug in your CNN!
    - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients



Solution: Introduce skipped connections

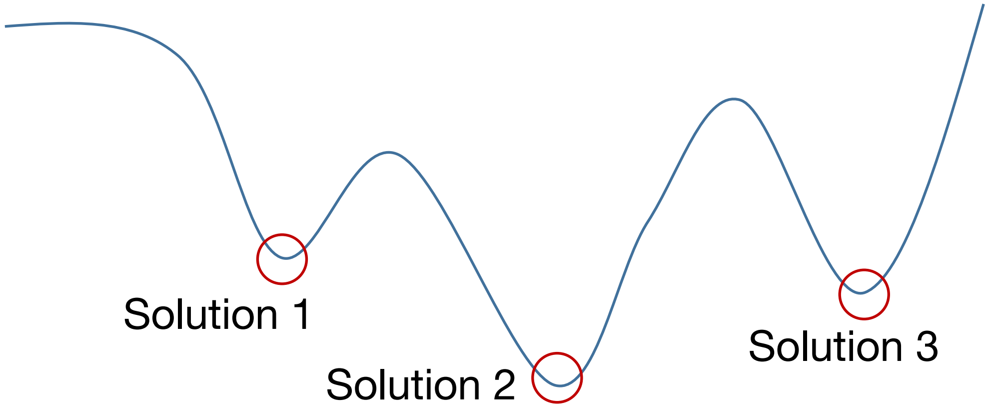
What are some common issues and pitfalls with physical layers?

- 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
- Third issue - physical layer results are not very repeatable...



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
- Third issue - physical layer results are not very repeatable...



Effective Solution: Add a small amount of noise to the physical layer output:

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

(tf.keras.layers.GaussianNoise)

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:

Simulated "Real-world" objects → Forward model → data → Inverse problem solver → Simulated "Real-world" objects

Noise, perturbations, etc. → Forward model

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:

Simulated "Real-world" objects → Forward model → data → Inverse problem solver → Simulated "Real-world" objects

Known, easy

Noise, perturbations, etc.

Hard to get working…

But you have insights and guarantees!

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:

Simulated "Real-world" objects → Forward model (Known, easy) → data → Inverse problem solver (Hard to get working…) → Simulated "Real-world" objects

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:

Simulated "Real-world" objects → Forward model → data → CNN as an inverse model → Simulated "Real-world" objects

Noise, perturbations, 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:

Simulated "Real-world" objects → Forward model → data → CNN as an inverse model → Simulated "Real-world" objects

Known, easy

Noise, perturbations, etc.

Also easy….

But it's a black box!