

Lecture 18: Coherent physical layers and general guidelines

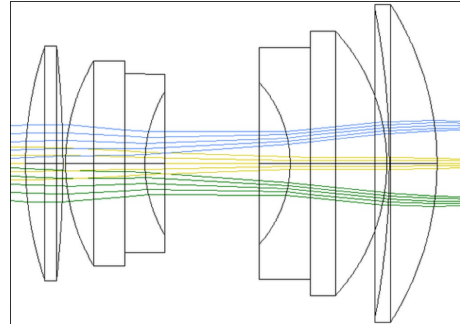
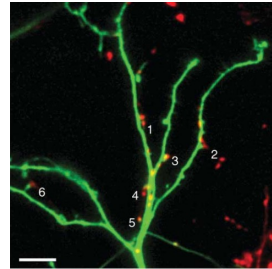
Machine Learning and Imaging

BME 548L

Roarke Horstmeyer

Summary of two models for image formation

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

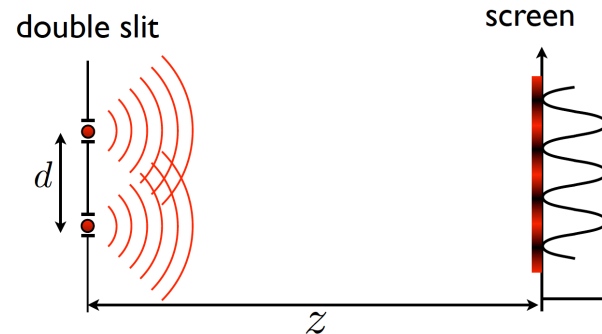
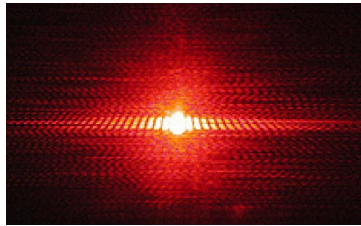


- Real, non-negative
- Models absorption and brightness

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

$$I_s = \mathbf{H B S}_0$$

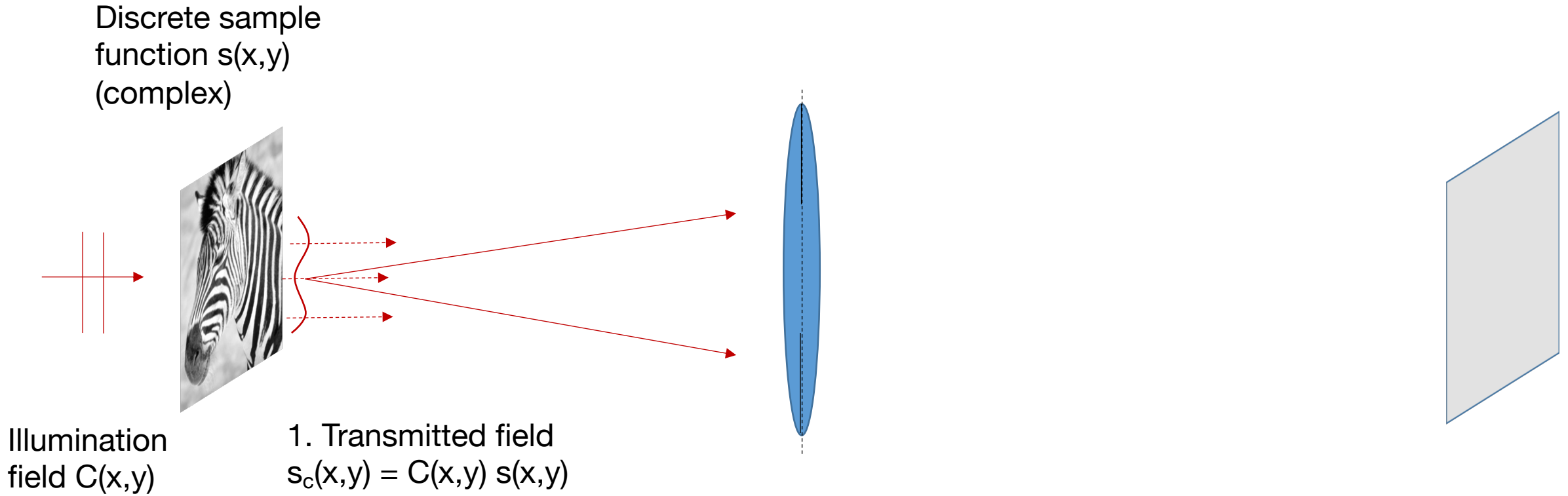
- Interpretation #2: Electromagnetic wave (*Coherent*)
- Model: Waves



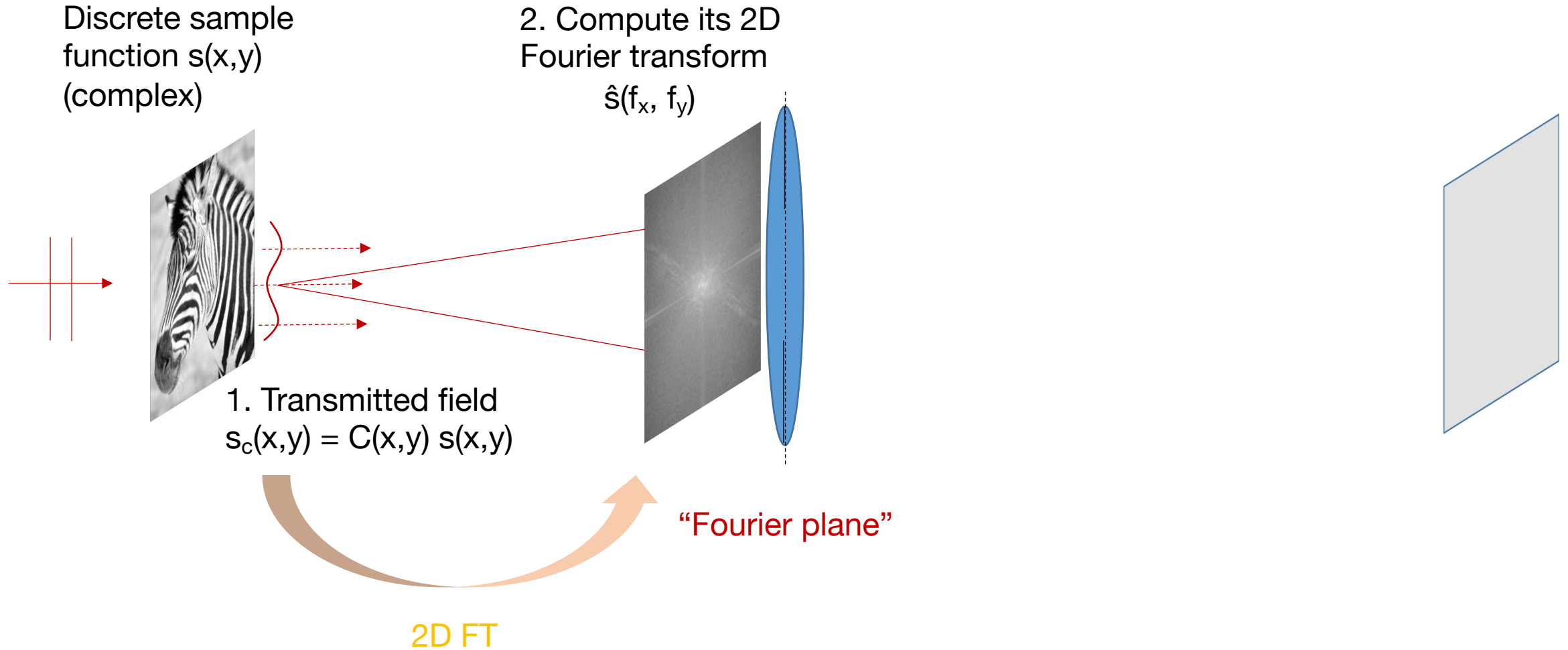
- Complex field
- Models Interference

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

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

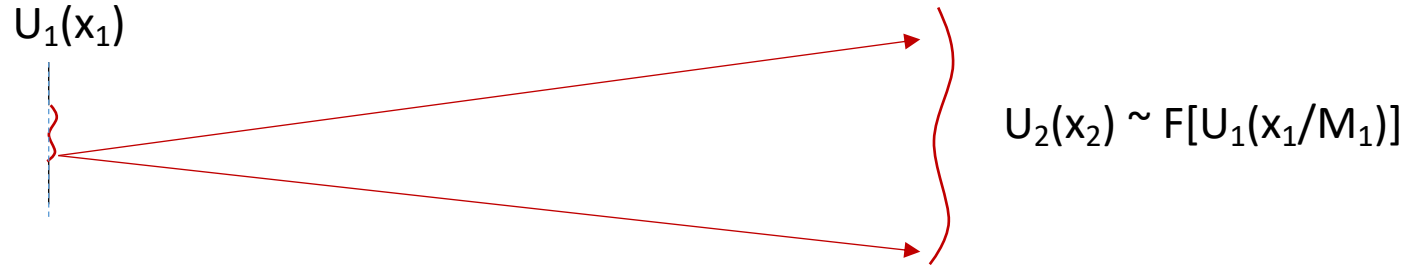


Model of image formation for wave optics (coherent light):

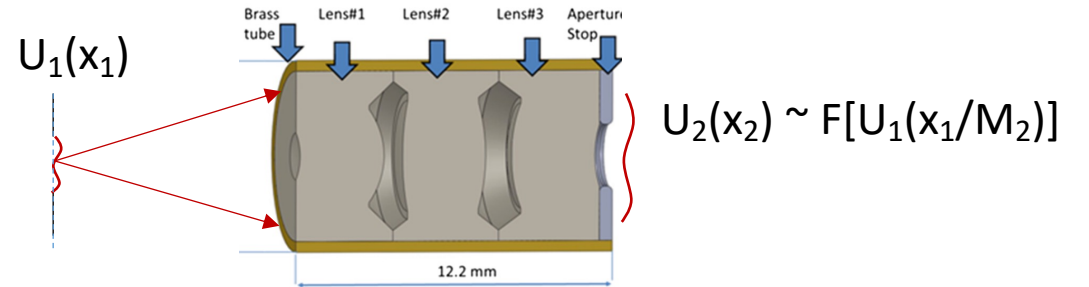


General rules for applying the Fourier transform in optics

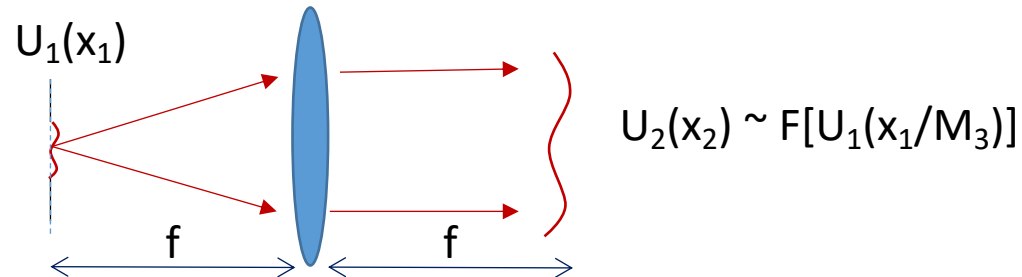
Situation 1: From an object to a plane “really far away”



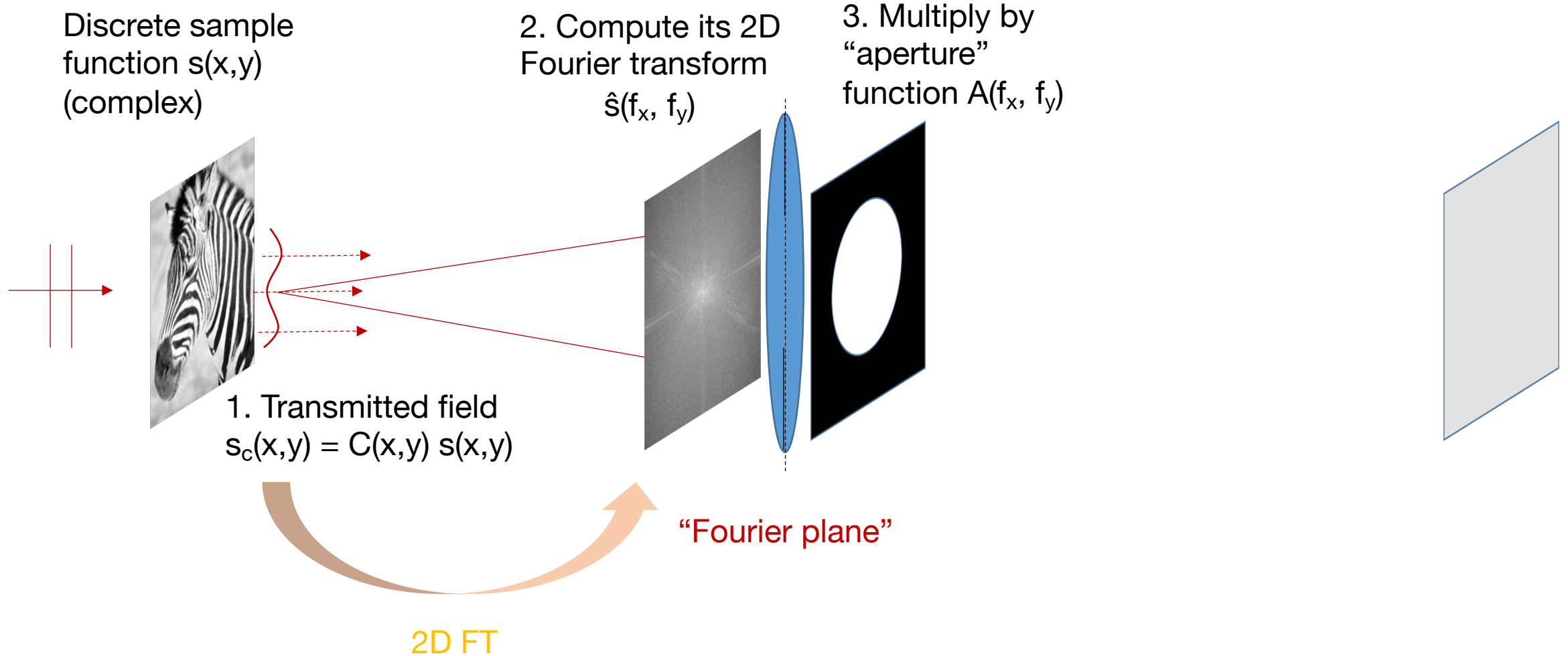
Situation 2: From an object to the back focal plane of the microscope objective lens



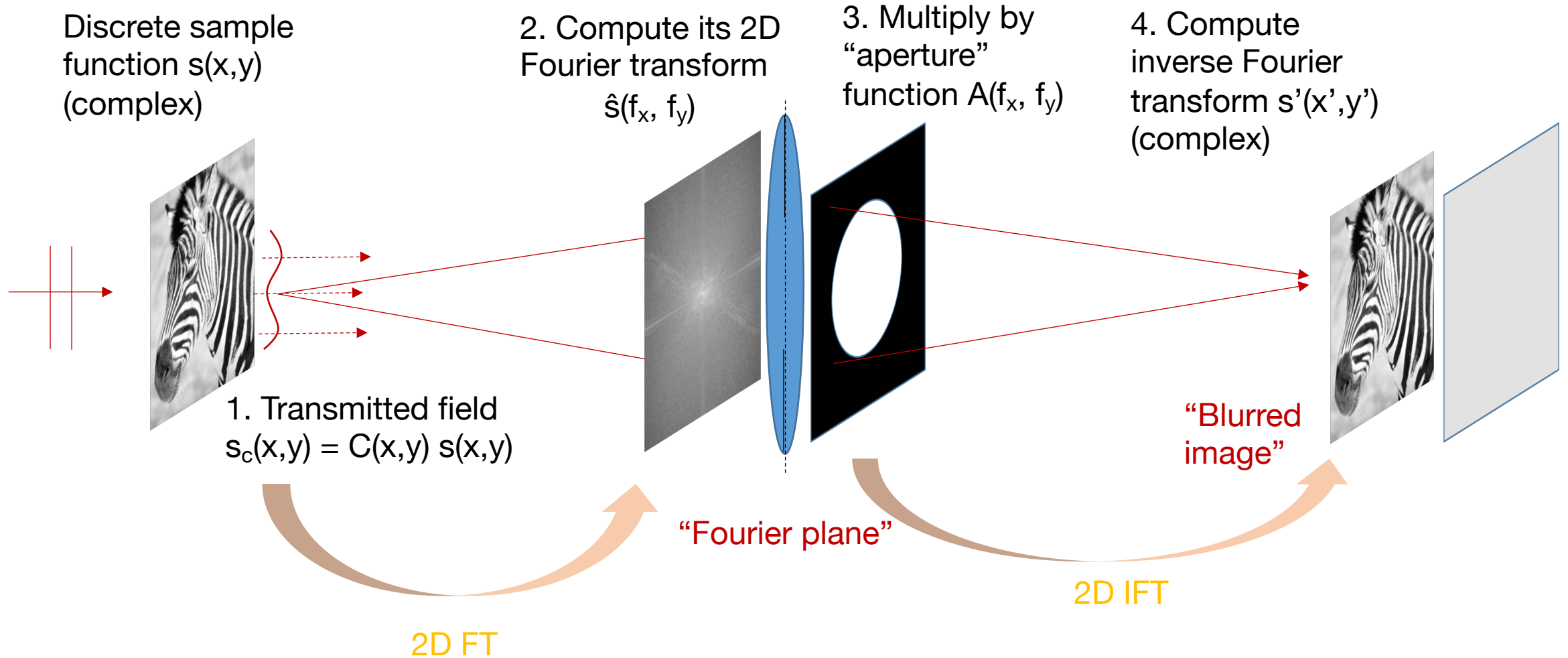
Situation 3: From an object to a plane 1 focal length away from a lens (1f-1f system)



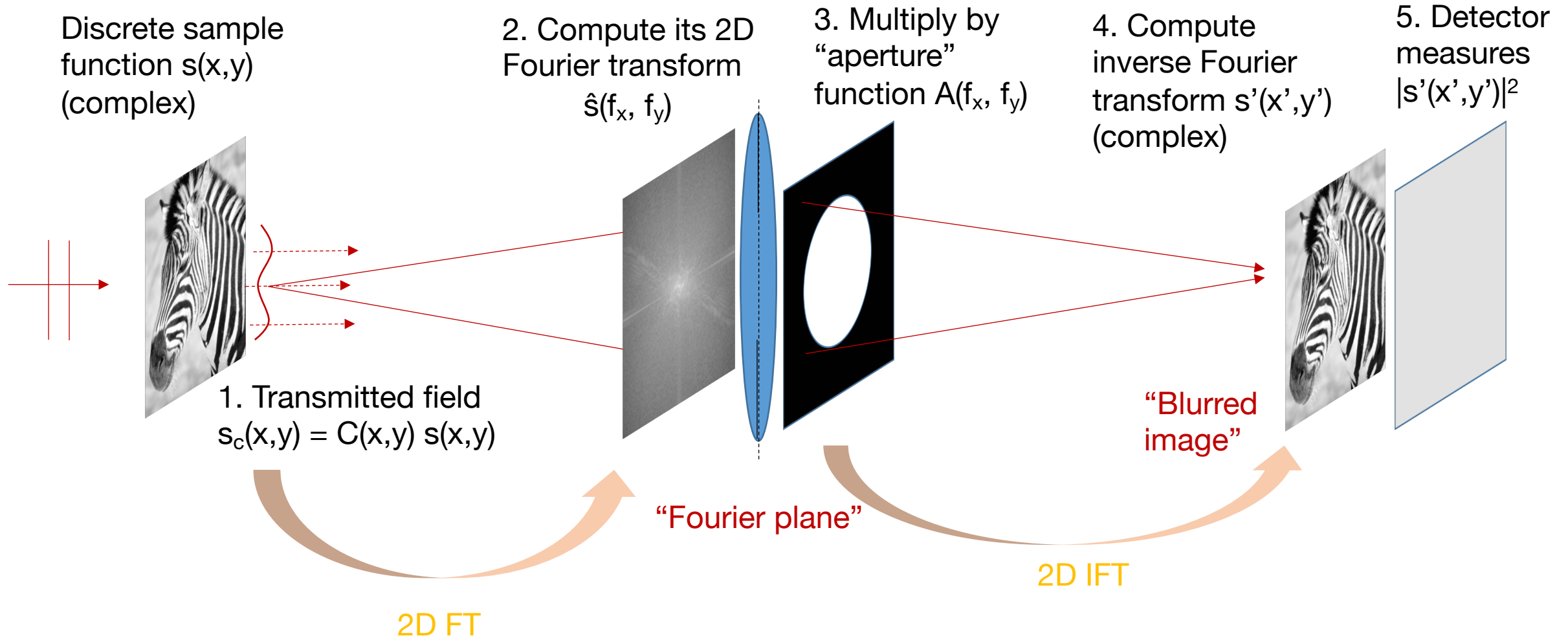
Model of image formation for wave optics (coherent light):



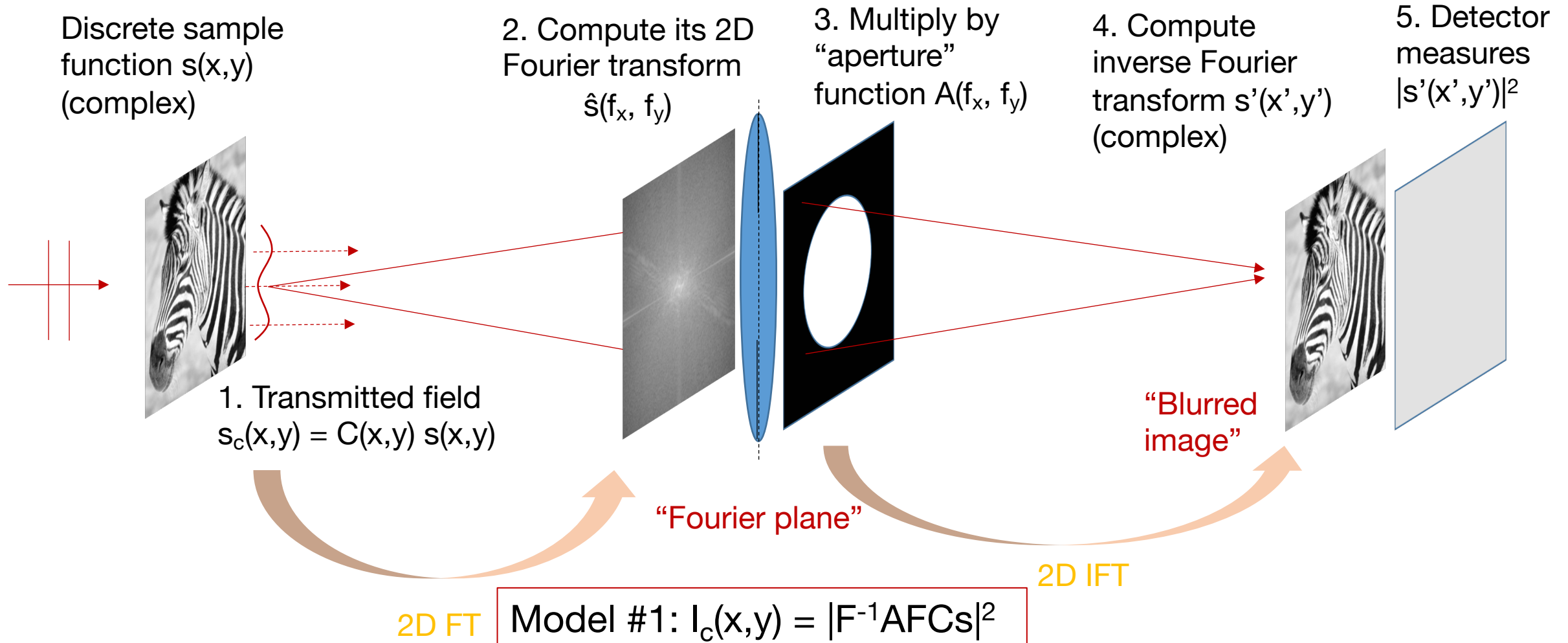
Model of image formation for wave optics (coherent light):



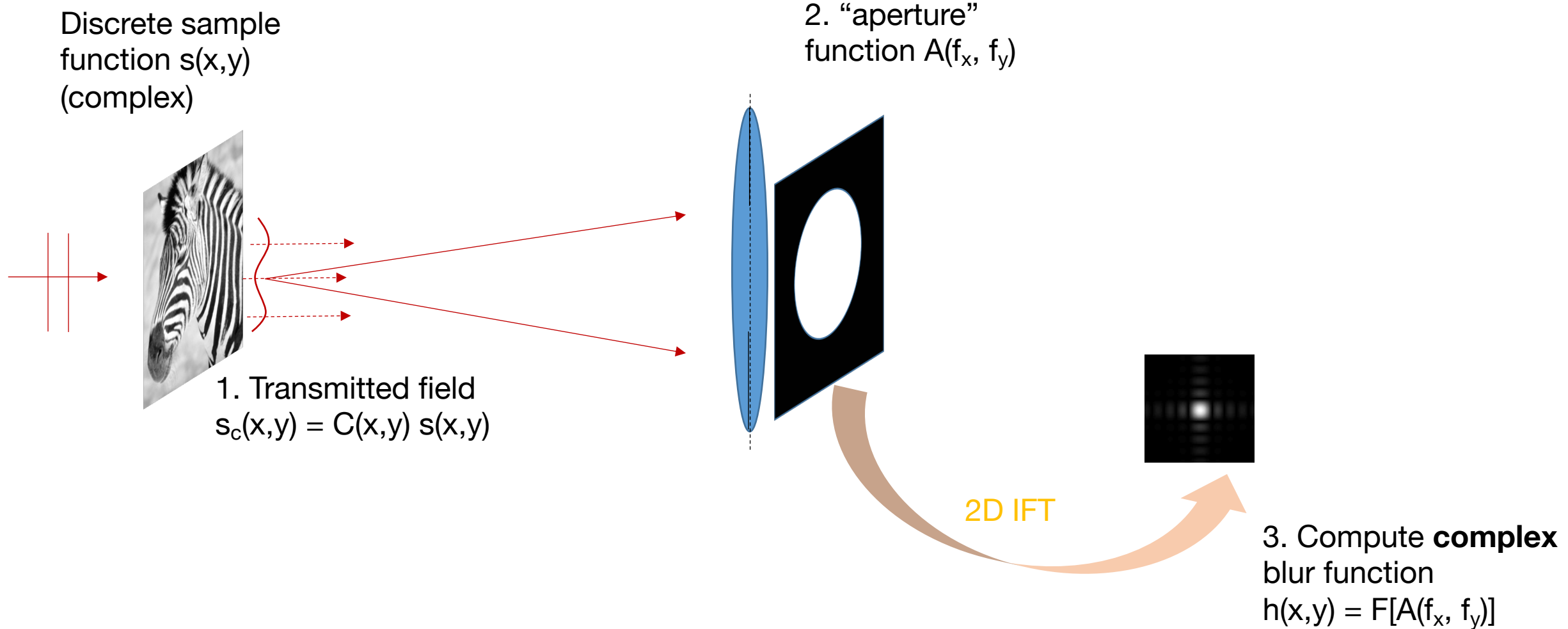
Model of image formation for wave optics (coherent light):



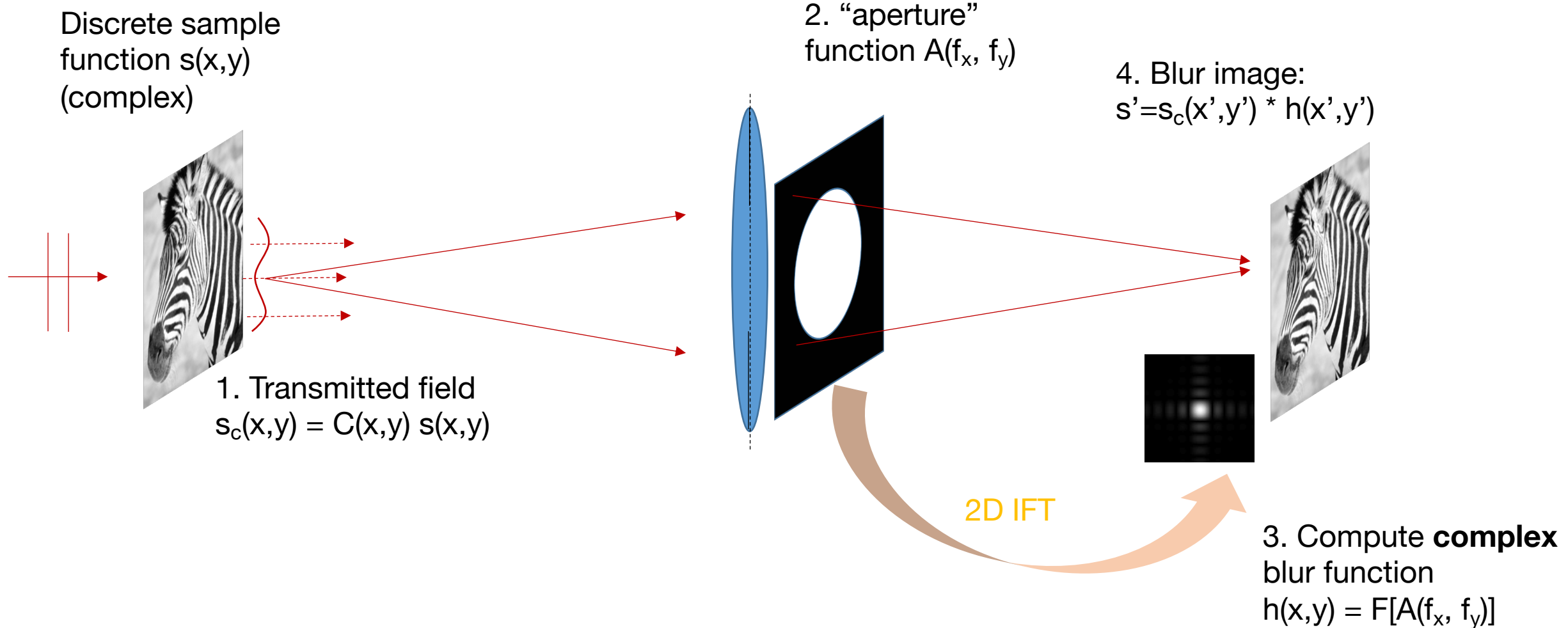
Model of image formation for wave optics (coherent light):



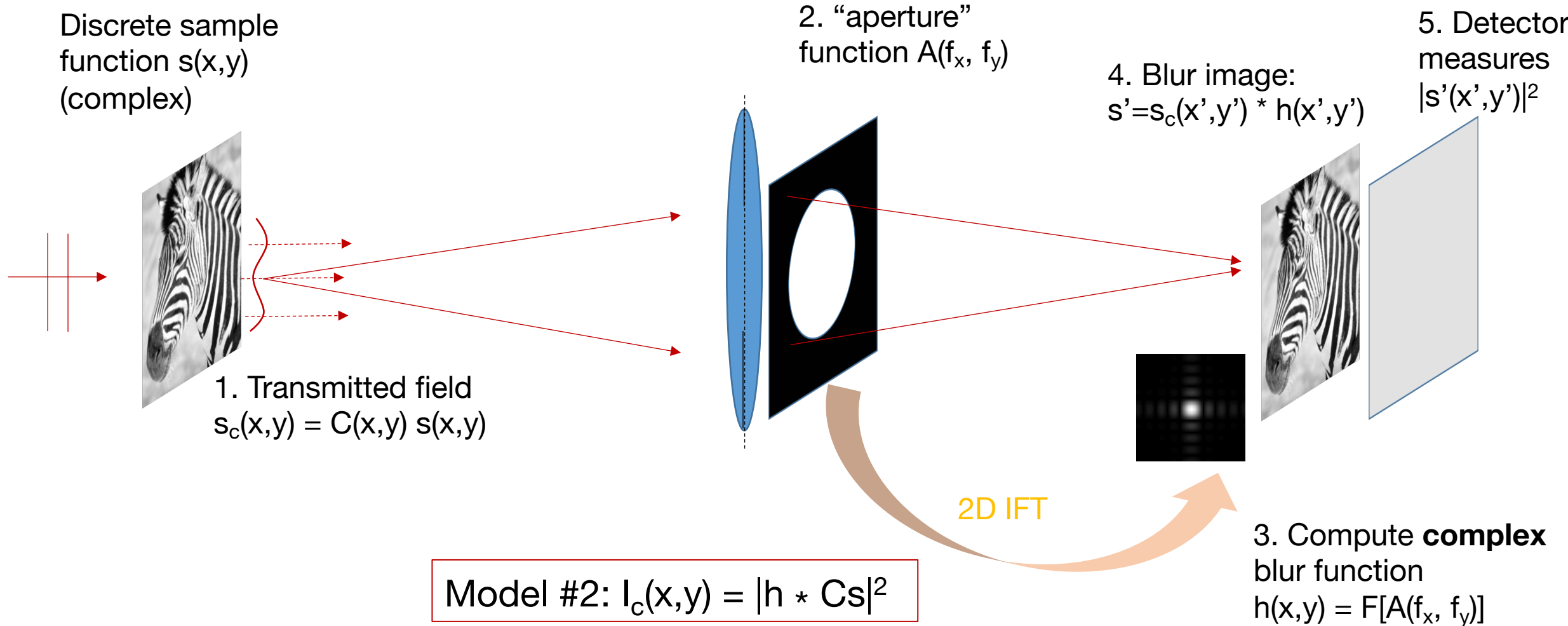
Model of image formation for wave optics (coherent light):



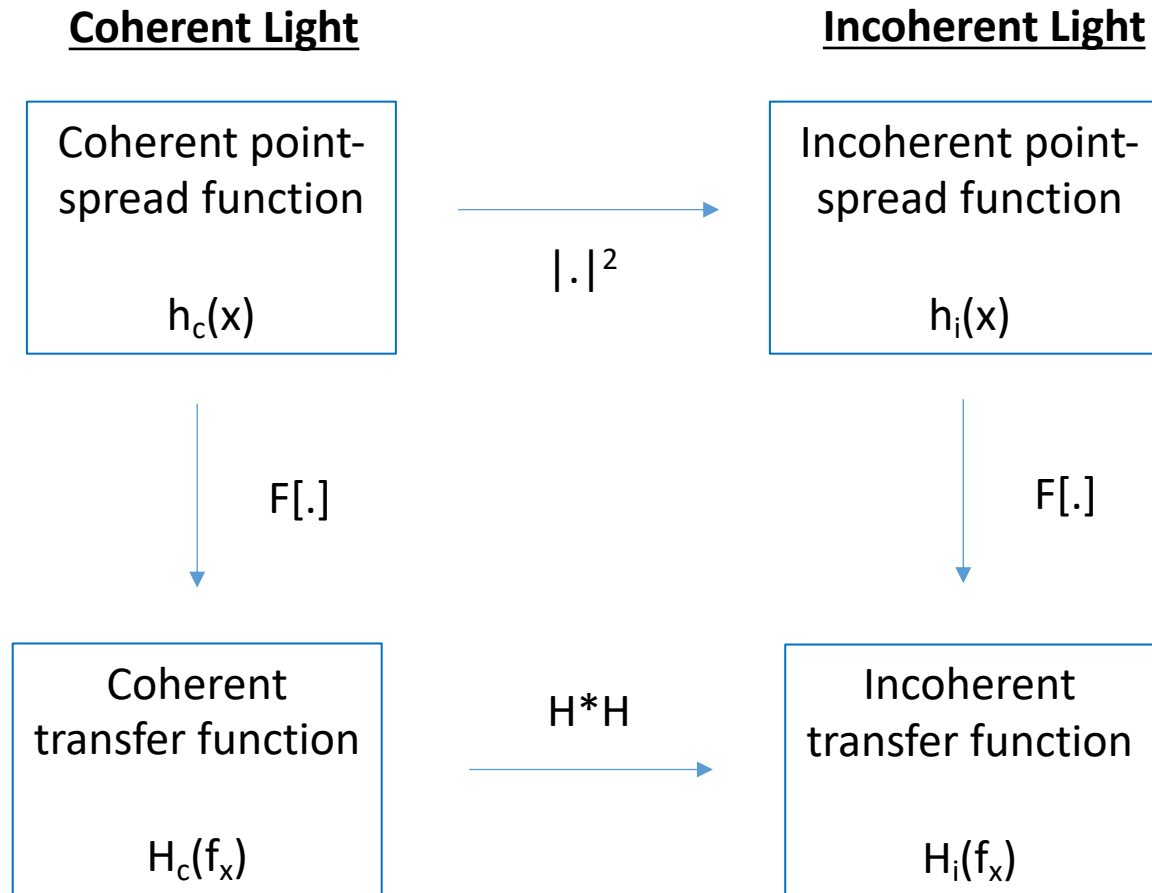
Model of image formation for wave optics (coherent light):



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



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

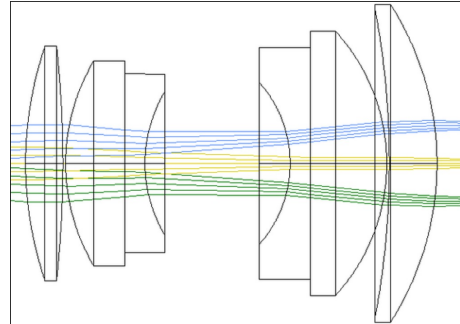
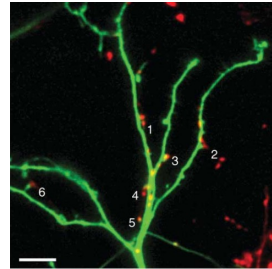


Incoherent PSF = Coherent PSF squared:

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

Summary of two models for image formation

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

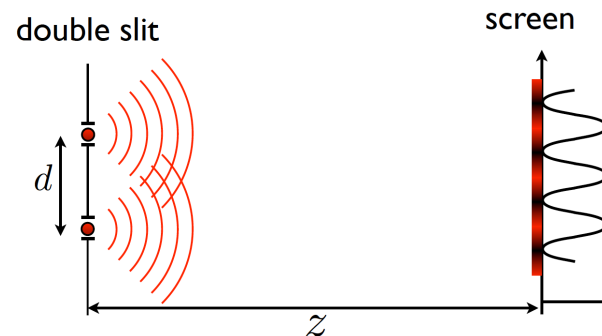
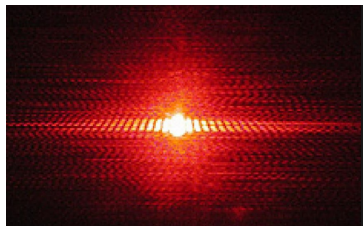


- Real, non-negative

$$I_s = h_i * B S_0$$

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

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



- Complex-valued

$$I_c = |h_c * C S_c|^2$$

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

Coherent image formation equation as CNN operations

$$I_C = D |h_c * C S_C|^2$$

Step 1: Multiply with weights

Step 2: Convolution

Step 3: Absolute value square (non-linearity)

Step 4: Down-sampling by detector

CNN layer

(Step 1: Normalization)

Step 2: Convolution

Step 3: Non-linearity

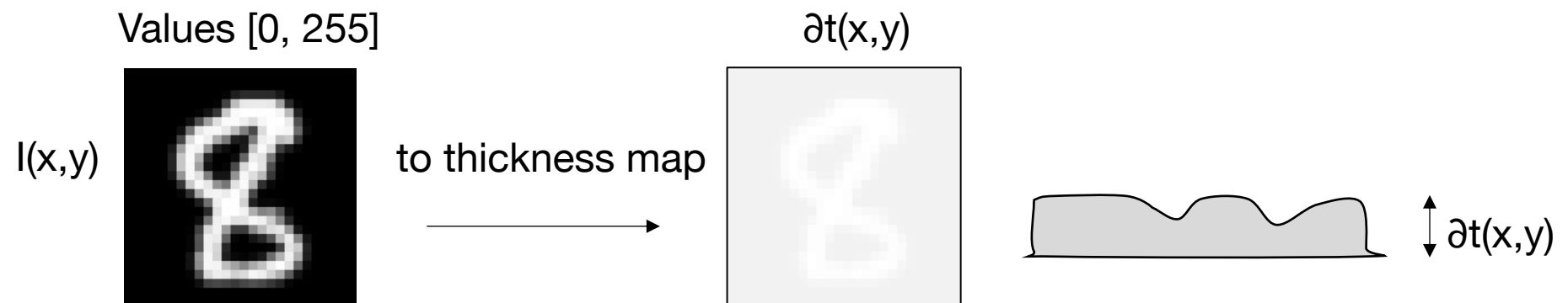
Step 4: Down-sampling by max pooling

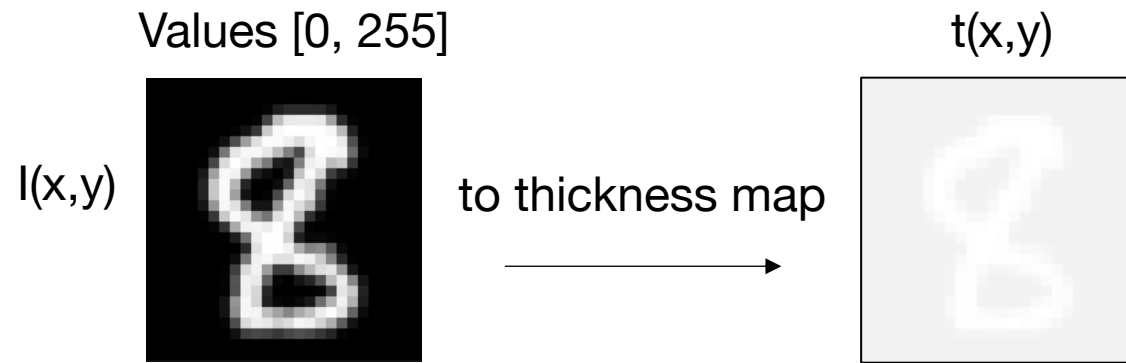
Example #1: Optimizing coherent illumination pattern for improved classification

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.

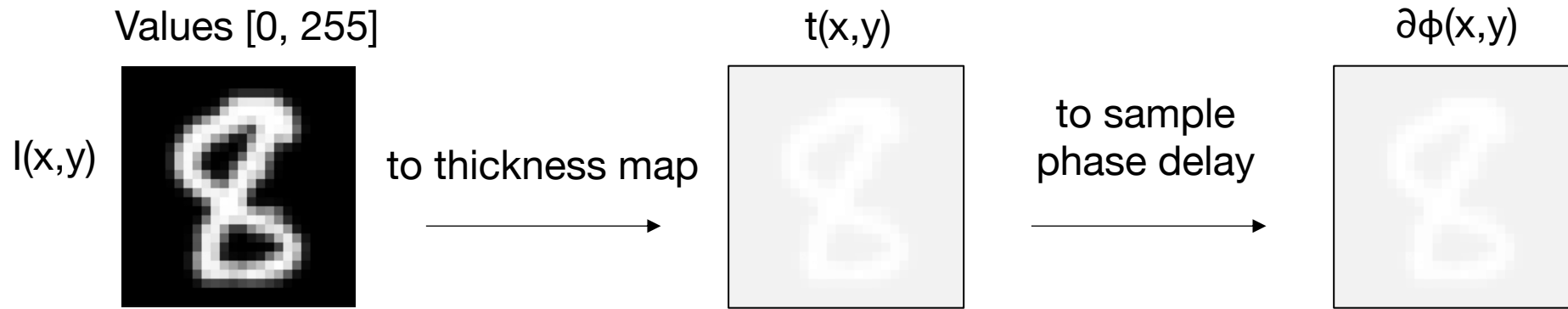
Question: What type of illumination should you use to maximize the classification accuracy of the numbers on the check?

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

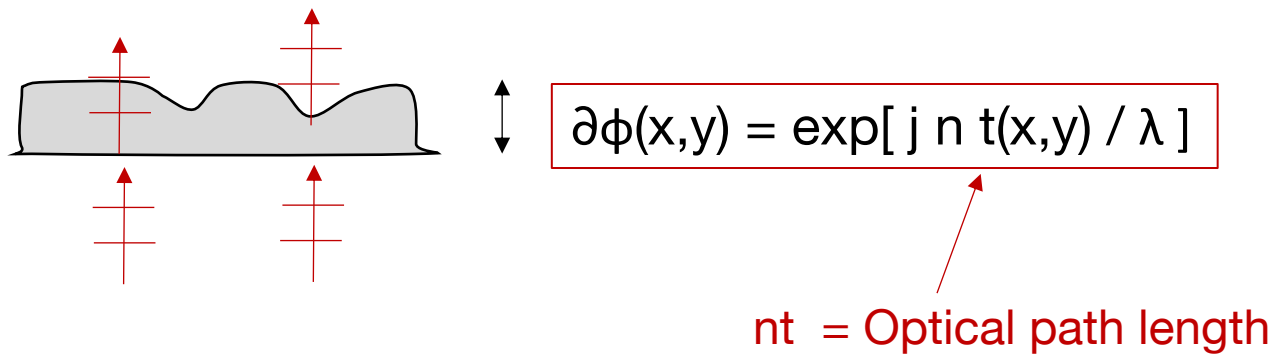


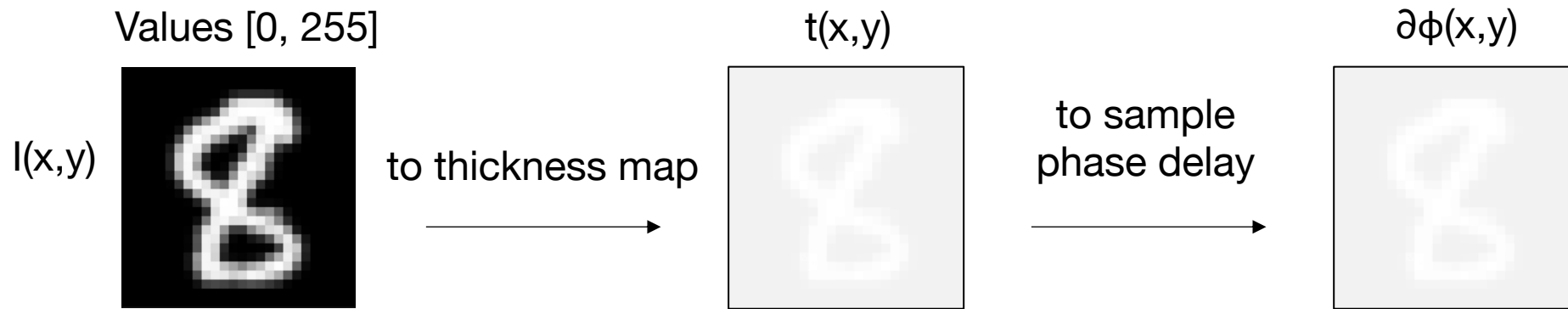


1. Normalize intensity map to 1
2. Define thickness map at some reasonable amount (100 μm max change)



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:





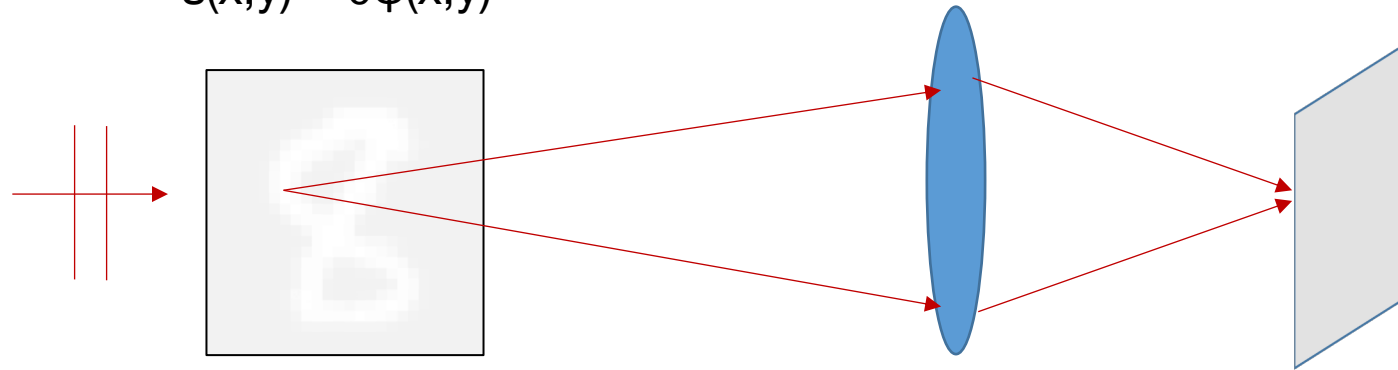
1. Normalize intensity map to 1
2. Define thickness map at some reasonable amount (100 μm max change)
3. Convert thickness map into optical phase delay:

```
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

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

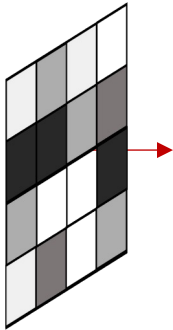
$$s(x,y) = \partial\phi(x,y)$$



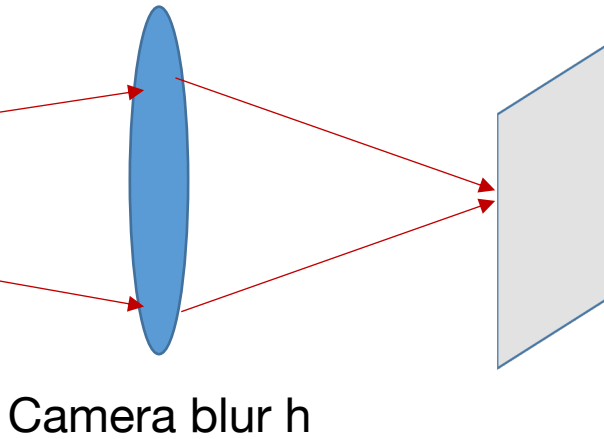
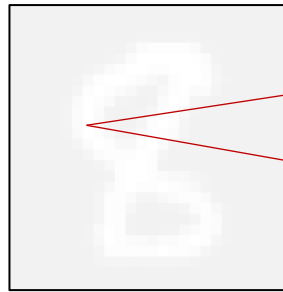
Example #1: Optimizing coherent illumination pattern for improved classification

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

Unknown
Illumination $c(x,y)$
(complex weight variable)



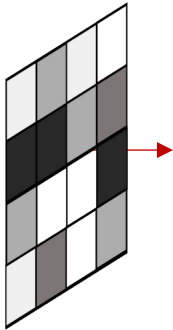
$$s(x,y) = \partial\phi(x,y)$$



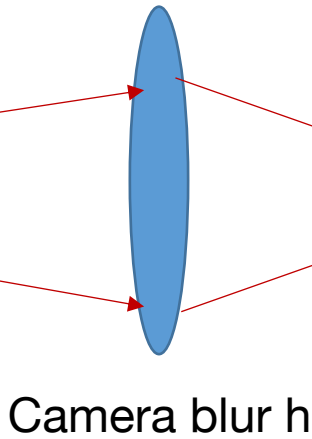
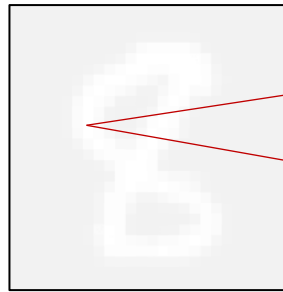
Example #1: Optimizing coherent illumination pattern for improved classification

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

Unknown
Illumination $c(x,y)$
(complex weight variable)

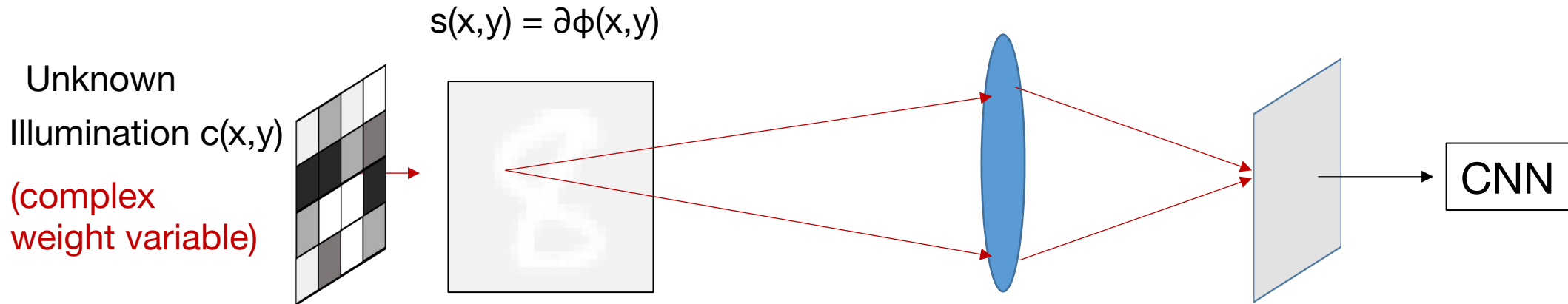


$$s(x,y) = \partial\phi(x,y)$$



Example #1: Optimizing coherent illumination pattern for improved classification

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



```

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

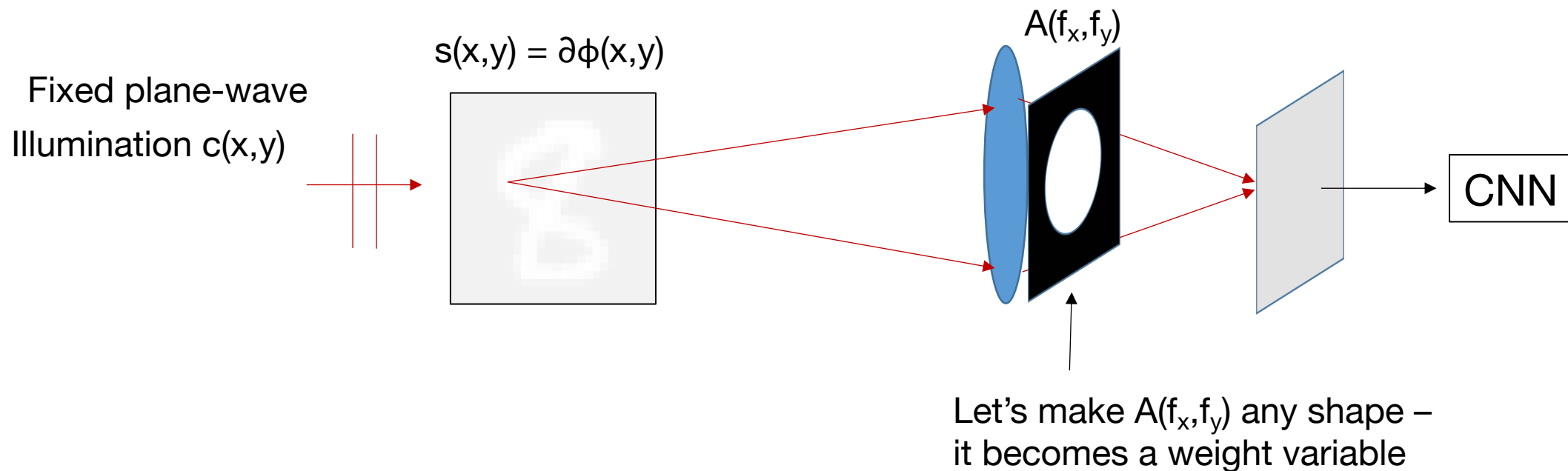
Example future situation: Hacking has brought online banking to a halt. We now rely on a special form of physical check that is made of visibly transparent plastic. To write the amount in, you press down with a pen-like instrument, and then the check is read out by shining a particular pattern of laser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

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

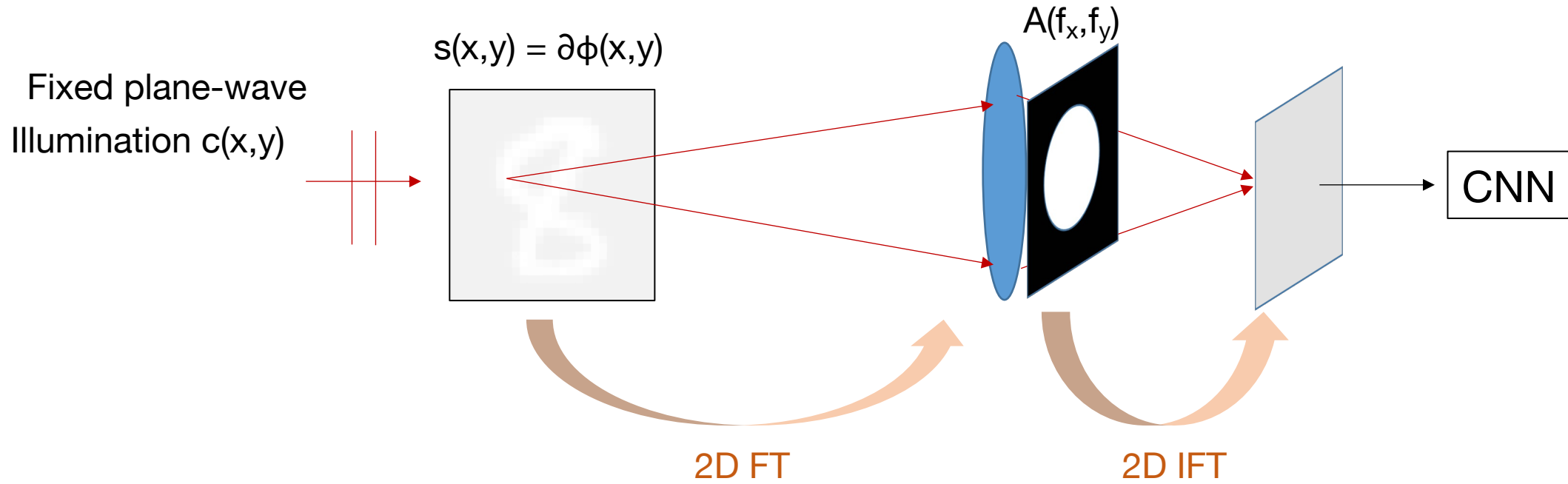
Example #2: Optimizing aperture shape for improved digit classification

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.

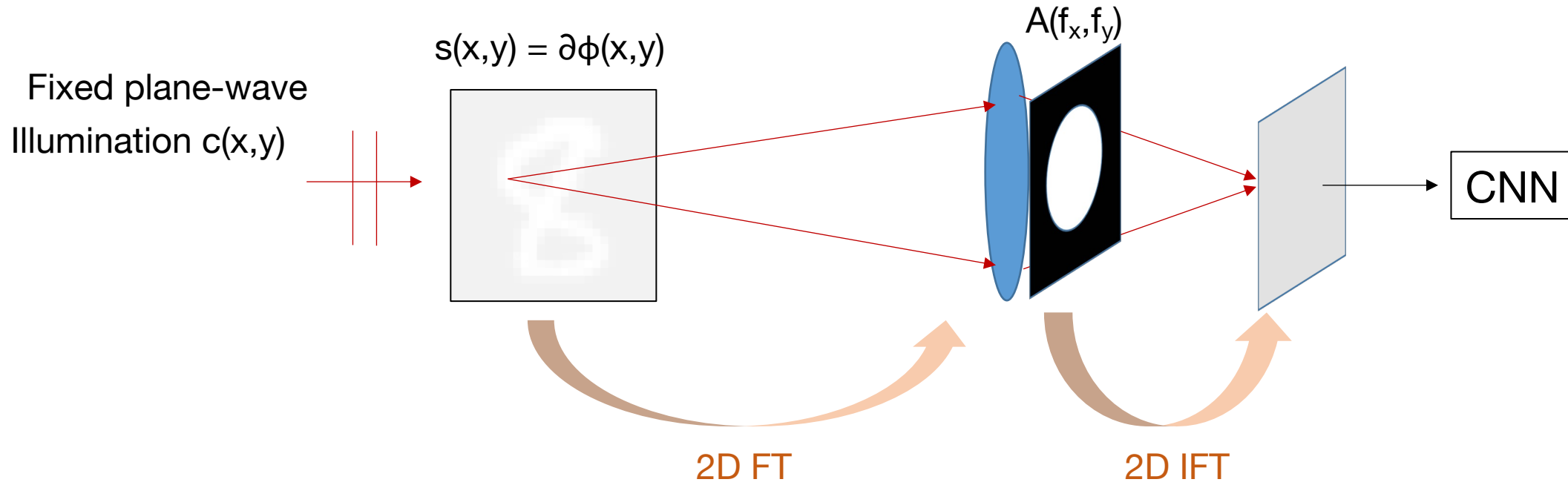
Question #2: What type of aperture shape should you use to maximize classification accuracy?



Example #2: Optimizing aperture shape for improved digit classification



Example #2: Optimizing aperture shape for improved digit classification



```

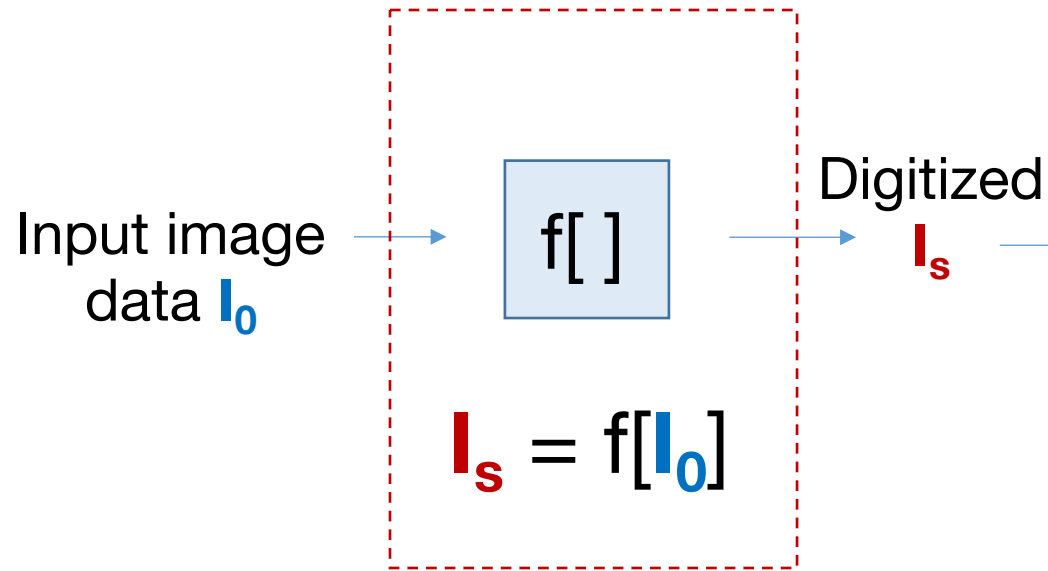
mnist_phase_delay = tf.reshape(mnist_phase_delay, [-1, image_size, image_size])
C0 = np.ones(image_size, image_size)
C0 = tf.constant(C0)
x_C_complex = tf.mul(mnist_phase_delay, C0)
fx_C_complex = tf.fft2d(x_C_complex)
ap_filter = tf.Variable([image_size, image_size])
filtered_x_C = tf.mul(fx_C_complex,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

```

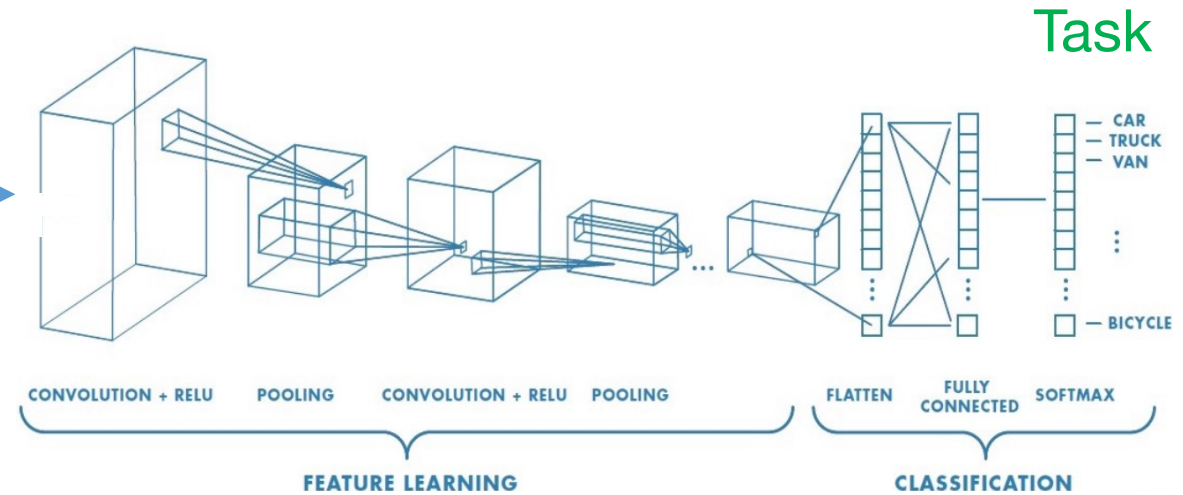
Remaining questions to address about physical layers:

- 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?

Physical Layers



Digital Layers



Some Examples:

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

Physical Layers

Input image data I_0

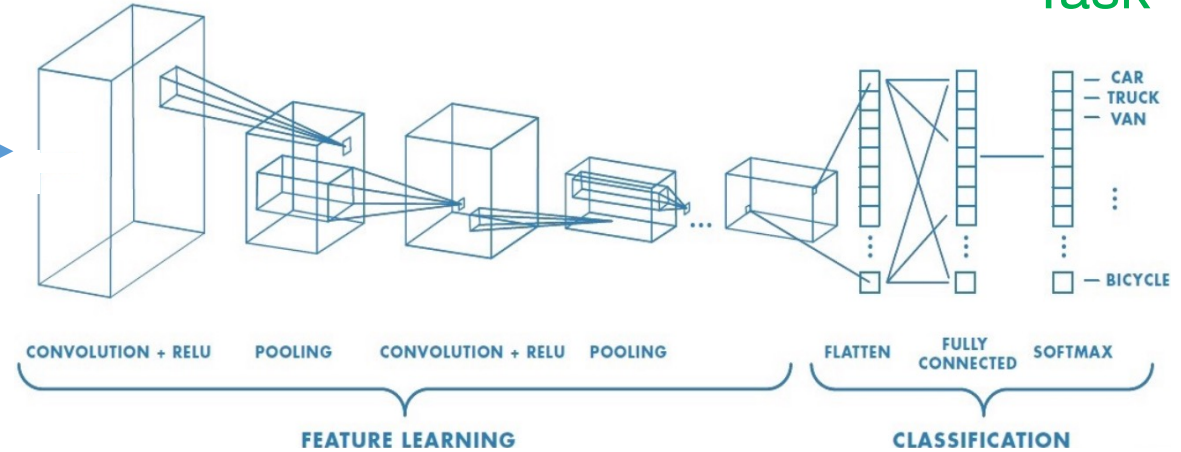
$f[]$

$$I_s = f[I_0]$$

Digitized I_s

Digital Layers

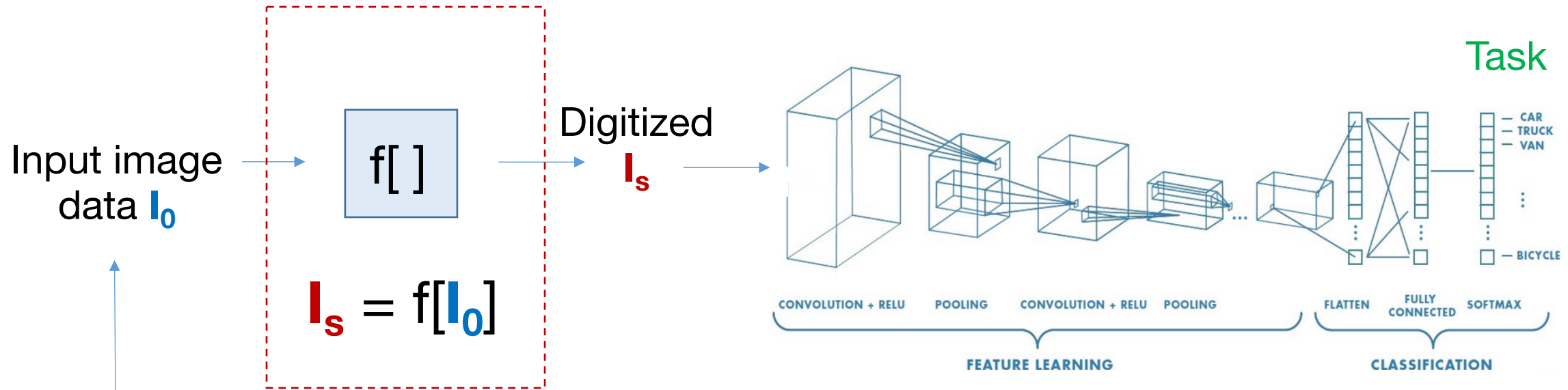
Task



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

Physical Layers

Digital Layers



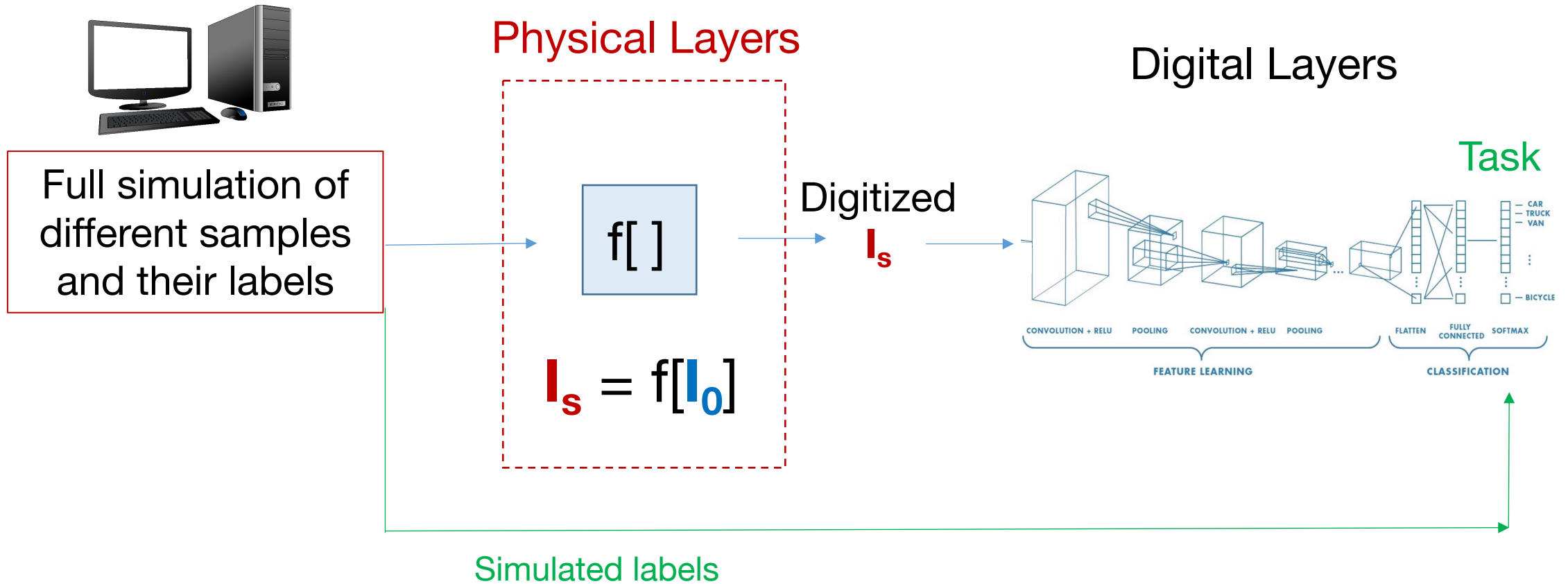
Q: Where and how should I implement my physical layer?

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

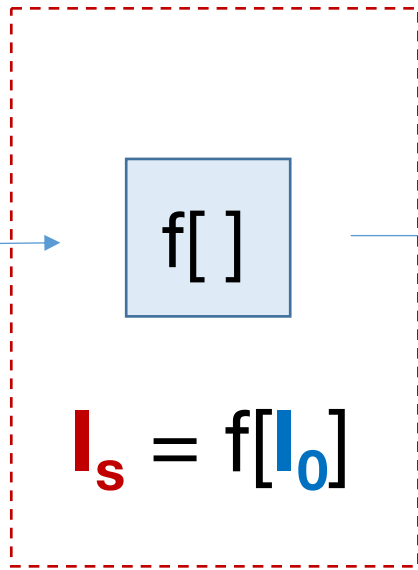


Full simulation of different samples and their labels

Examples:

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

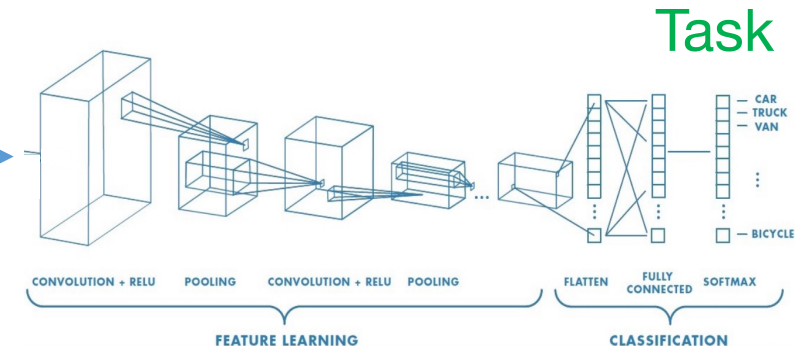
Physical Layers



Digitized

I_s

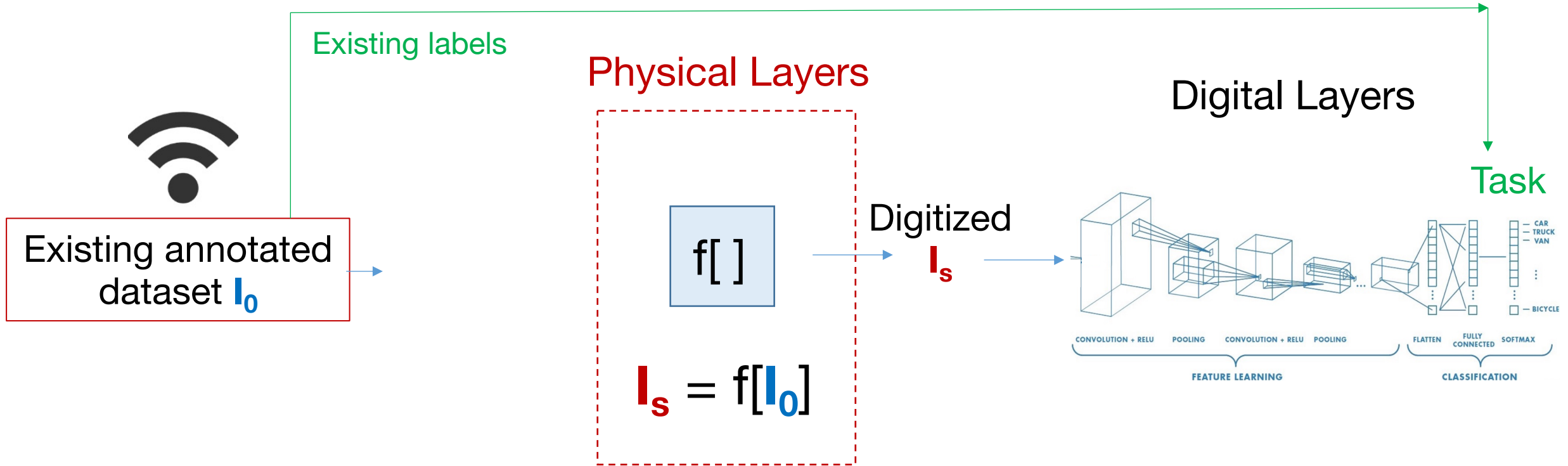
Digital Layers



Simulated labels

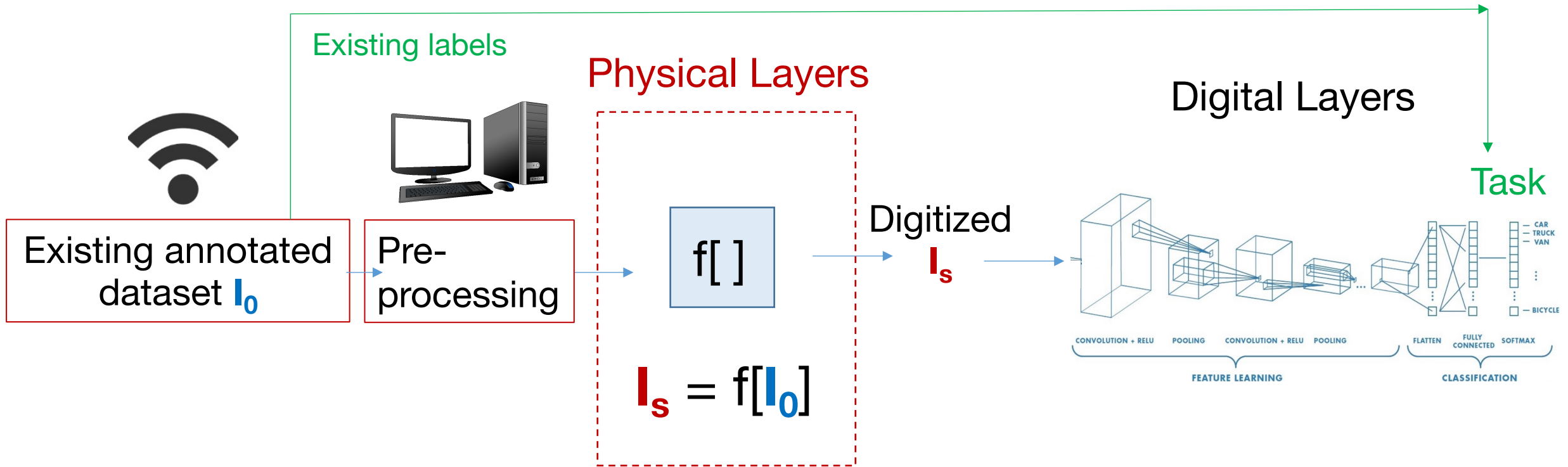
Situation #1: Fully simulated physical layers

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



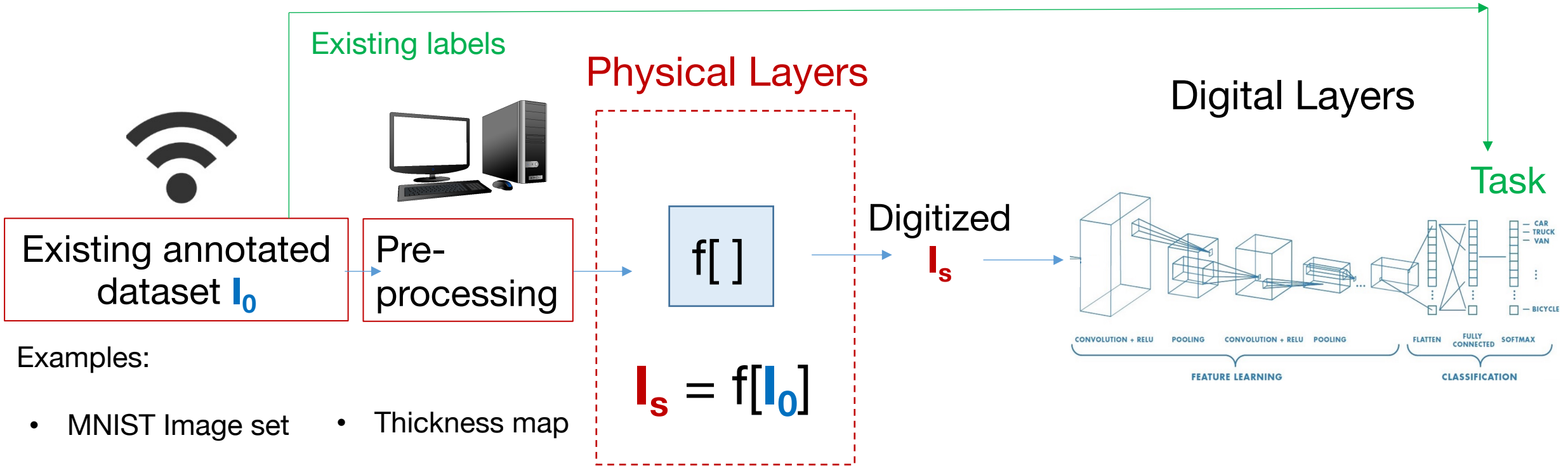
Situation #1: Fully simulated physical layers

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



 TensorFlow

ML Optimization

 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

Pros: Utilize old code, easier to archive, troubleshoot

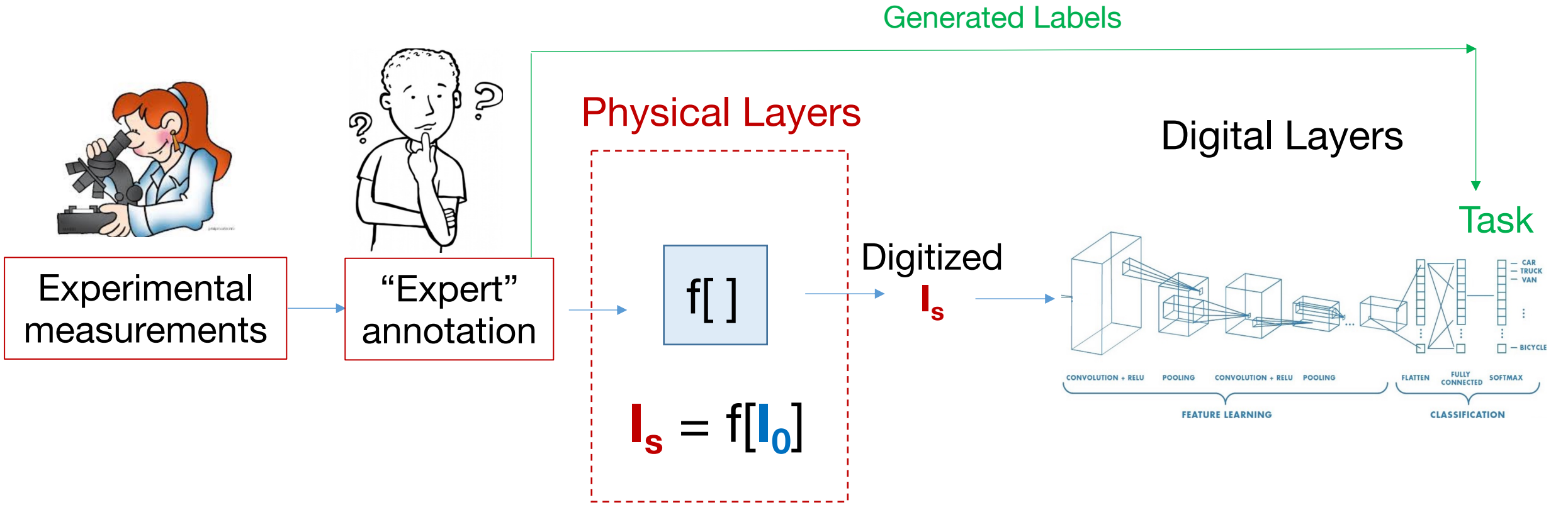
Cons: Large datasets are slow to load, hard to fit in GPU memory, code in 2 places



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

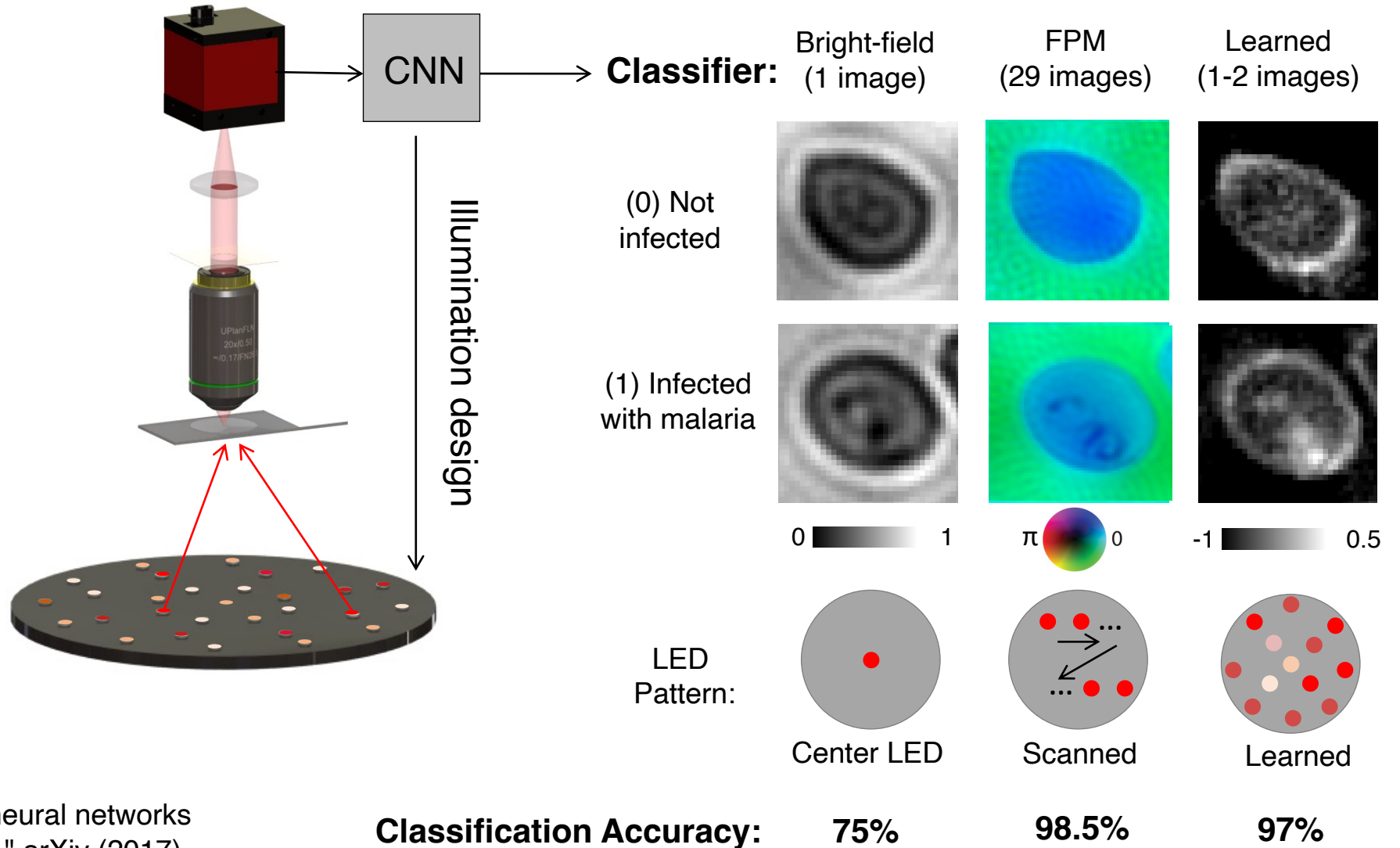
Cons: Harder to bug-check /compare to prior work if closely integrated

Situation #2: Experimentally-driven physical layers



Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:



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

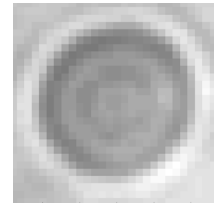
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:



Data set for physical layer optimization:

Turn on LED 1, capture image 1:



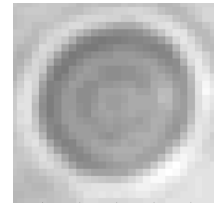
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:

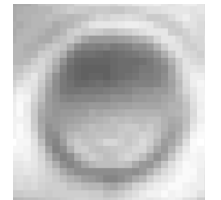


Data set for physical layer optimization:

Turn on LED 1, capture image 1:



Turn on LED 1, capture image 2:



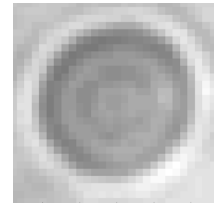
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:

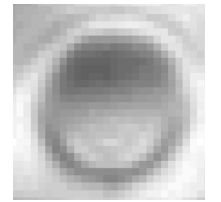


Data set for physical layer optimization:

Turn on LED 1, capture image 1:

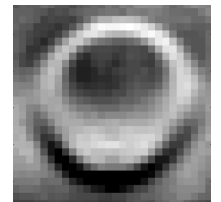


Turn on LED 1, capture image 2:



⋮

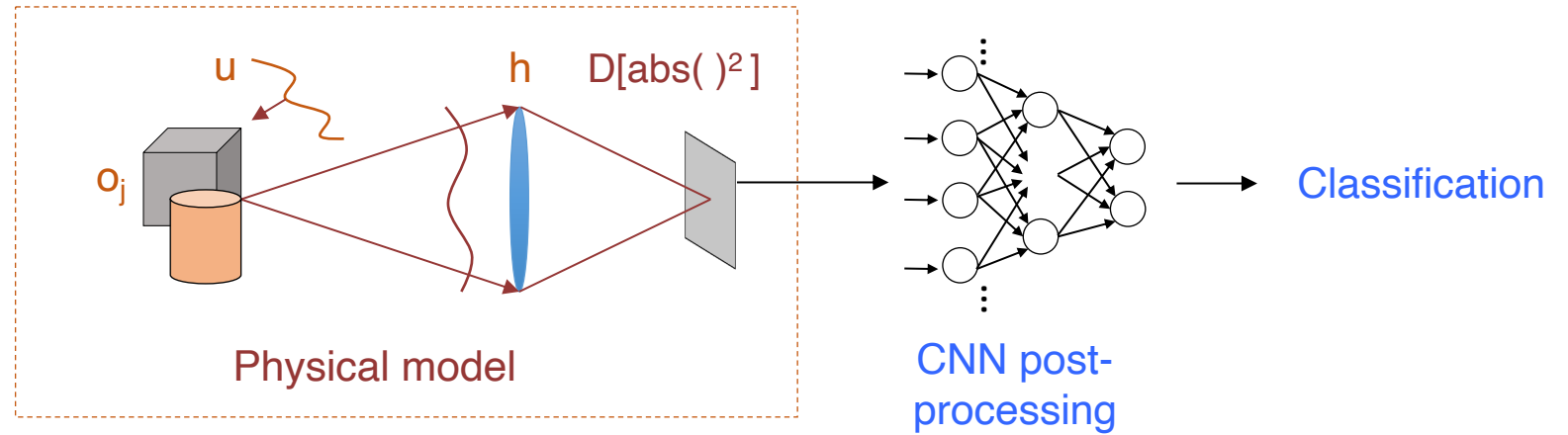
Turn on LED 32, capture image 32:



Save all 32 images (96 with 3 colors)

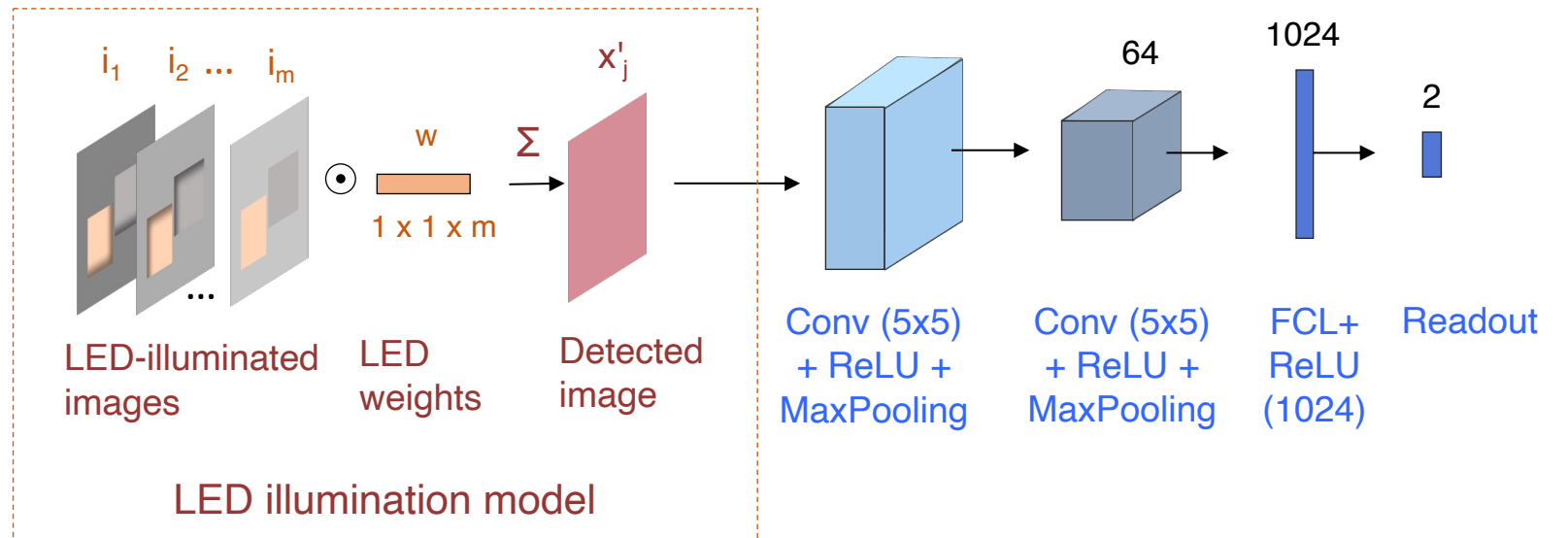
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:

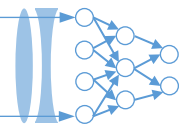


Physical layer:

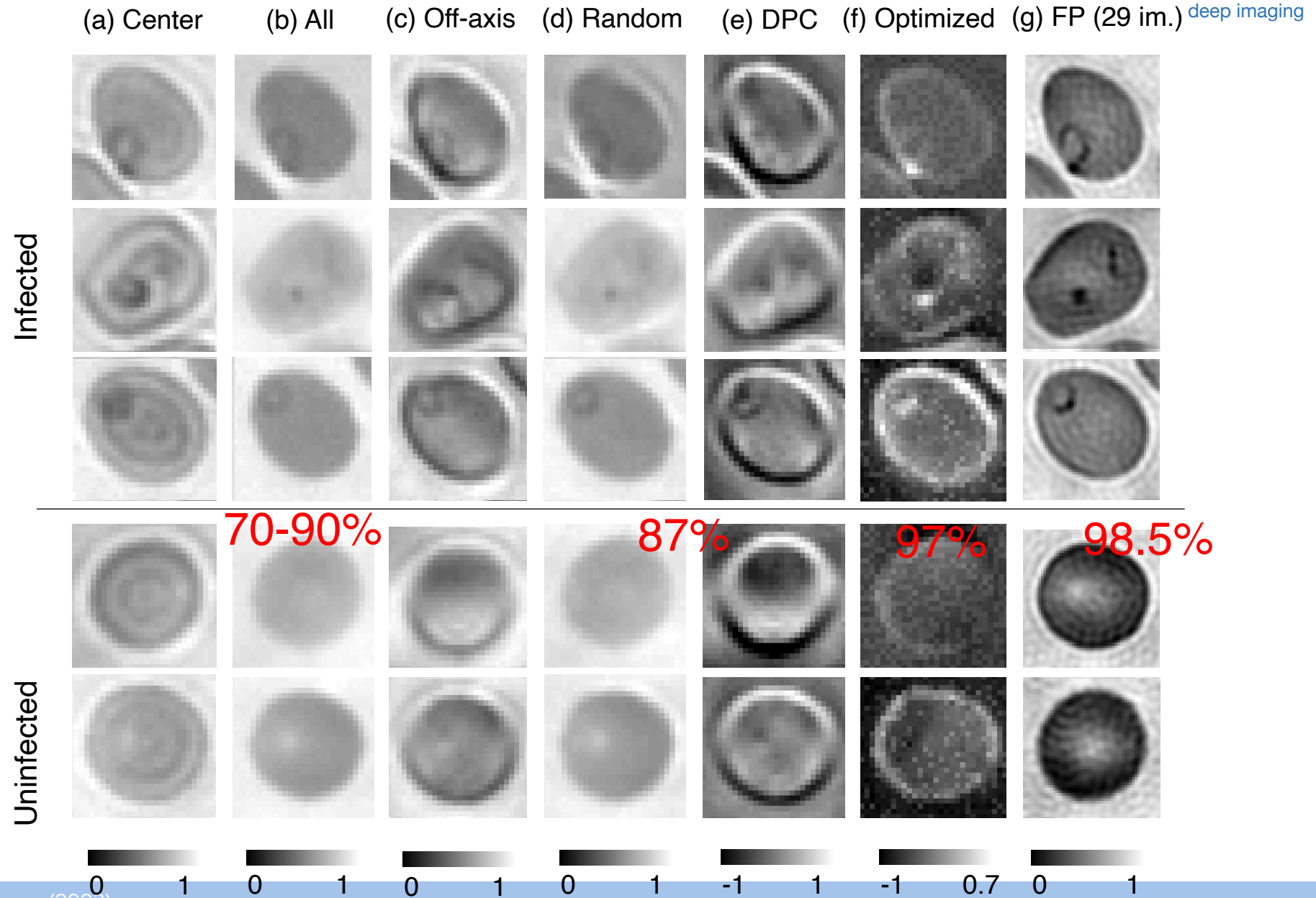
$$I_s = \sum w_j I_j$$



Situation #2: Experimentally-driven physical layers

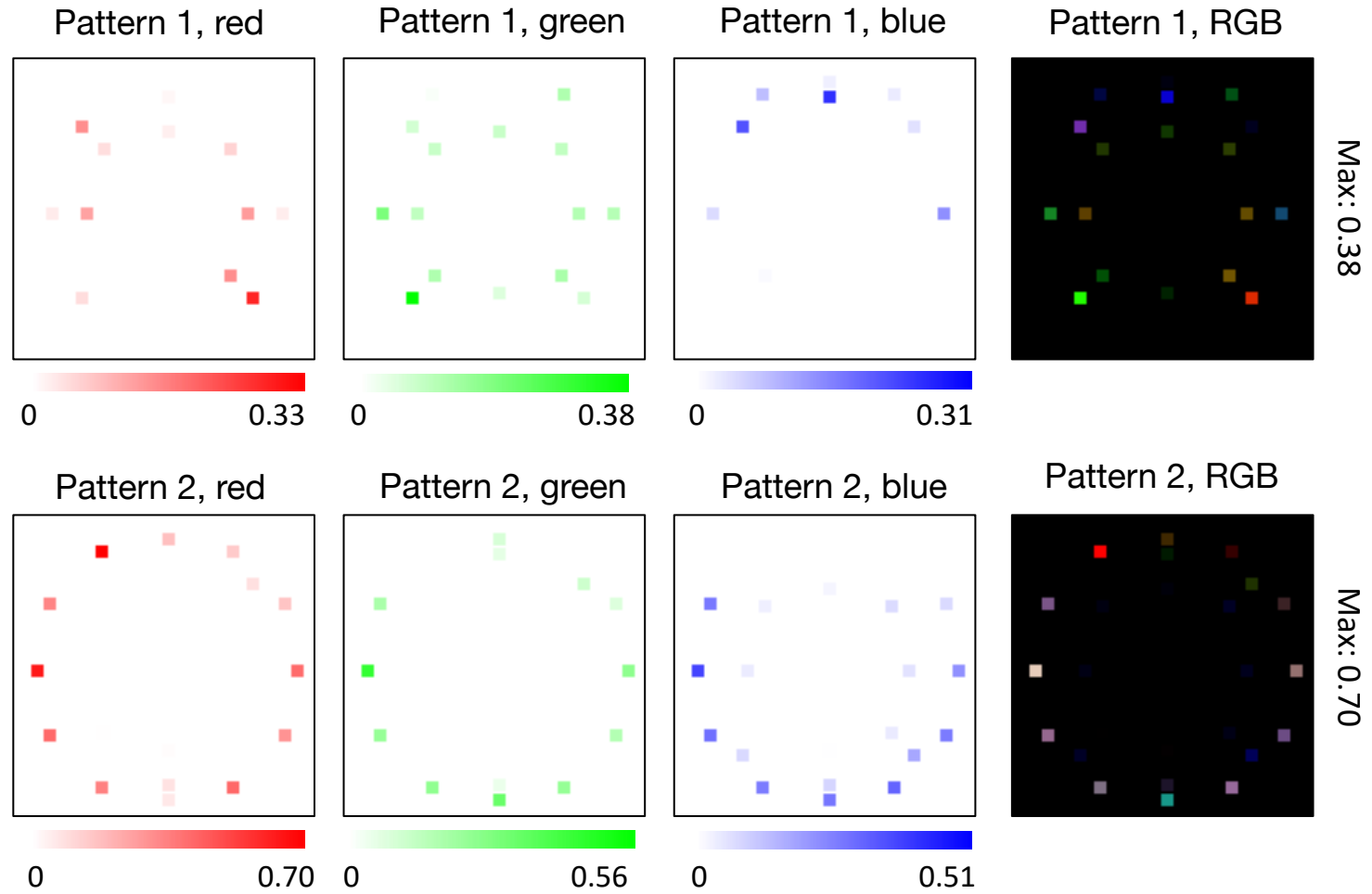


Example: CNN-Optimized illumination for classification of malaria:

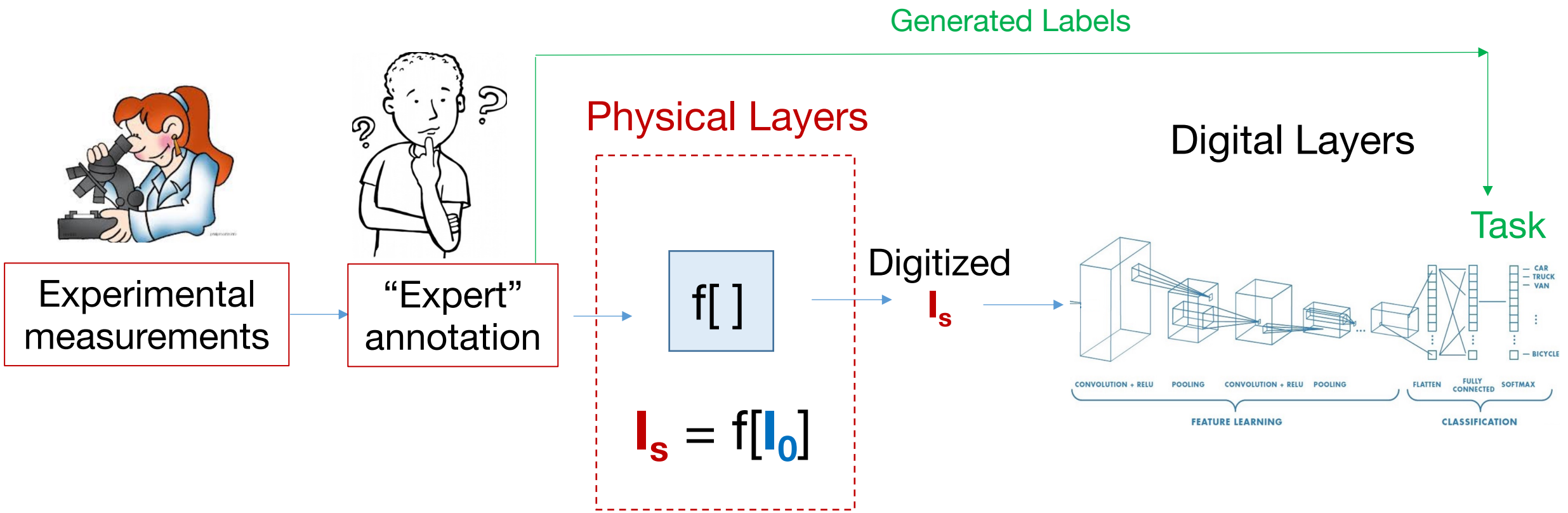


Situation #2: Experimentally-driven physical layers

Optimized color LED patterns to classify malaria



Situation #2: Experimentally-driven physical layers



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

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?

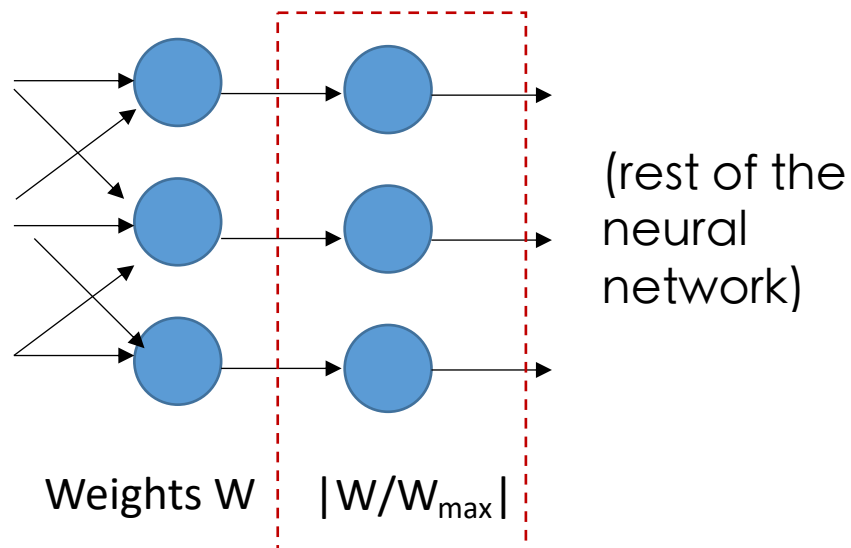
Example: Constrain weights to be non-negative values less than one

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)

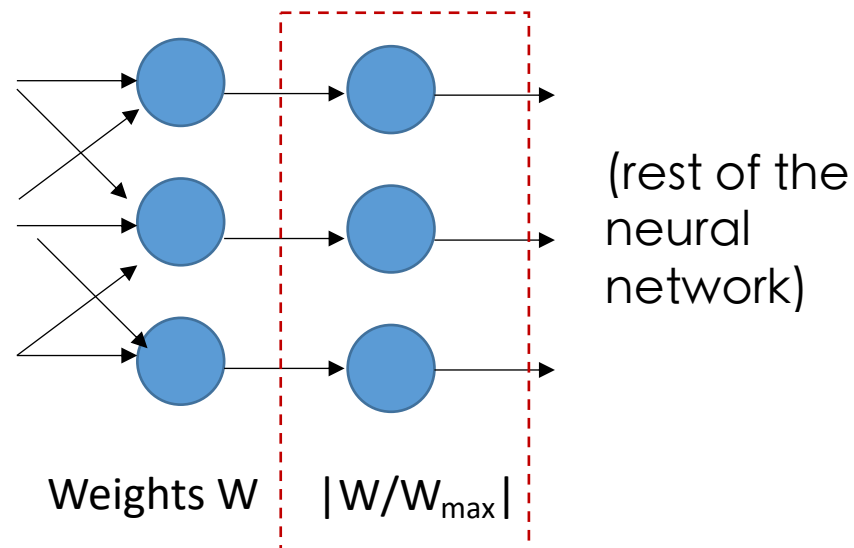


How can I add some constraints to my physical weights?

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

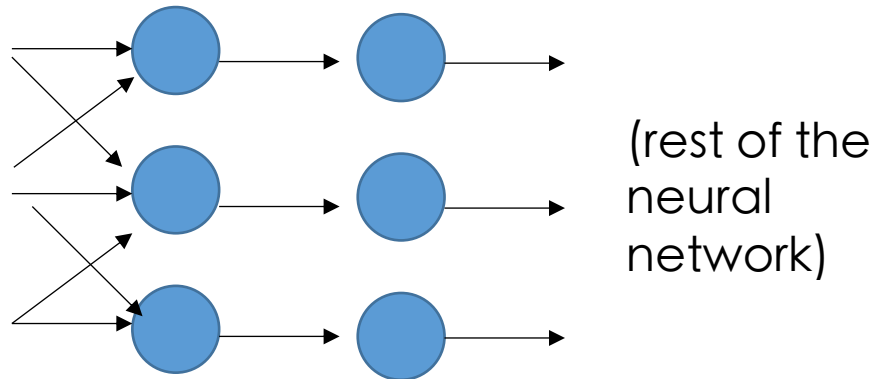
Solution: Perform annealing with a temperature parameter

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



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

Increase α with iteration number

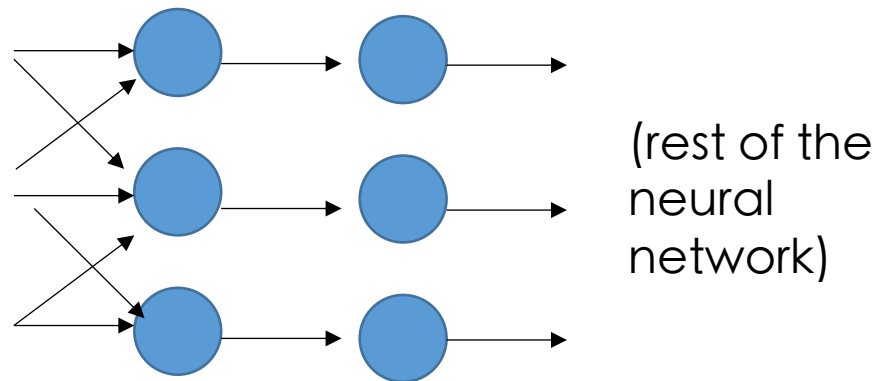
$$\text{Soft-max}(x) = \exp(-x) / \sum \exp(-x)$$

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

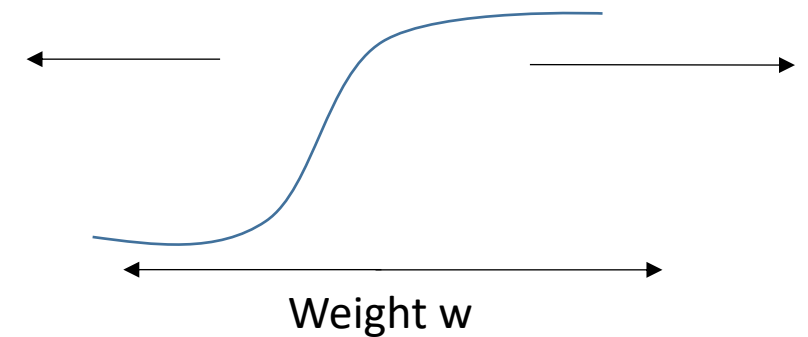
Solution: Perform annealing with a temperature parameter



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

Increase α with iteration number

Drive w to be large, so $\text{softmax}(w) \rightarrow 0$ or 1



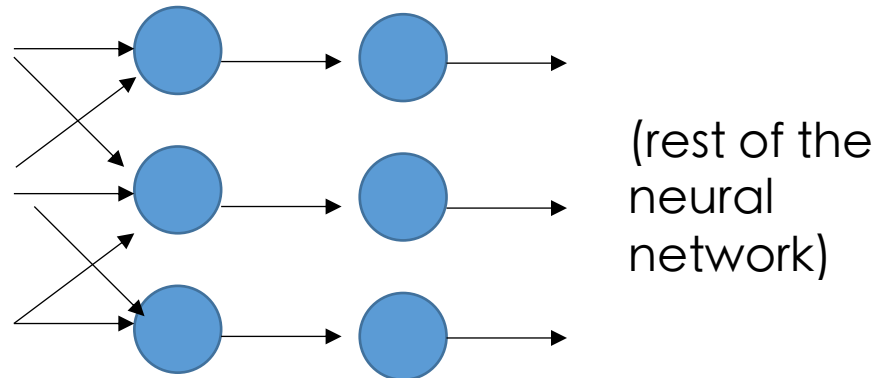
$\text{Soft-max}(x) = \frac{\exp(-x)}{\sum \exp(-x)}$

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



Pros:

- Works pretty well
- Flexibly address convergence issues

Cons:

- A bit sensitive

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

Increase α with iteration number

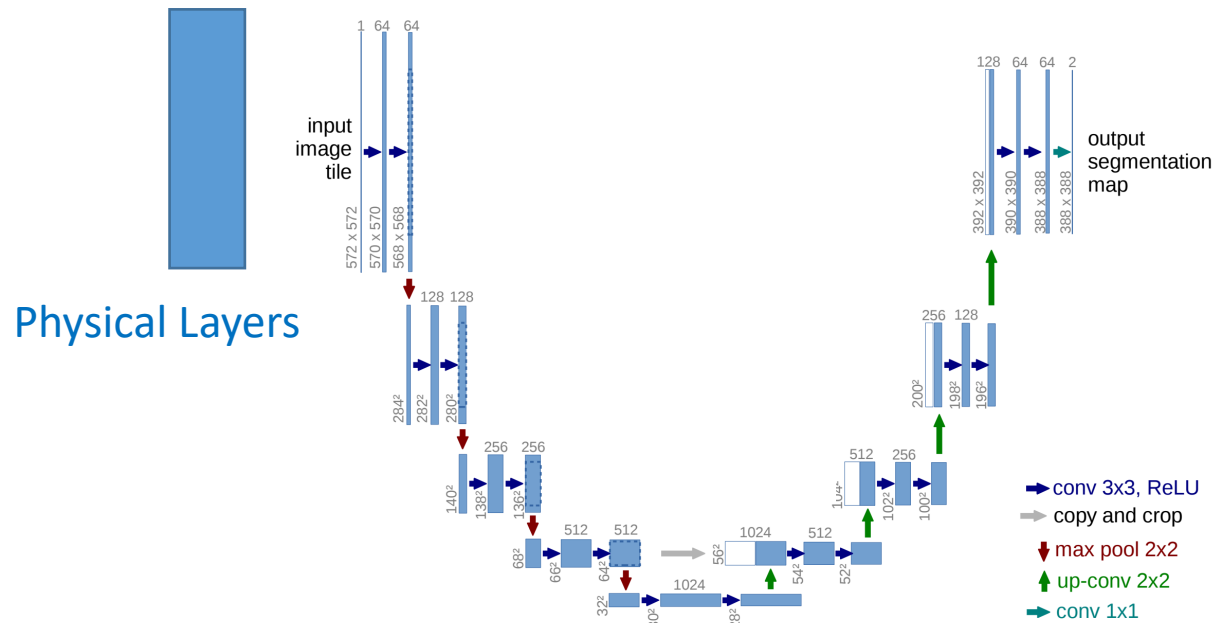
$$\text{Soft-max}(x) = \exp(-x) / \sum \exp(-x)$$

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

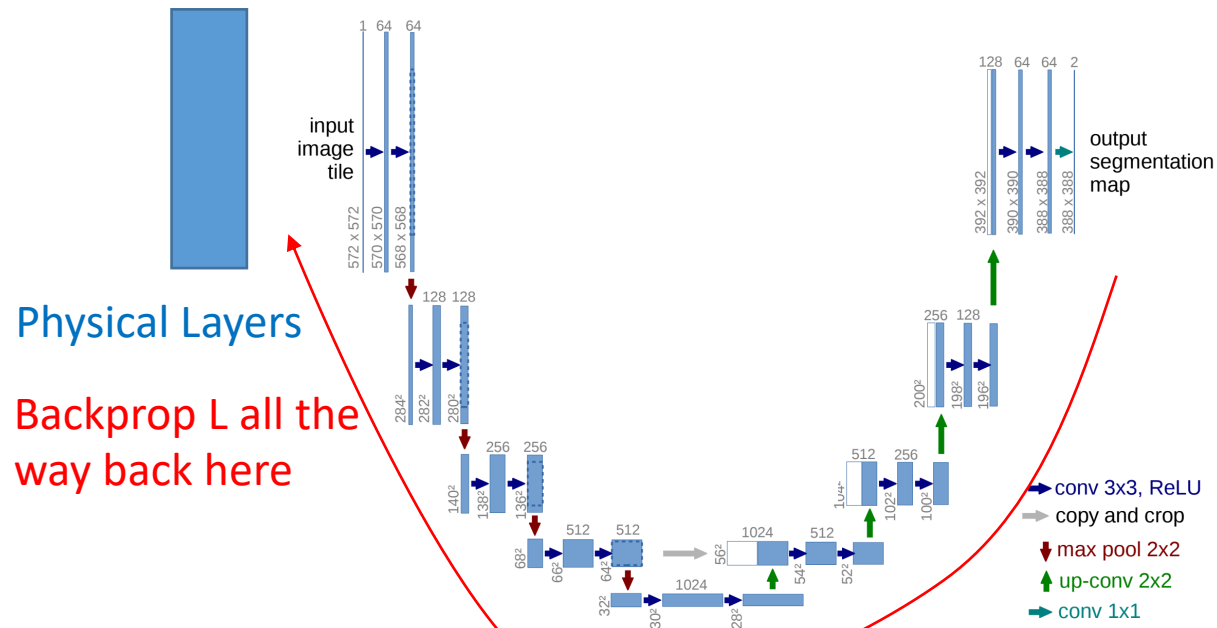
What are some common issues and pitfalls with physical layers?

- 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



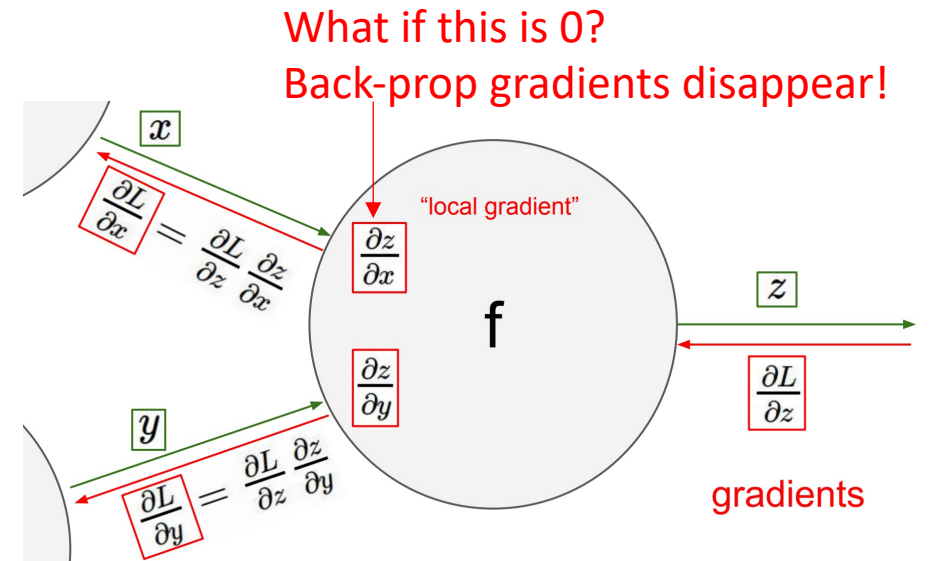
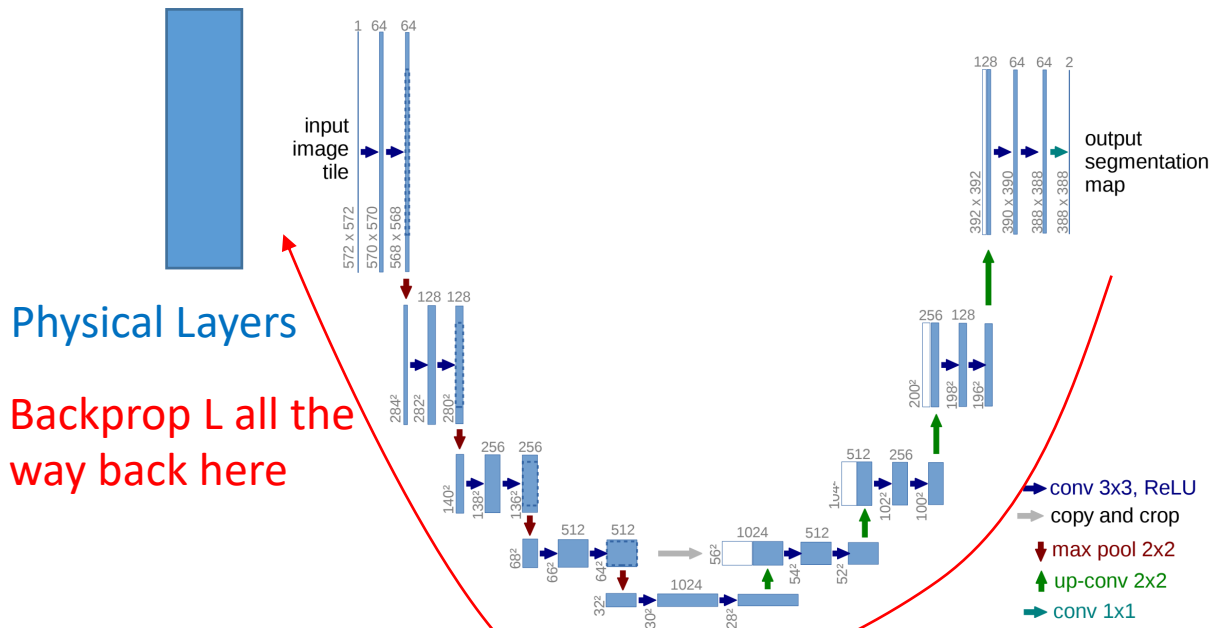
What are some common issues and pitfalls with physical layers?

- 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?

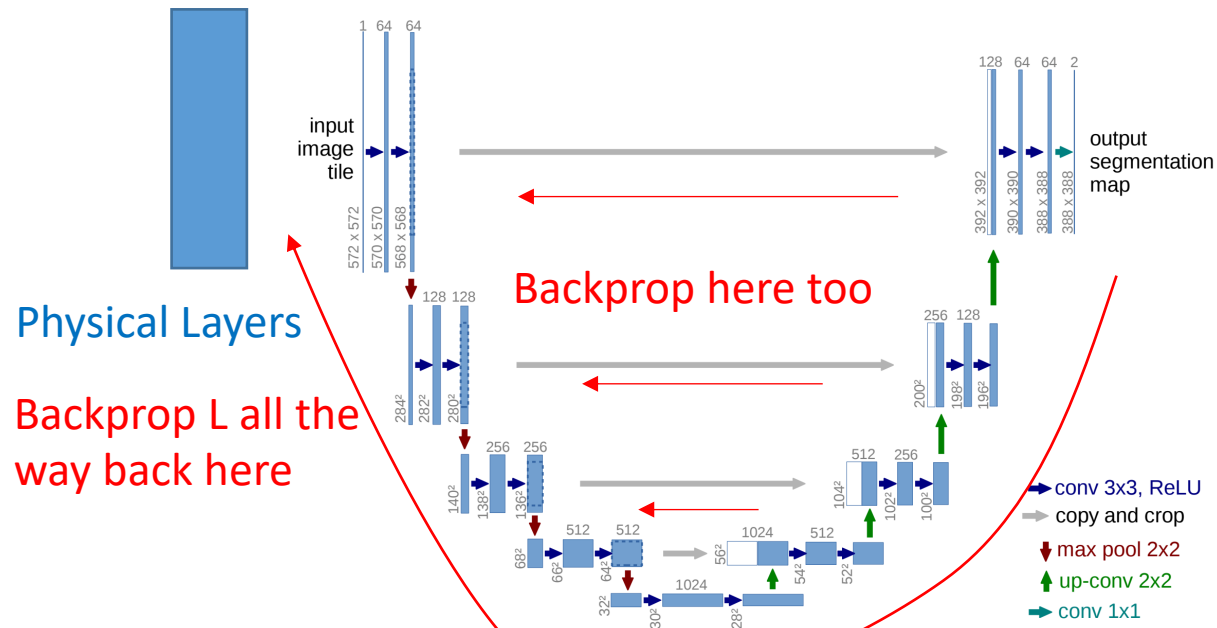
- Most common issue – you have a bug in your CNN!
 - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients



From Stanford CS231n

What are some common issues and pitfalls with physical layers?

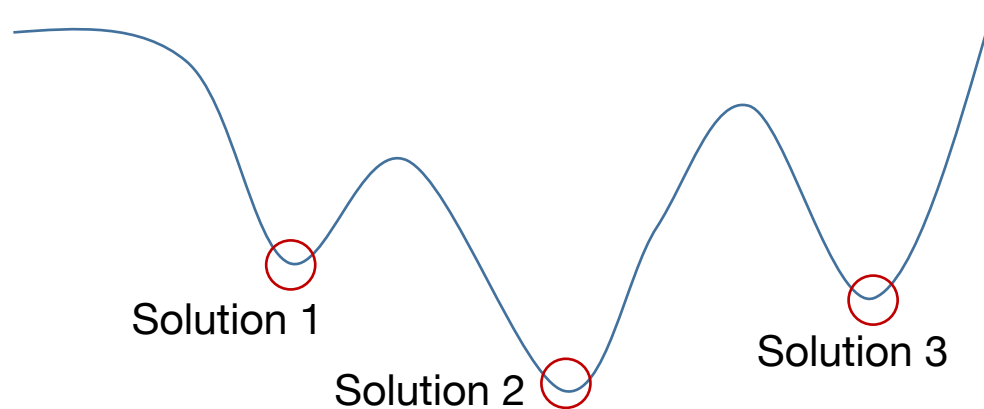
- Most common issue – you have a bug in your CNN!
 - Solution: “Disable “ physical layer (set to constant), and get network to work!
- Another common challenge - vanishing gradients



Solution: Introduce skipped connections

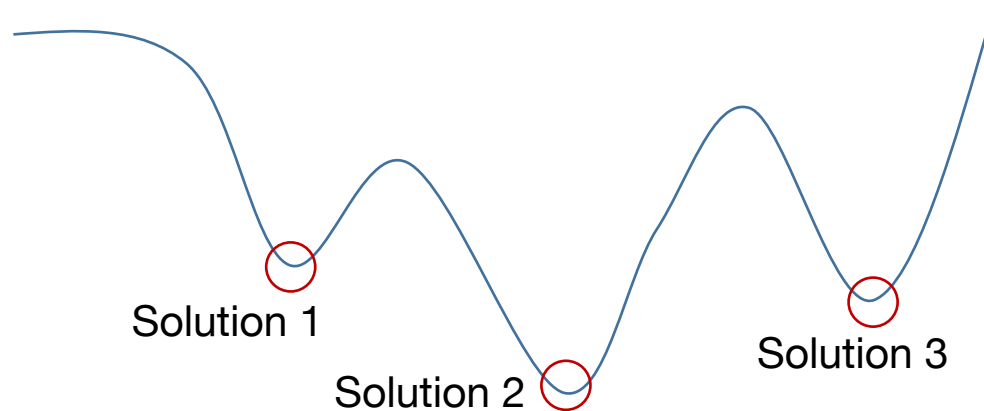
What are some common issues and pitfalls with physical layers?

- 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:

$$I_s = \sum w_j I_j + n$$

tf.keras.layers.GaussianNoise