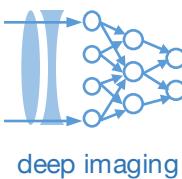


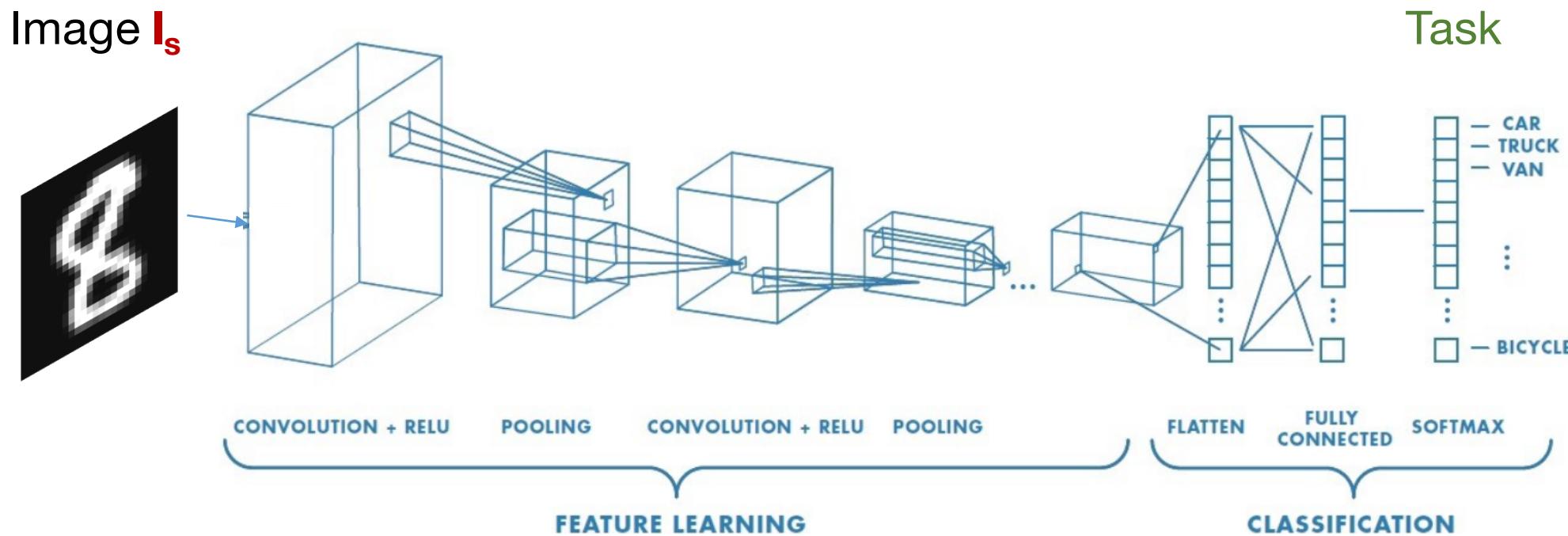
Lecture 16: Examples of Physical Layers for CNNs

Machine Learning and Imaging

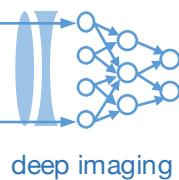
BME 590L
Roarke Horstmeyer



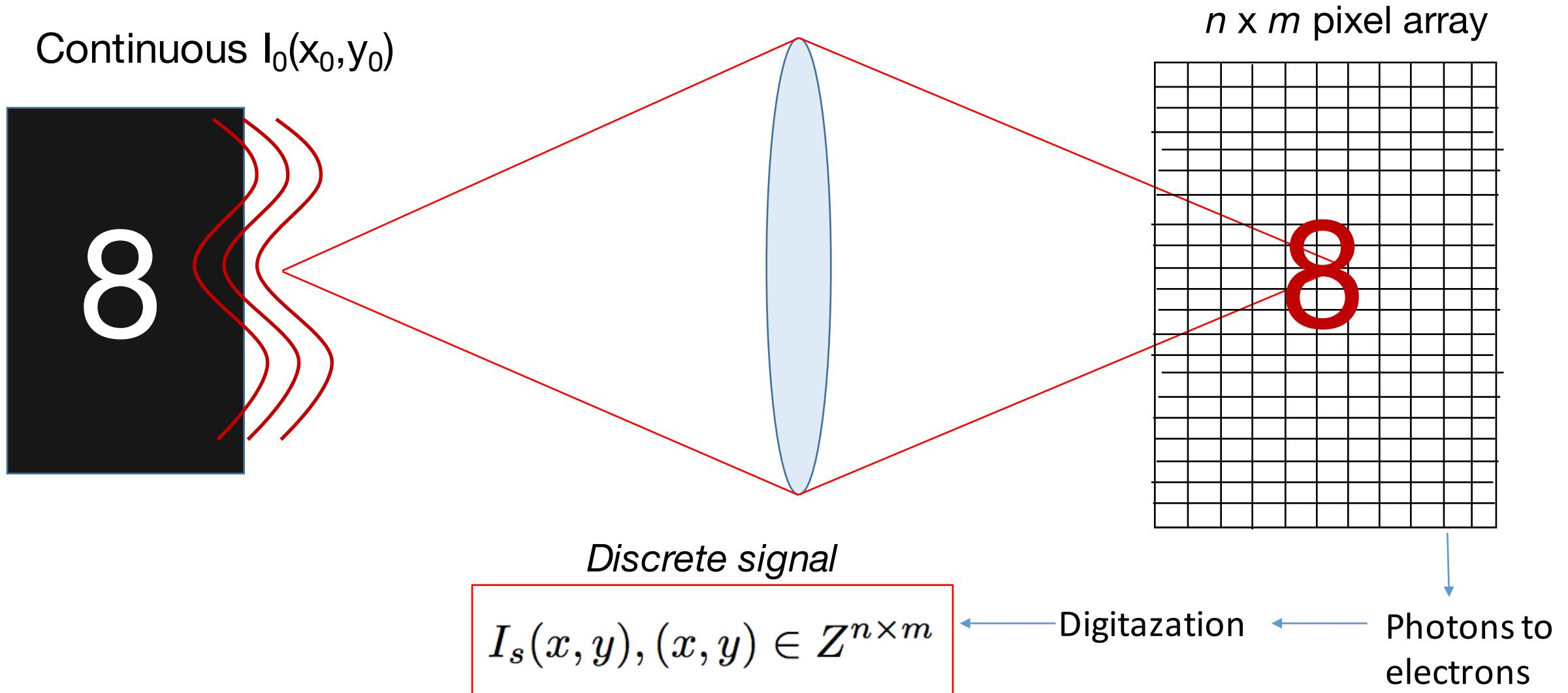
Bringing together physical and digital image representations



$$\text{Task} = \mathbf{W}_n \dots \text{ReLU}[\mathbf{W}_1 \text{ ReLU}[\mathbf{W}_0 \mathbf{I}_s] \dots]$$

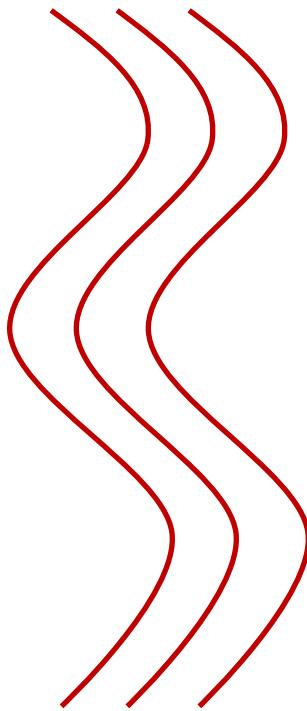


Simple model of image formation



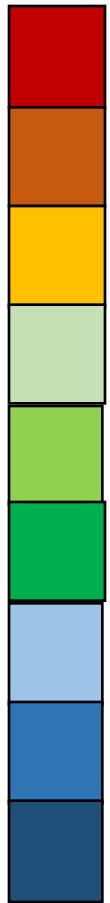
What does the Sampling Theorem mean for us?

Continuous functions

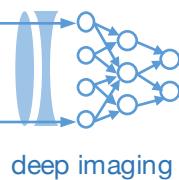


*conditions

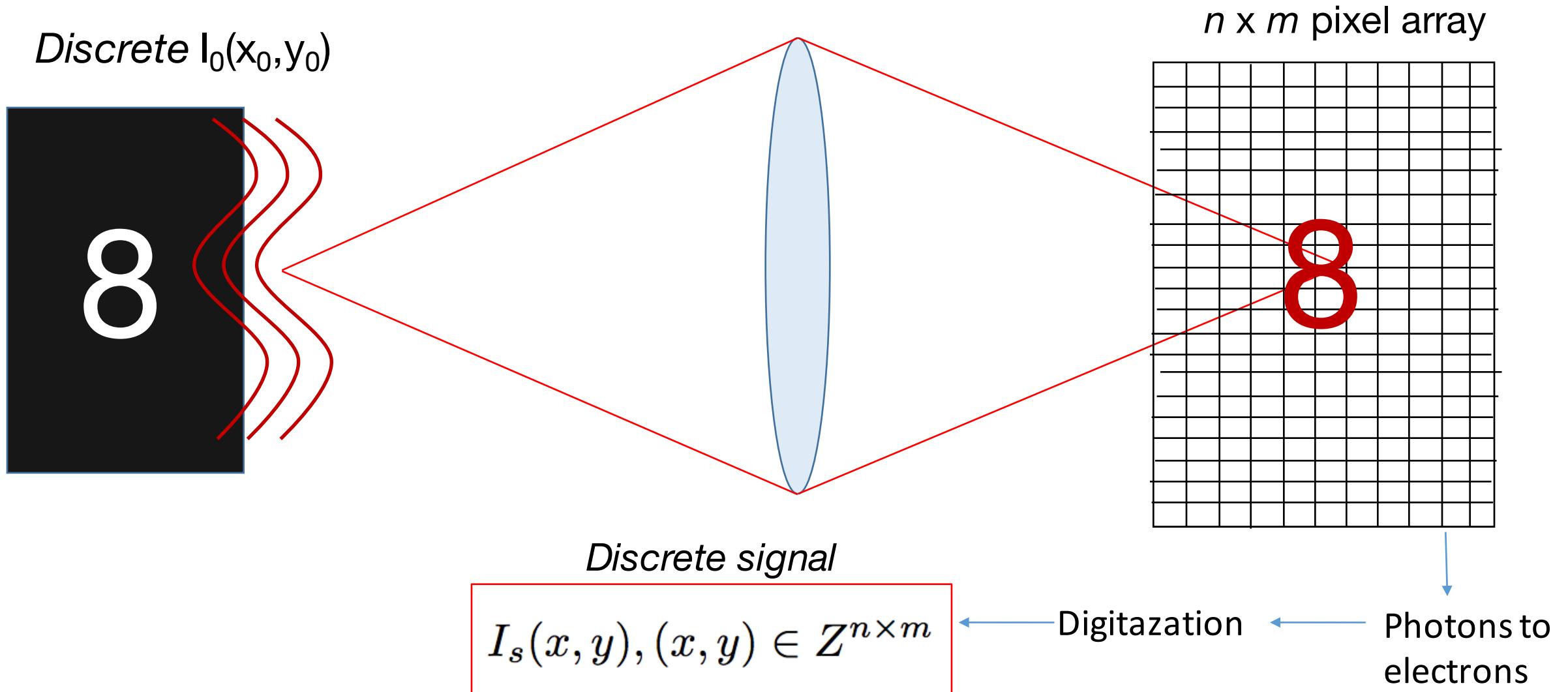
Discretize vectors
(and matrices)



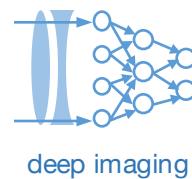
17
20
22
21
23
25
24
26
29



Simple model of image formation



Bringing together physical and digital image representations



Physical Layers

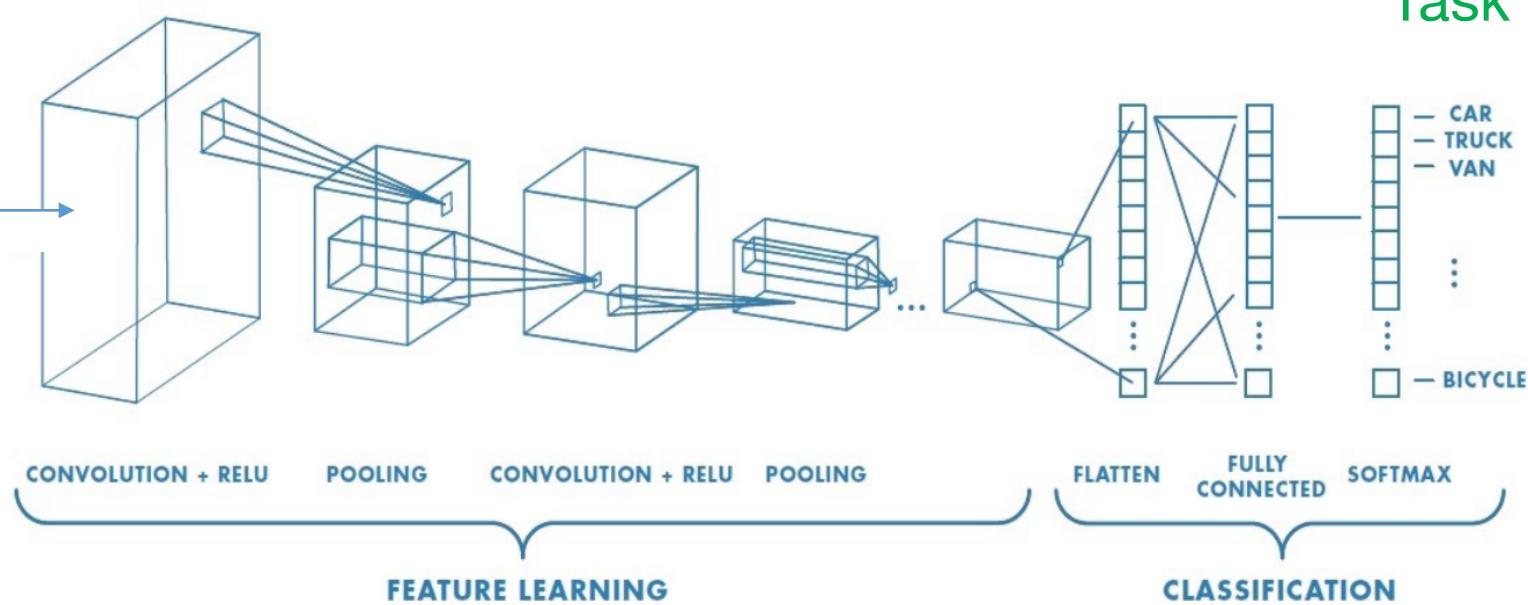
Physical Object I_0

$$f[]$$

Digitized image
 I_s

$$I_s = f[I_0]$$

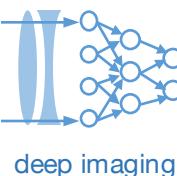
Digital Layers



Digital layers

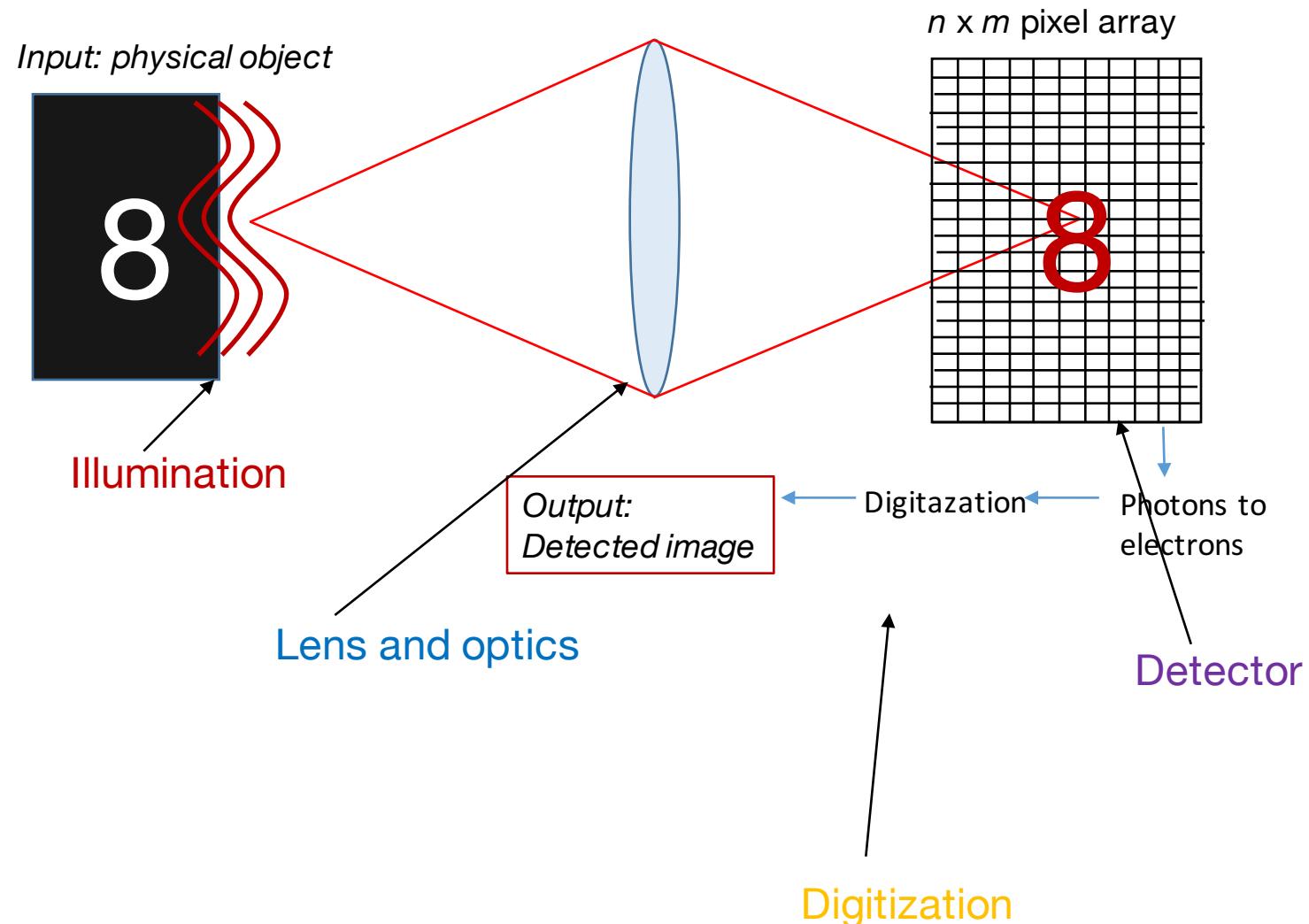
Physical layers

$$\text{Task} = W_n \dots \text{ReLU}[W_1 \text{ReLU}[W_0 f[I_0]] \dots]$$



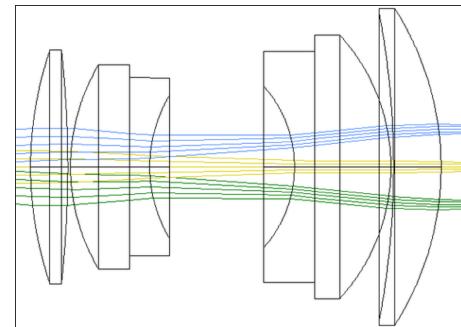
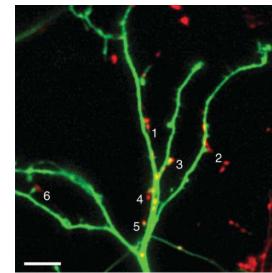
What physical parameters effect image formation $f[]$?

- **Illumination**
 - Spatial pattern
 - Angle of incidence
 - Color, polarization
- **Lens and optics**
 - Position/orientation
 - Shape
 - Focus
 - Transparency
- **Detector**
 - Pixel size
 - Pixel shape & fill factor
 - Color filters
 - Other filters
- **Digitization**
 - E to P curves
 - Digitization schemes/thresholds
 - Data transmission, multiplexing
- **Physical object**



First model of light we'll consider – Incoherent light

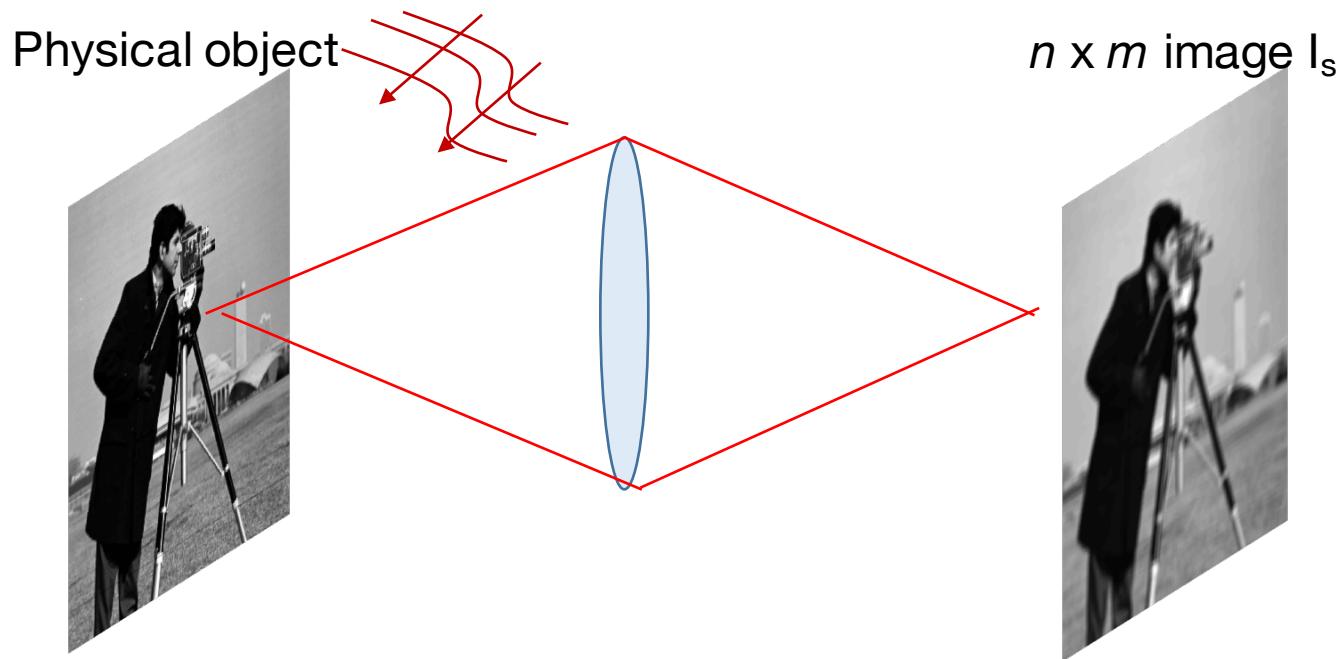
- Interpretation #1: Radiation (*Incoherent*)
- Model: Rays



- Real, non-negative
- Models absorption and brightness

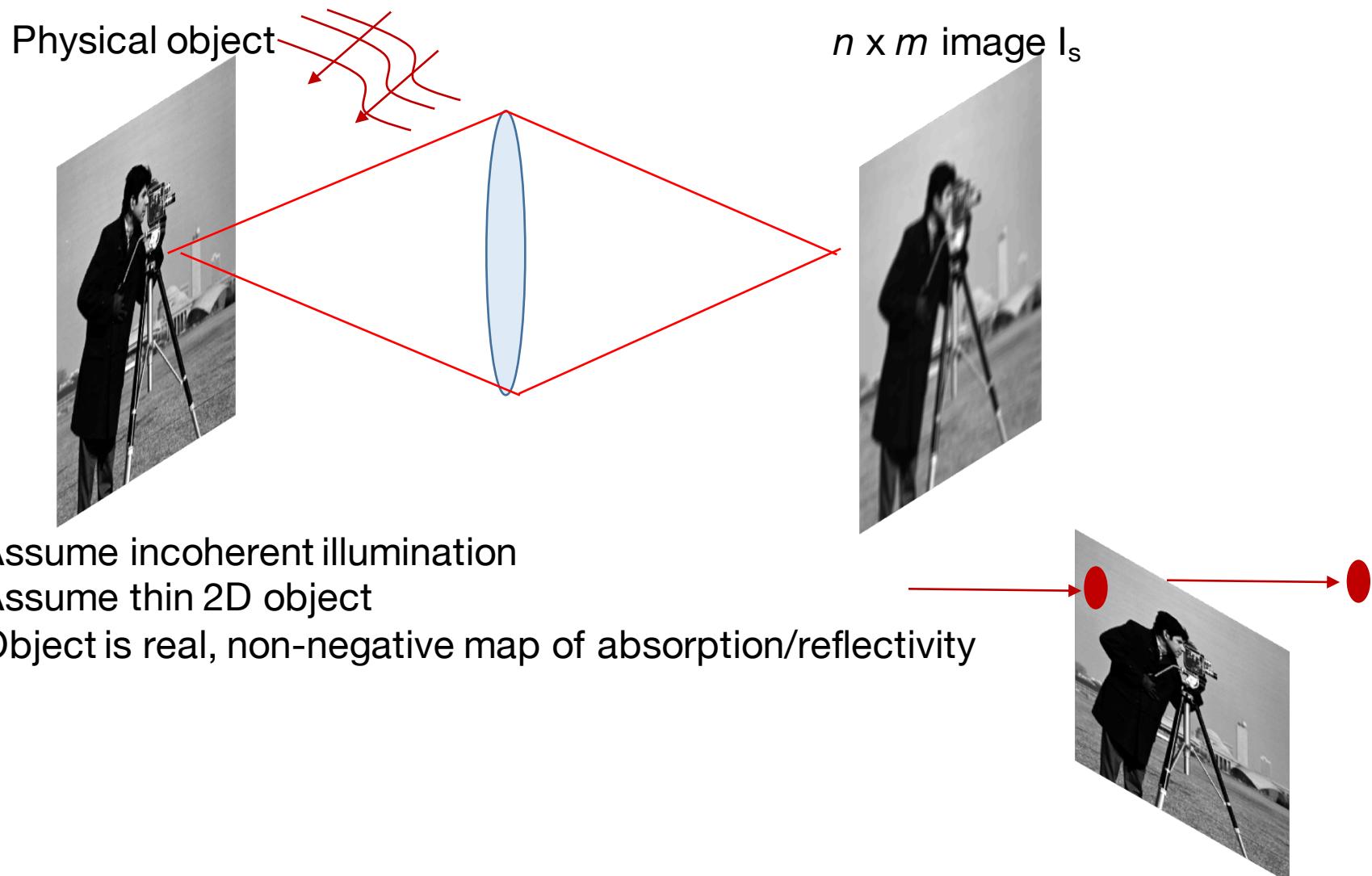
$$I_{\text{tot}} = I_1 + I_2$$

Example #1: Optimized illumination pattern (one color)

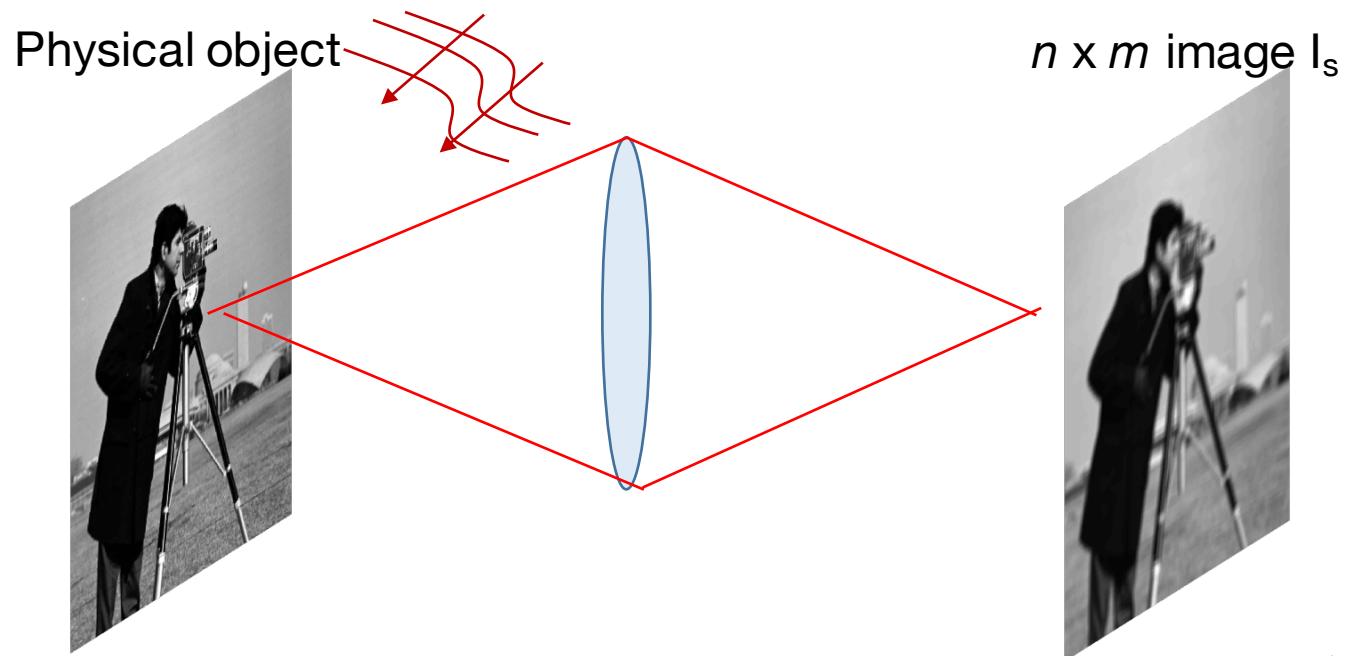


- Assume incoherent illumination
- Assume thin 2D object
- Object is real, non-negative map of absorption/reflectivity

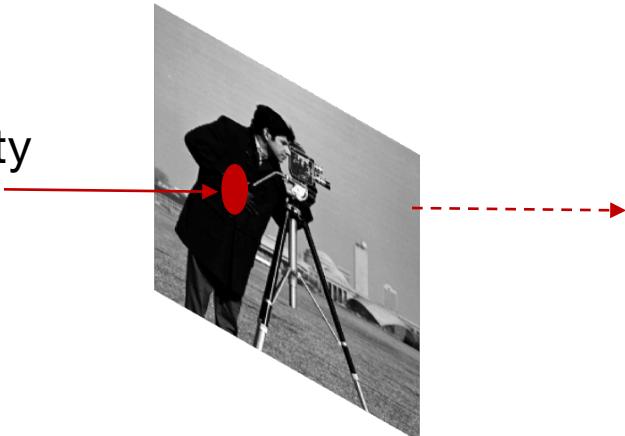
Example #1: Optimized illumination pattern (one color)



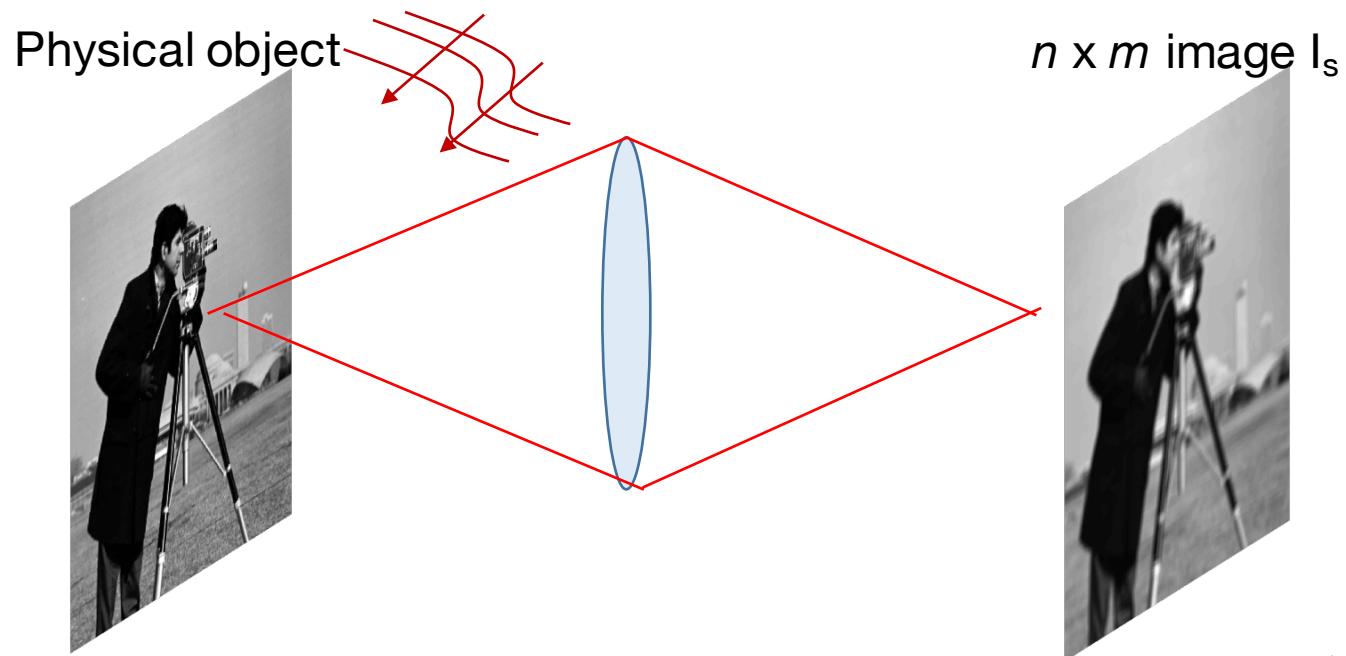
Example #1: Optimized illumination pattern (one color)



- Assume incoherent illumination
- Assume thin 2D object
- Object is real, non-negative map of absorption/reflectivity

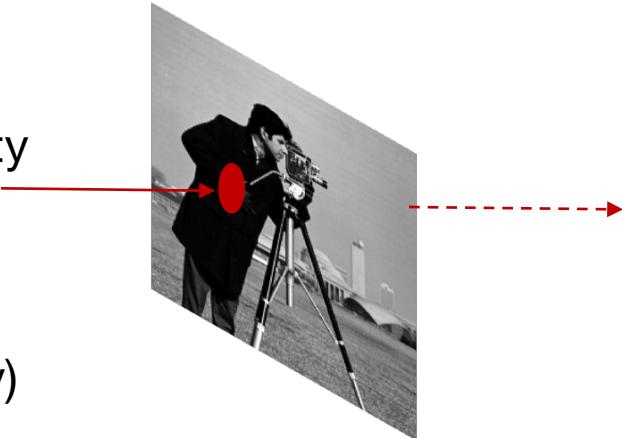


Example #1: Optimized illumination pattern (one color)

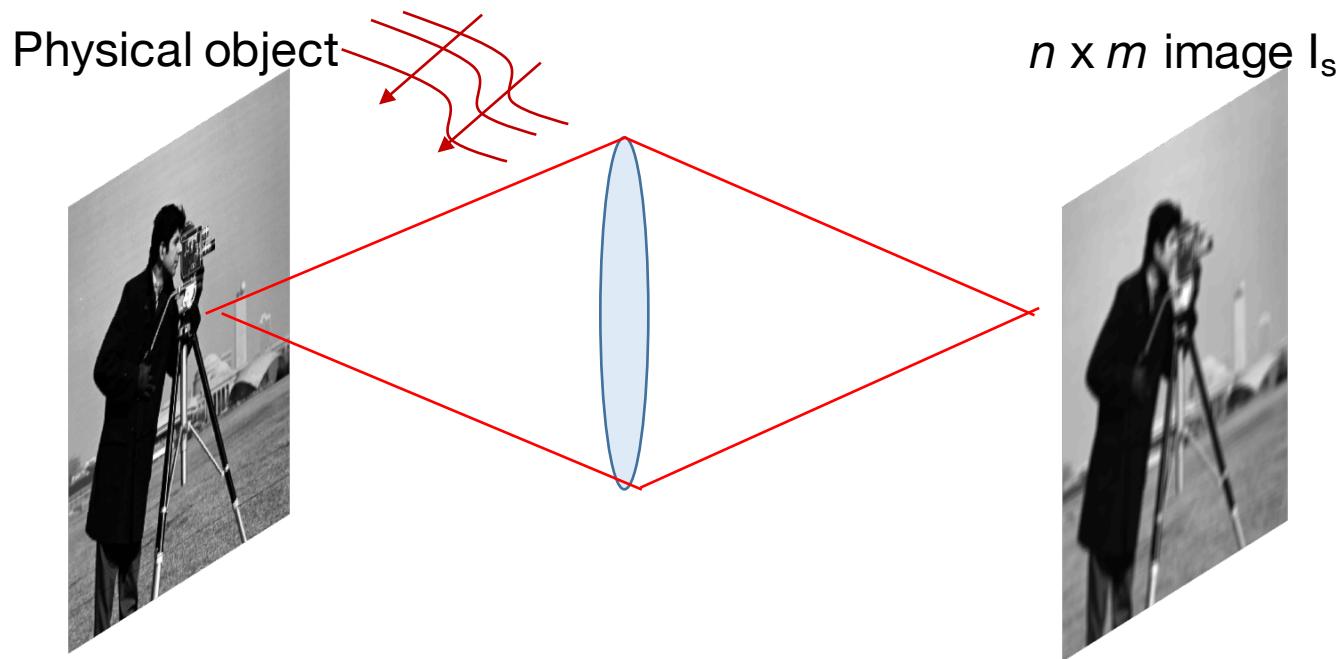


- Assume incoherent illumination
- Assume thin 2D object
- Object is real, non-negative map of absorption/reflectivity

Object absorption: $I_0(x,y)$
 Illumination pattern: $s(x,y)$
 Light exiting object surface: $I_e(x,y) = I_0(x,y) \circ s(x,y)$



Example #1: Optimized illumination pattern (one color)

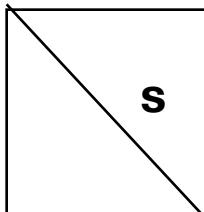


- Assume incoherent illumination
- Assume thin 2D object
- Object is real, non-negative map of absorption/reflectivity

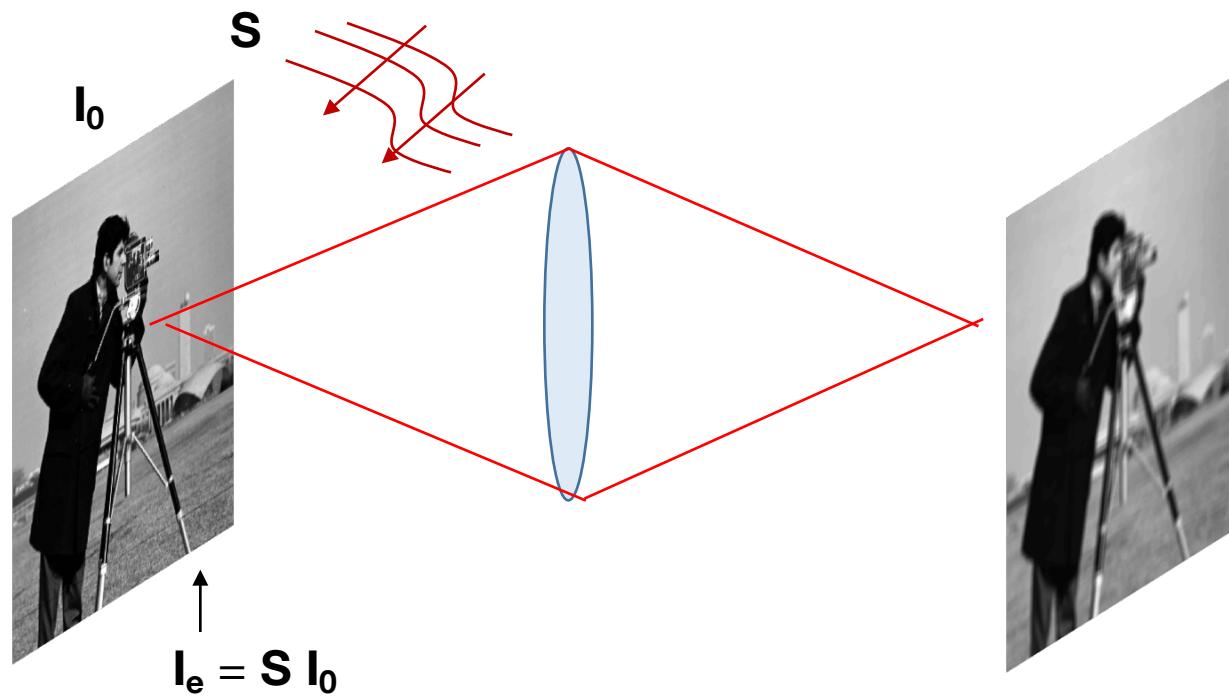
Modeling
incoherent
illumination

$$I_e(x,y) = I_0(x,y) \circ s(x,y)$$

$$I_e = S I_0$$

$$\text{diag}(S) = s$$


Example #1: Optimized illumination pattern (one color)



First, assume perfect camera:
intensity at image plane $I_p = I_e = S I_0$

Example #1: Optimized illumination pattern (one color)

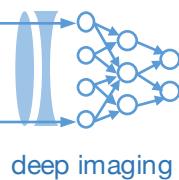


Training data:

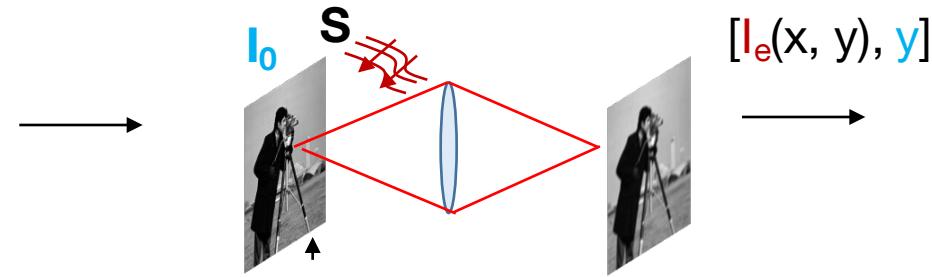
$[I_0(x, y), y]$

I_0 : 100 x 100

Label y : 1x2



Example #1: Optimized illumination pattern (one color)



Training data:

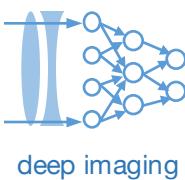
$$[I_0(x, y), y]$$

I_0 : 100 x 100

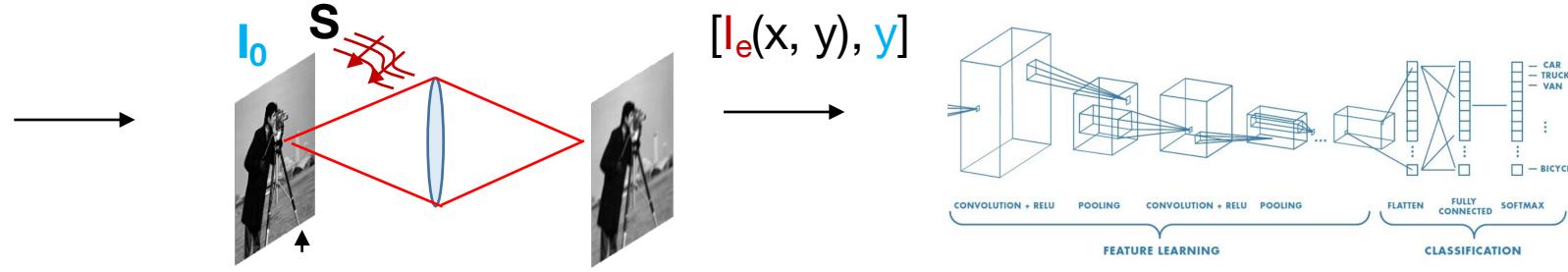
Label y : 1x2

$$I_e(x, y) = S I_0(x, y)$$

Physical Layer



Example #1: Optimized illumination pattern (one color)



Training data:

$$[I_0(x, y), y]$$

I_0 : 100 x 100

Label y : 1x2

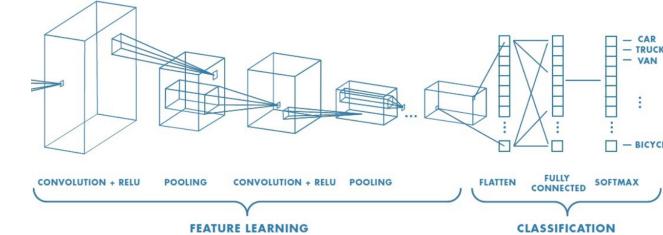


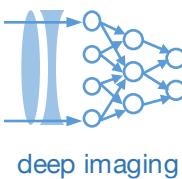
$$I_e(x, y) = S I_0(x, y)$$

Physical Layer

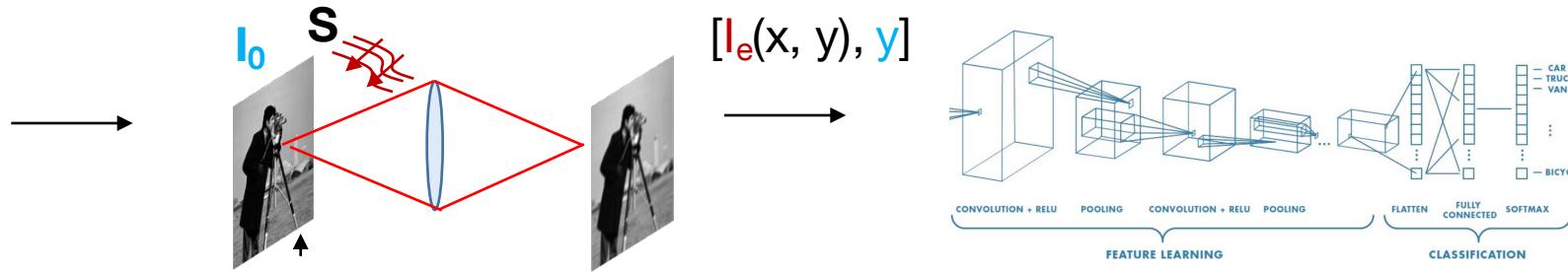


$$\text{Task} = W_n \dots \text{ReLU}[W_1 \text{ReLU}[W_0 I_e] \dots]$$





Example #1: Optimized illumination pattern (one color)



Training data:

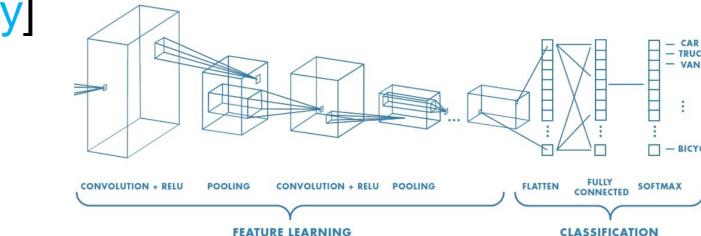
$$[I_0(x, y), y]$$

I_0 : 100 x 100

Label y : 1x2

$$I_e(x,y) = S I_0(x,y)$$

Physical Layer

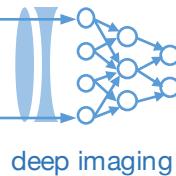


Tensorflow
1.0 code

```
training_images = tf.placeholder(tf.float32, [image_size, None])
training_labels = tf.placeholder(tf.float32, [None, 3])
illumination_pattern = tf.truncated_normal([image_size, 1], stddev = 0.1)
illumination_matrix = tf.linalg.diag(illumination_pattern)
illumianted_images = tf.matmul(illumination_matrix, training_images)

#Now, train CNN of your choice using [illumianted_images, training_labels]
#Output of train/test will be CNN weights, classification performance AND illumination_pattern!
```

See CoLab Notebook for implementation in Tensorflow 2.0

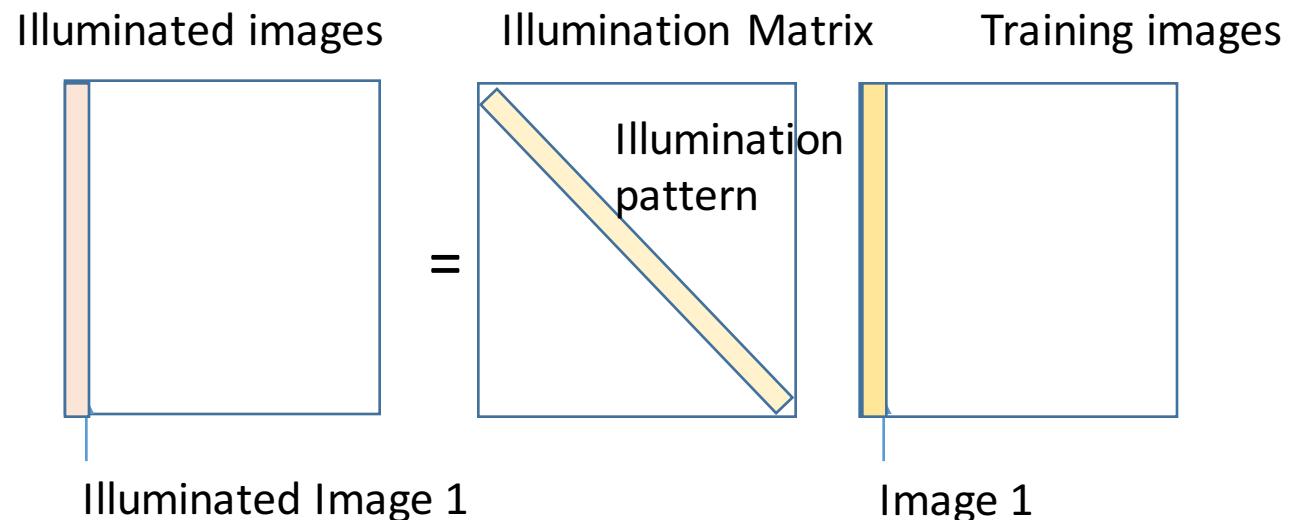


Option 1: tf.linalg.matmul

1.0 code

```
training_images = tf.placeholder(tf.float32, [image_size, None])
training_labels = tf.placeholder(tf.float32, [None, 3])
illumination_pattern = tf.truncated_normal([image_size, 1], stddev = 0.1)
illumination_matrix = tf.linalg.diag(illumination_pattern)
illuminated_images = tf.matmul(illumination_matrix, training_images)

#Now, train CNN of your choice using [illuminated_images, training_labels]
#Output of train/test will be CNN weights, classification performance AND illumination_pattern!
```



Option 2: tf.linalg.multiply (see CoLab Notebook)

We can also add in some lens blur

Lenses blur and rescale images:
(We'll learn how exactly next few weeks)

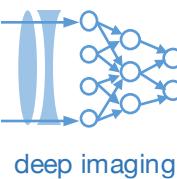
Input
intensity



$$\text{Convolution filter } h \\ * \\ \begin{matrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \end{matrix}$$



Output intensity



We can also add in some lens blur

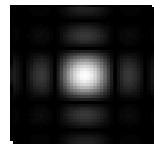
Lenses blur and rescale images:
(We'll learn how exactly next few weeks)

Input intensity



Convolution filter h

$*$



=

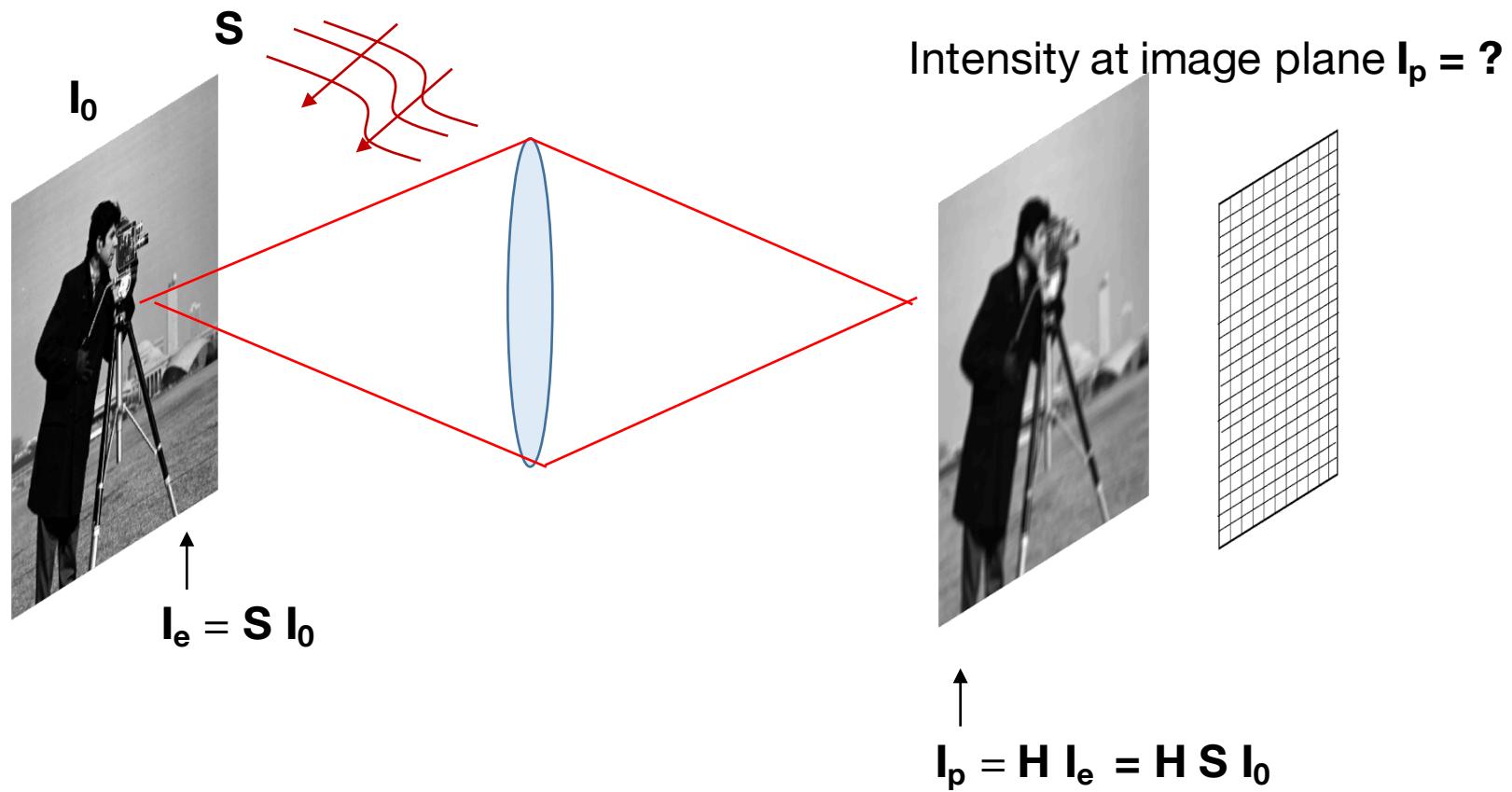


Output intensity

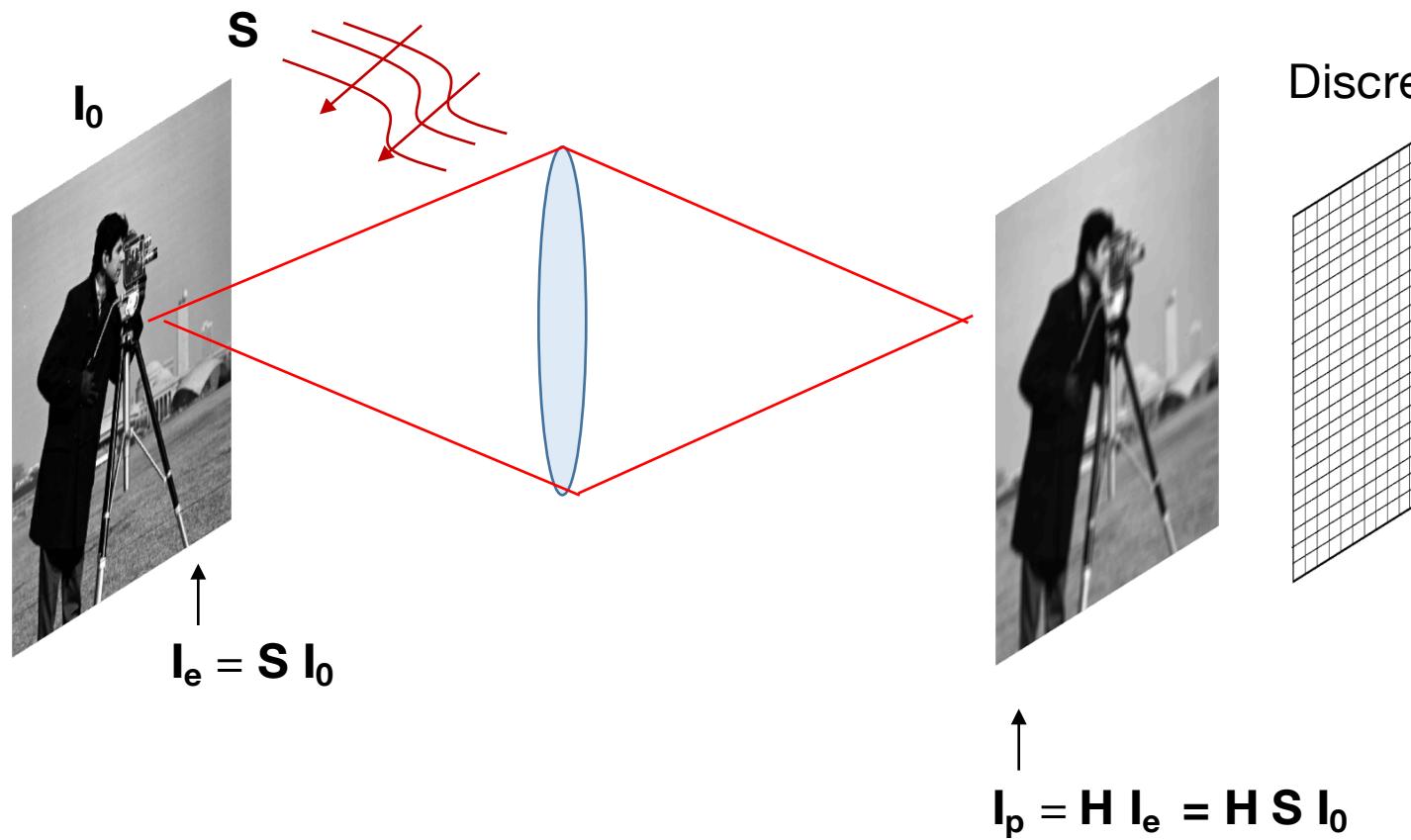
$$I_p(x, y) = I_e(x/M, y/M) * h(x, y)$$

Assuming we've resized by M , $I_p = I_e * h = H I_e$

Simple mathematical model of image formation



Simple mathematical model of image formation

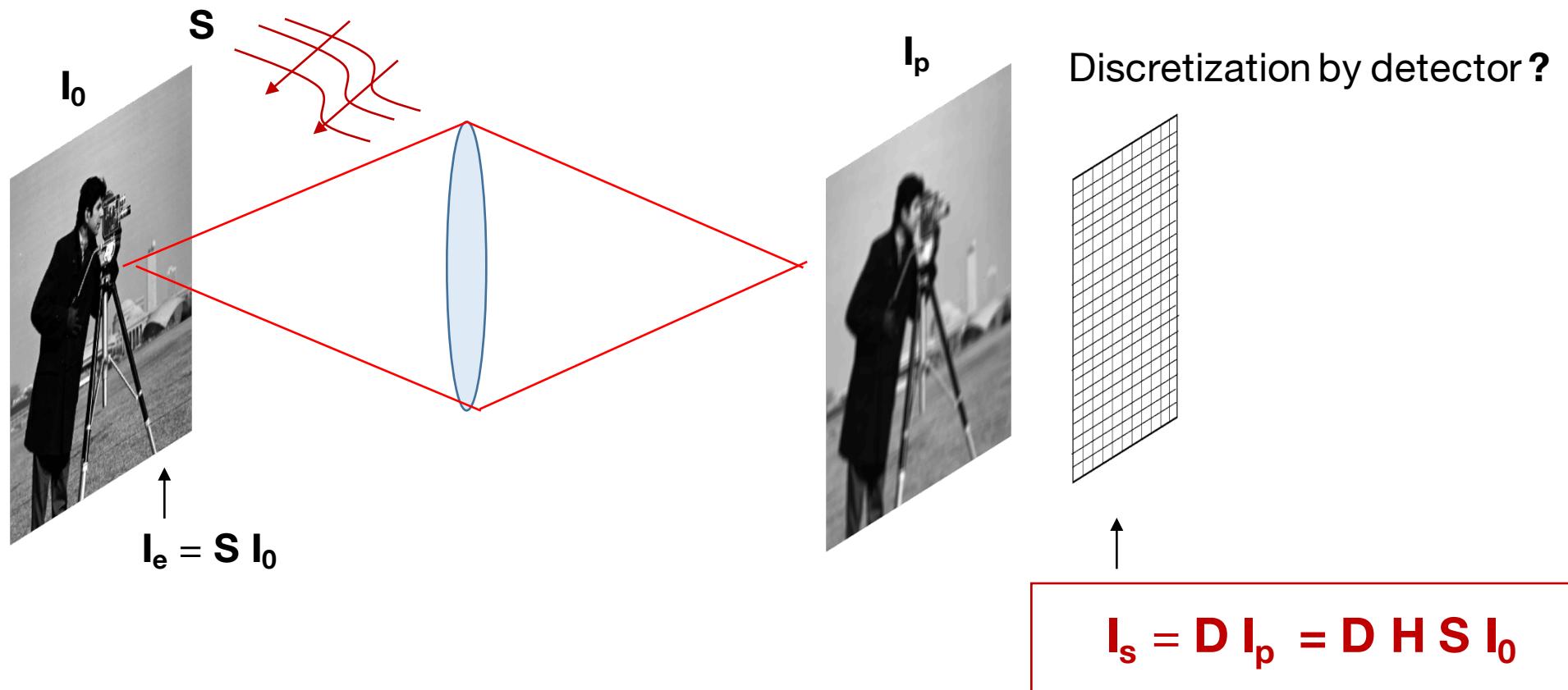


Use downsampling matrix
(sum-pooling)

$$D =$$

0.5 0.5 0 0 0
0.5 0.5 0 0 0
0.5 0.5 0 0 0
⋮ ⋮ ⋮ ⋮ ⋮

Simple mathematical model of image formation



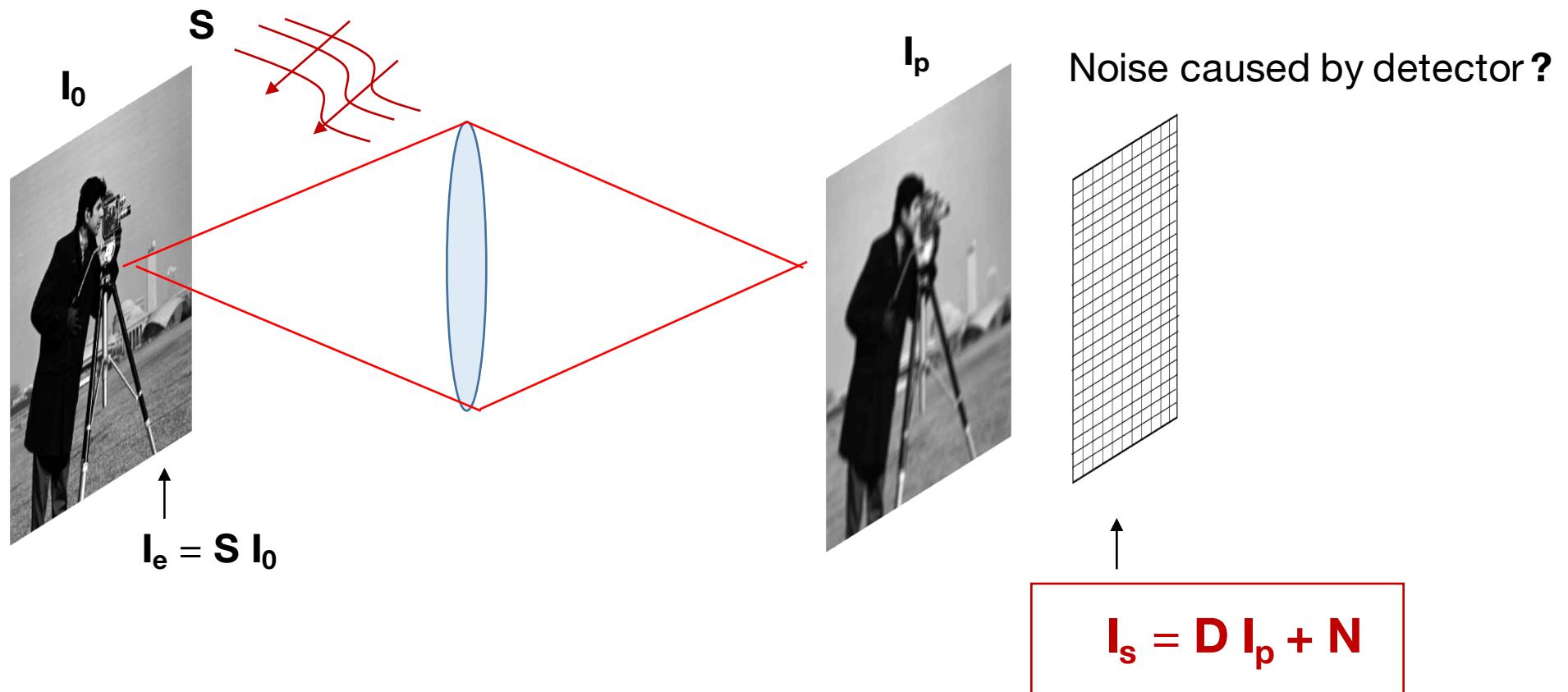
$$D =$$

w1 w2 0 0 0
w3 w4 0 0 0
w5 e6 0 0 0

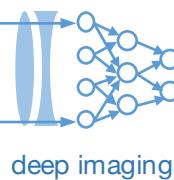
⋮ ⋮ ⋮

*Can make these learnable weights

Simple mathematical model of image formation



Can also add in detector-dependent noise $N = k * np.random.randn(dx, dy)$
 (zero-mean Gaussian noise example)

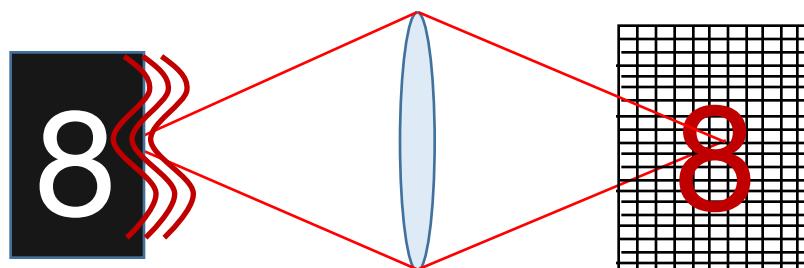


Task

Physical Layers

Physical world

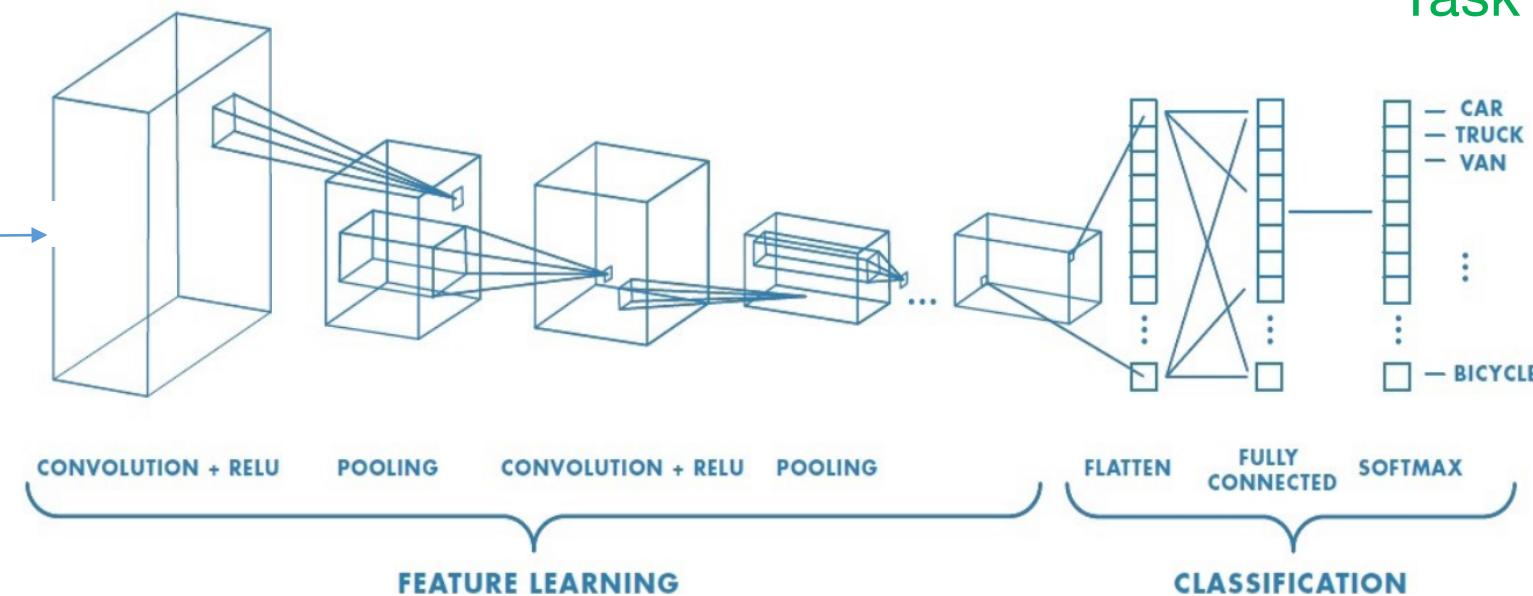
$$I_0(x_0, y_0)$$



Digital Image

$$I_s(x, y)$$

Digital Layers



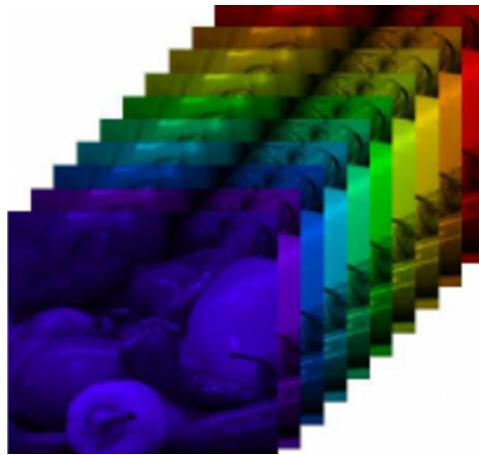
Digital layers

$$\text{Task} = W_n \dots \text{ReLU}[W_1 \text{ReLU}[W_0 f[I_0]]] \dots$$

Physical layers

$$\text{Task} = W_n \dots \text{ReLU}[W_1 \text{ReLU}[W_0 \mathbf{D} \mathbf{H} \mathbf{S} I_0]] \dots$$

Example #2: Optimized color filter for a grayscale camera

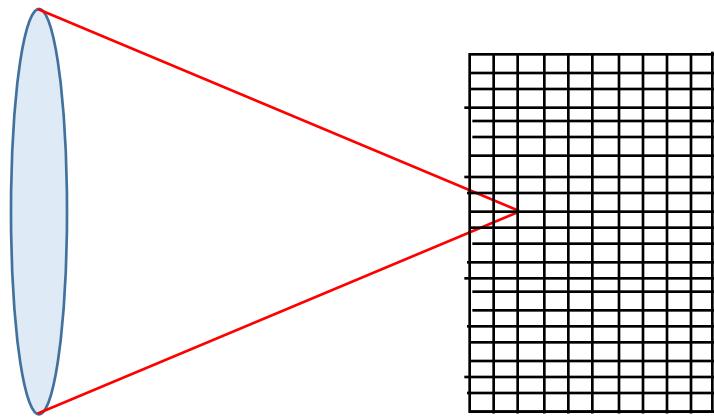
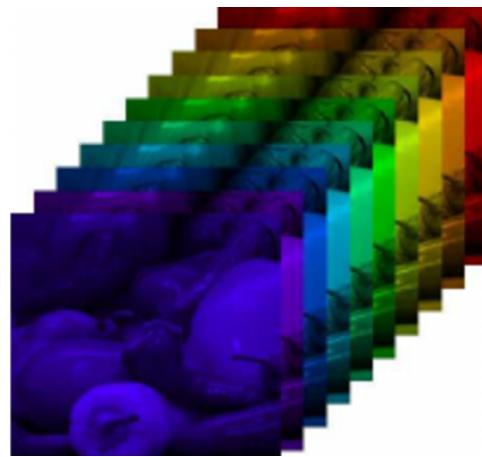


“Ground truth” object:

$$I_0(x, y, \lambda)$$

100 x 100 pix. x 30 spectral channels

Example #2: Optimized color filter for a grayscale camera



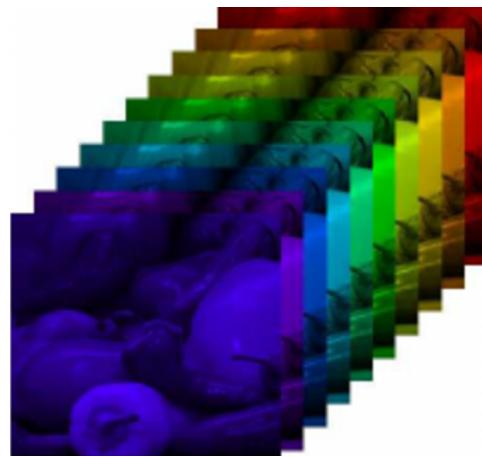
Monochromatic
camera sensor

“Ground truth” object:

$$I_0(x, y, \lambda)$$

100 x 100 pix. x 30 spectral channels

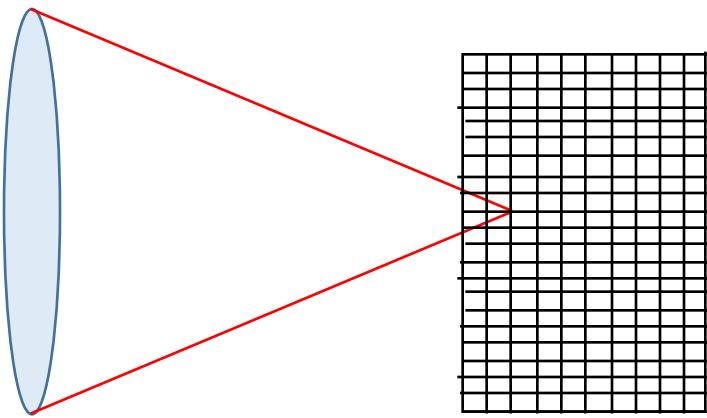
Example #2: Optimized color filter for a grayscale camera



“Ground truth” object:

$$I_0(x, y, \lambda)$$

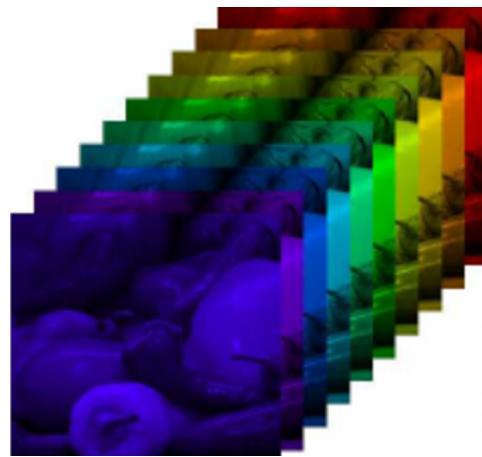
100 x 100 pix. x 30 spectral channels



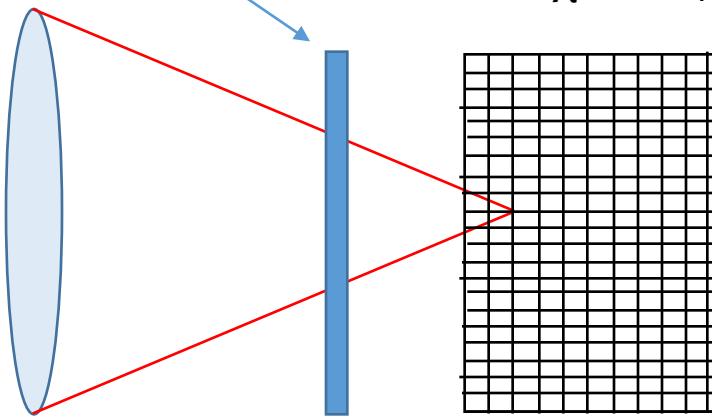
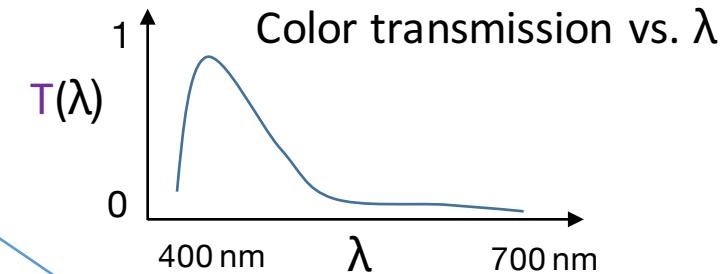
Monochromatic
camera sensor

$$I_s(x, y) = \sum_{\lambda} I_0(x, y, \lambda)$$

Example #2: Optimized color filter for a grayscale camera



Let's put a
color filter to
put here!



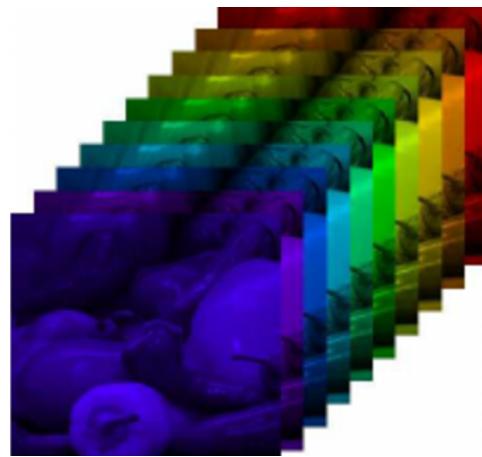
Monochromatic
camera sensor

“Ground truth” object:

$$I_0(x, y, \lambda)$$

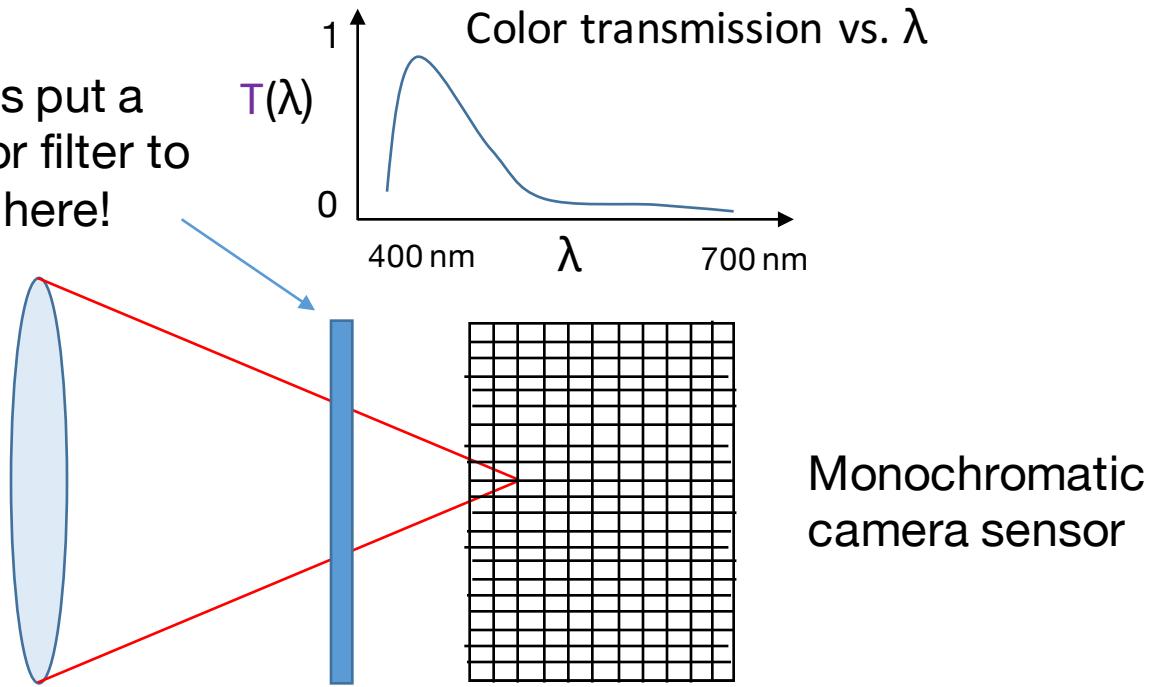
100 x 100 pix. x 30 spectral channels

Example #2: Optimized color filter for a grayscale camera



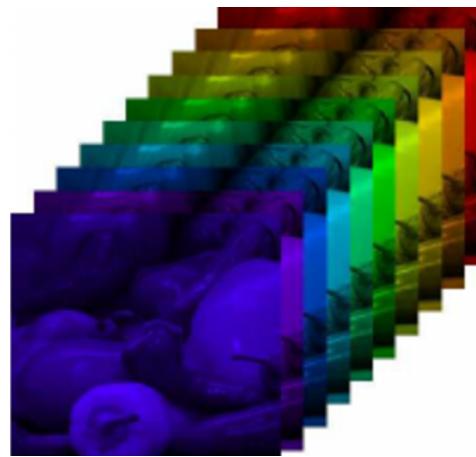
“Ground truth” object:
 $I_0(x, y, \lambda)$
 100 x 100 pix. x 30 spectral channels

Let's put a
color filter to
put here!

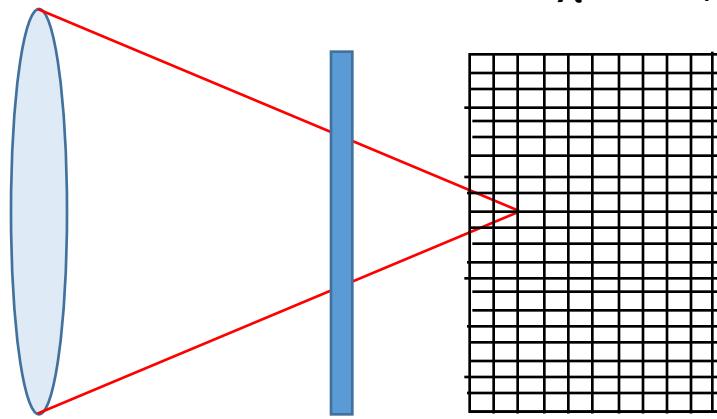
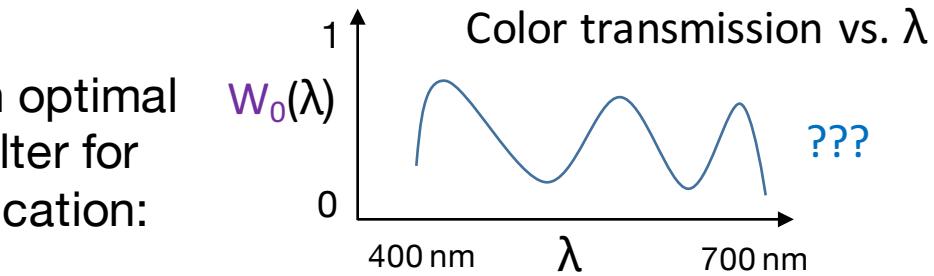


$$I_s(x, y) = \sum_{\lambda} T(\lambda) I_0(x, y, \lambda)$$

Example #2: Optimized color filter for a grayscale camera



Design optimal
color filter for
classification:



Monochromatic
camera sensor

Training data:

$$[I_0(x, y, \lambda), y]$$

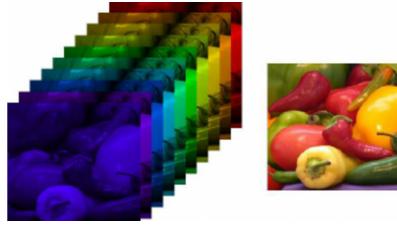
I_0 : 100 x 100 pix. x 30

Label y : 1x3 - pepper, broccoli, green beans

$$I_s(x, y) = \sum_{\lambda} W_0(\lambda) I_0(x, y, \lambda)$$

Physical Layer

Example #2: Optimized color filter for a grayscale camera



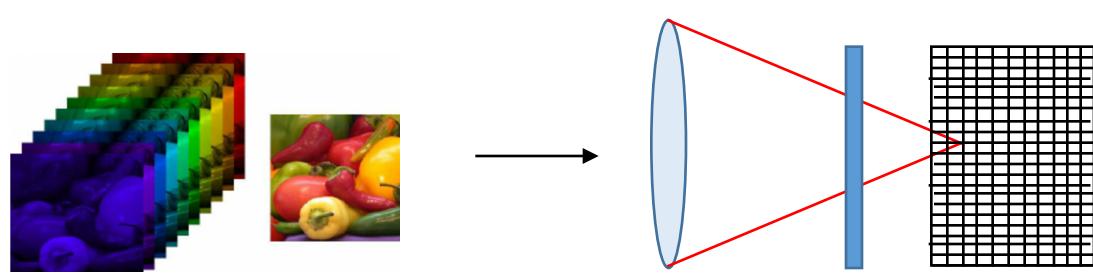
Training data:

$[I_0(x, y, \lambda), y]$

I_0 : 100 x 100 x 30

Label y : 1x3

Example #2: Optimized color filter for a grayscale camera



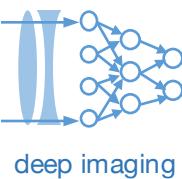
Training data:

$[I_0(x, y, \lambda), y]$
 $I_0: 100 \times 100 \times 30$

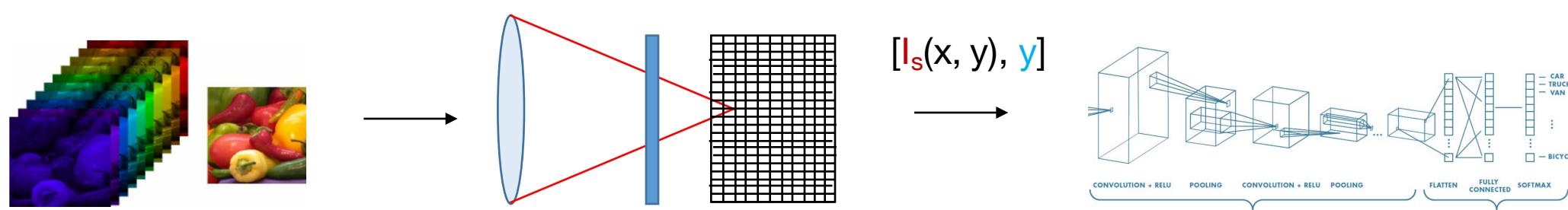
Label $y: 1 \times 3$

$$I_s(x, y) = \sum_{\lambda} W_0(\lambda) I_0(x, y, \lambda)$$

Physical Layer



Example #2: Optimized color filter for a grayscale camera



Training data:

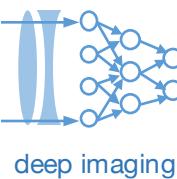
$[I_0(x, y, \lambda), y]$
 $I_0: 100 \times 100 \times 30$

Label $y: 1 \times 3$

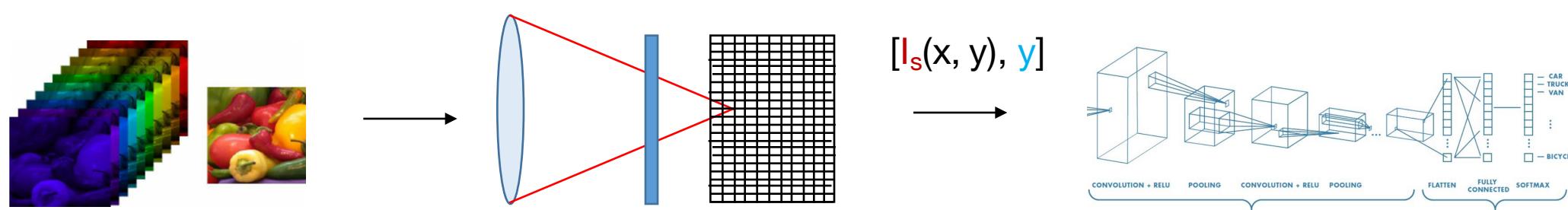
$$I_s(x, y) = \sum_{\lambda} W_0(\lambda) I_0(x, y, \lambda)$$

Physical Layer

$$\text{Task} = W_n \dots \text{ReLU}[W_1 \text{ReLU}[W_0 I_s] \dots]$$



Example #2: Optimized color filter for a grayscale camera



Training data:

$[I_0(x, y, \lambda), y]$
 $I_0: 100 \times 100 \times 30$

Label $y: 1 \times 3$

$$I_s(x, y) = \sum_{\lambda} W_0(\lambda) I_0(x, y, \lambda)$$

Physical Layer

Task = $W_n \dots \text{ReLU}[W_1 \text{ReLU}[W_0 I_s] \dots]$

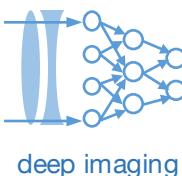
```

multispectral_data = tf.placeholder(tf.float32, [None, num_colors, image_size])
veg_labels = tf.placeholder(tf.float32, [None, 3])
filter_weights = tf.truncated_normal([num_colors, 1], stddev = 0.1)
filtered_images = tf.einsum('aij,jk->aijk', multispectral_data, filter_weights)

#Now, train CNN of your choice using [filtered_images, veg_labels]
#Output of training/testing will be CNN weights, classification performance AND filter_weights!

```

Example implementation with Tensorflow 1.0 code



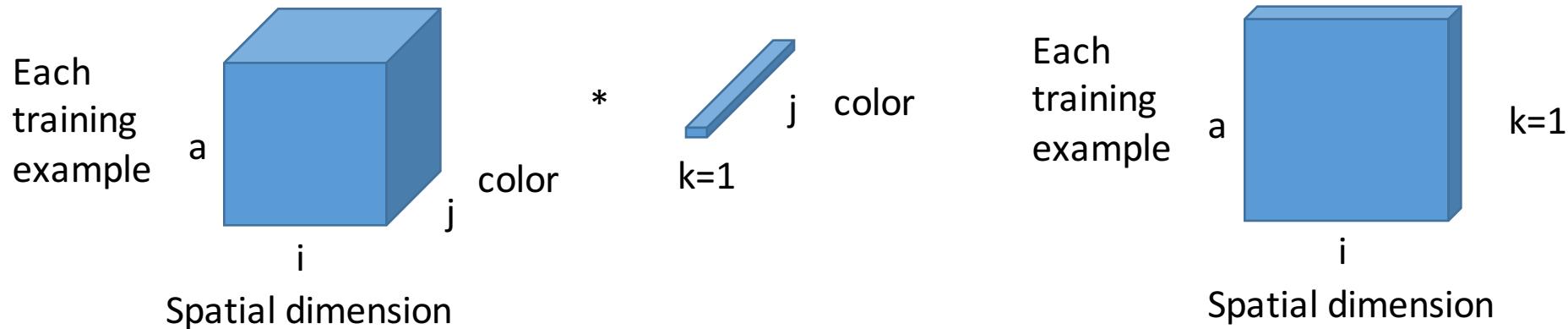
Tensorflow: operations to sum along 3rd (or higher) dimension

Option 1: Einsum (shown as Tensorflow 1.0 code, and also applicable in Tensorflow 2.0)

```
filtered_images = tf.einsum('aij,jk->aijk', multispectral_data, filter_weights)
```

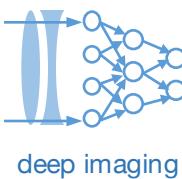
Option 2: tf.reduce_sum

```
filtered_images = tf.reduce_sum(multispectral_data * filter_weights, axis=2)
```



Option 3: Locally connected conv2D with a 1x1 filter size

<https://github.com/keras-team/keras/blob/master/keras/layers/local.py#L183>



Summary: simple physical layers for incoherent imaging

- Deal with sample/image intensities I , real and non-negative

- Effect of illumination is element-wise multiplication

 λ

$$I_e(x,y) = \mathbf{S} I_0(x,y)$$

- Imaging systems blur the object via point-spread function matrix H

$$I_b(x,y) = H I_0(x,y)$$

- Discrete pixels down-sample the object via

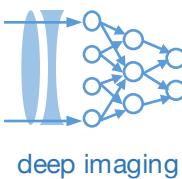
$$I_d(x,y) = D I_0(x,y)$$

- Add noise into measurement

$$I_N(x,y) = D I_0(x,y) + N$$

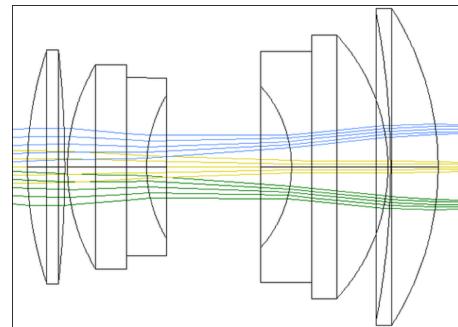
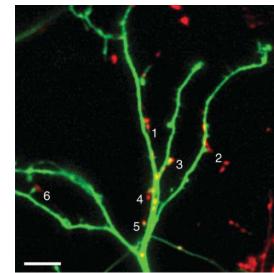
- Different colors add linearly

$$I_s(x, y) = \sum I_0(x, y, \lambda)$$



What is light and how can we model it?

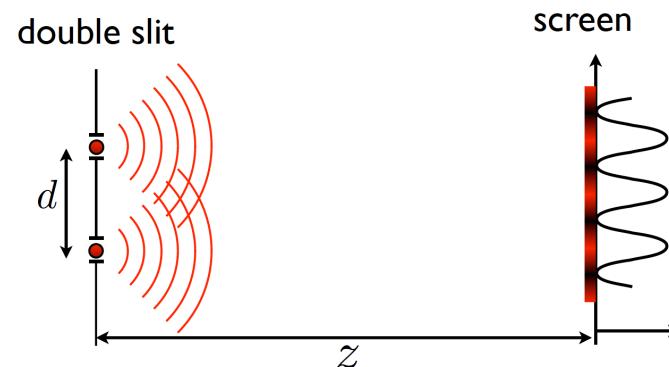
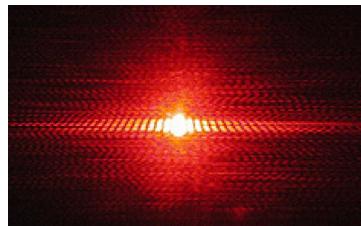
- Interpretation #1: Radiation (*Incoherent*)
- Model: Rays



- Real, non-negative
- Models absorption and brightness

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

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



- Complex field
- Models Interference

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

Next class: Modeling coherent radiation as a wave