Lecture 20: Coherent physical layers and general guidelines

Machine Learning and Imaging

BME 548L
Roarke Horstmeyer
Announcements:

• Project Proposals due today
  • Submit here: https://deepimaging.github.io/homework/projectproposal

• Homework 4 is posted - https://deepimaging.github.io/homework/hw4
  • Due Thursday Nov. 5th
Summary of two models for image formation

• **Interpretation #1: Radiation** *(Incoherent)*
  
  • Model: Rays
  
  ![Rays Image](image1)

  - Real, non-negative
  - Models absorption and brightness
  
  \[ I_{\text{tot}} = I_1 + I_2 \]

  \[ I_s = H B S_0 \]

• **Interpretation #2: Electromagnetic wave** *(Coherent)*
  
  • Model: Waves
  
  ![Waves Image](image2)

  - Complex field
  - Models Interference
  
  \[ E_{\text{tot}} = E_1 + E_2 \]
Model of image formation for wave optics (coherent light):

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

1. Transmitted field $s_c(x,y) = C(x,y) s(x,y)$
Model of image formation for wave optics (coherent light):

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

2. Compute its 2D Fourier transform
   \( \hat{s}(f_x, f_y) \)

Discrete sample function \( s(x,y) \) (complex)
General rules for applying the Fourier transform in optics

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

\[ U_1(x_1) \quad \rightarrow \quad U_2(x_2) \sim F[U_1(x_1/M_1)] \]

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

\[ U_1(x_1) \quad \rightarrow \quad U_2(x_2) \sim F[U_1(x_1/M_2)] \]

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

\[ U_1(x_1) \quad \rightarrow \quad U_2(x_2) \sim F[U_1(x_1/M_3)] \]
Model of image formation for wave optics (coherent light):

1. Transmitted field
   \[ s_c(x,y) = C(x,y) \cdot s(x,y) \]

2. Compute its 2D Fourier transform
   \[ \hat{s}(f_x, f_y) \]

3. Multiply by “aperture” function
   \[ A(f_x, f_y) \]
Model of image formation for wave optics (coherent light):

1. Transmitted field 
   \[ s_{c}(x,y) = C(x,y) s(x,y) \]

2. Compute its 2D Fourier transform 
   \[ \hat{s}(f_x, f_y) \]
   "Fourier plane"

3. Multiply by "aperture" function \[ A(f_x, f_y) \]

4. Compute inverse Fourier transform 
   \[ s'(x',y') \]
   "Blurred image"

- Discrete sample function \( s(x,y) \) (complex)
- 2D Fourier transform
- "Fourier plane"
- 2D inverse Fourier transform
Model of image formation for wave optics (coherent light):

1. Transmitted field $s_c(x,y) = C(x,y) \cdot s(x,y)$
2. Compute its 2D Fourier transform $\hat{s}(f_x, f_y)$
3. Multiply by “aperture” function $A(f_x, f_y)$
4. Compute inverse Fourier transform $s'(x', y')$ (complex)
5. Detector measures $|s'(x', y')|^2$

Discrete sample function $s(x,y)$ (complex)
Model of image formation for wave optics (coherent light):

1. Transmitted field $s_c(x,y) = C(x,y) s(x,y)$
2. Compute its 2D Fourier transform $\hat{s}(f_x, f_y)$
3. Multiply by “aperture” function $A(f_x, f_y)$
4. Compute inverse Fourier transform $s'(x',y')$ (complex)
5. Detector measures $|s'(x',y')|^2$

Model #1: $I_c(x,y) = |F^{-1}AFCs|^2$
Model of image formation for wave optics (coherent light):

1. Transmitted field
   \[ s_c(x,y) = C(x,y) \cdot s(x,y) \]

2. “Aperture” function
   \[ A(f_x, f_y) \]

3. Compute complex blur function
   \[ h(x,y) = F[A(f_x, f_y)] \]
Model of image formation for wave optics (coherent light):

1. Transmitted field
   \[ s_c(x,y) = C(x,y) \cdot s(x,y) \]

2. “aperture” function
   \[ A(f_x, f_y) \]

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

4. Blur image:
   \[ s' = s_c(x',y') \ast h(x',y') \]
Model of image formation for wave optics (coherent light):

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

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

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

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

5. Detector measures
   \(|s'(x',y')|^2\)

Model #2: \( I_c(x,y) = |h * Cs|^2 \)
You typically go between 4 functions to describe one imaging system:

- Coherent point-spread function: $h_c(x)$
- Coherent transfer function: $H_c(f_x)$
- Incoherent point-spread function: $h_i(x)$
- Incoherent transfer function: $H_i(f_x)$

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

$$I_s = h_i \ast B \cdot S_0$$

**Interpretation #2: Electromagnetic wave (Coherent)**
- Model: Waves
  - Complex-valued
  - Sample abs./phase $S$
  - Illumination wave $B$
  - Blur in $H$

$$I_c = |h_c \ast C \cdot S_c|^2$$
Coherent image formation equation as CNN operations

\[ I_c = D \vert h_c \ast C S_c \vert^2 \]

- 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:

\[ \partial \phi(x,y) = \exp\left[ j n t(x,y) / \lambda \right] \]

\( nt \) = 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:

```python
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 = tf.cos(thickness_map * n / wavelength)
mnist_phase_delay_imag = tf.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: $I_c(x,y) = |h \ast Cs|^2$

$s(x,y) = \partial \phi(x,y)$
Example #1: Optimizing coherent illumination pattern for improved classification

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

Unknown (complex weight variable)

Illumination $c(x,y)$

Camera blur $h$

$s(x,y) = \partial \phi(x,y)$
Example #1: Optimizing coherent illumination pattern for improved classification

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

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

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

Camera blur $h$

CNN
Example #1: Optimizing coherent illumination pattern for improved classification

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

Unknown

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

$\text{detected}_\text{image}$ then enters standard CNN classification pipeline

```python
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)
```
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?

Let's make \( A(f_x, f_y) \) any shape – it becomes a weight variable.
Example #2: Optimizing aperture shape for improved digit classification

\[ s(x,y) = \partial \phi(x,y) \]

\[ A(f_x, f_y) \]

Fixed plane-wave
Illumination \( c(x,y) \)

2D FT

2D IFT

CNN
Example #2: Optimizing aperture shape for improved digit classification

Fixed plane-wave Illumination $c(x,y)$

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

$A(f_x, f_y)$

CNN

2D FT

2D IFT

```
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

Input image data $I_0$

$f[]$

$I_s = f[I_0]$

Digitized

Digital Layers

Task

Some Examples:
- Optimized illumination
- Optimized sensor specifications
- Number of measurements and locations
- Radiation dosage, biomarkers
Q: Where and how should I implement my physical layer?
Q: Where and how should I implement my physical layer?

A: It depends on your data and implementation

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

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

\[ I_s = f[I_0] \]

- **Physical Layers**
  - Full simulation of different samples and their labels
  - Simulated labels

- **Digital Layers**
  - Task
  - 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

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

Examples:
- [Ultrasound scatterers, segmentation boundary]
- [Simulated cell body types, location]
- [CT phantom, 3D mesh surfaces]
Situation #1: Fully simulated physical layers

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

Existing annotated dataset $I_0$

Existing labels

Physical Layers: $I_s = f(I_0)$

Digitized

Digital Layers

Task
Situation #1: Fully simulated physical layers

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

$$I_s = f(I_0)$$
Situation #1: Fully simulated physical layers

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

Existing annotated dataset $I_0$

Pre-processing

Physical Layers

$\text{f} [ \ ]$

Digitized

$I_s = f[I_0]$

Existing labels

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

Digital Layers

Task
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

ML Optimization

Big dataset → TensorFlow

TensorFlow → TensorFlow
Situation #1: Fully simulated physical layers

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

Simulation and/or pre-processing
Python/Matlab/other

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

Experimental measurements → “Expert” annotation → Physical Layers

\[ \mathbf{l}_s = f(\mathbf{l}_0) \]

Digitized \( \mathbf{l}_s \) → Generated Labels → Digital Layers

Task
Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:

CNN → Classifier:
- Bright-field (1 image)
- FPM (29 images)
- Learned (1-2 images)

Illumination design:

0 1
π
-1 0.5

LED Pattern:
- Center LED
- Scanned
- Learned

Classification Accuracy:
- 75%
- 97%
- 98.5%

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:

\[ I_s = \sum w_j I_j \]

LED illumination model

<table>
<thead>
<tr>
<th>Situation #2: Experimentally-driven physical layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example: CNN-Optimized illumination for classification of malaria:</td>
</tr>
<tr>
<td>Physical layer:</td>
</tr>
<tr>
<td>[ I_s = \sum w_j I_j ]</td>
</tr>
<tr>
<td>LED illumination model</td>
</tr>
<tr>
<td>LED-illuminated images</td>
</tr>
<tr>
<td>Conv (5x5) + ReLU + MaxPooling</td>
</tr>
<tr>
<td>64</td>
</tr>
</tbody>
</table>
### Situation #2: Experimentally-driven physical layers

Example: CNN-Optimized illumination for classification of malaria:

<table>
<thead>
<tr>
<th>(a) Center</th>
<th>(b) All</th>
<th>(c) Off-axis</th>
<th>(d) Random</th>
<th>(e) DPC</th>
<th>(f) Optimized</th>
<th>(g) FP (29 im.)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
<td><img src="image2.png" alt="Image" /></td>
<td><img src="image3.png" alt="Image" /></td>
<td><img src="image4.png" alt="Image" /></td>
<td><img src="image5.png" alt="Image" /></td>
<td><img src="image6.png" alt="Image" /></td>
<td><img src="image7.png" alt="Image" /></td>
</tr>
</tbody>
</table>

**Infected**

- 70-90%

**Uninfected**

- 87%
- 97%
- 98.5%
Situation #2: Experimentally-driven physical layers

Optimized color LED patterns to classify malaria

Pattern 1, red
Pattern 1, green
Pattern 1, blue
Pattern 1, RGB

Max: 0.38

Pattern 2, red
Pattern 2, green
Pattern 2, blue
Pattern 2, RGB

Max: 0.70
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)

Weights $W$, $|W/W_{\text{max}}|$ (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

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

Increase \( \alpha \) 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.

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

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

\[
\text{Soft-max}(x) = \frac{\exp(-x)}{\Sigma \exp(-x)}
\]

Increase \( \alpha \) with iteration number.
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

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

Pros:
- Works pretty well
- Flexibly address convergence issues

Cons:
- A bit sensitive

Increase \( \alpha \) with iteration number

\[
\text{Soft-max}(x) = \frac{\exp(-x)}{\sum \exp(-x)}
\]
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...

Solution 1
Solution 2
Solution 3
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:

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

(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

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

- **Simulated “Real-world” objects**
- **Known, easy**
  - Forward model
  - Hard to get working...
  - But you have insights and guarantees!
- **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 I did in grad school to get ready for an experiment:

- Simulated “Real-world” objects
- Known, easy Forward model
- Hard to get working… data
- Inverse problem solver
- 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 \rightarrow Forward model \rightarrow data \rightarrow CNN as an inverse model \rightarrow 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 → Known, easy Forward model → data → CNN as an inverse model → Also easy…. Simulated "Real-world" objects

Noise, perturbations, etc. → But it's a black box!
```