Non-Local Image Denoising
&
Integral Images

Slides by: Stephan Garbin

Outline

1. **Non-Local Denoising**
   - Problem
   - Noise Model
   - Motivation for Non-Local Filters
   - Non-Local Means
   - Weighting for Non-Local Means

Outline

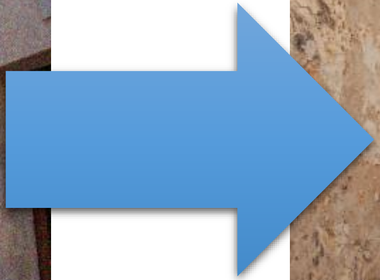
1. **Non-Local Denoising**
   - Problem
   - Noise Model
   - Motivation for Non-Local Filters
   - Non-Local Means
   - Weighting for Non-Local Means
2. **Efficient Template Matching**
   - Problem
   - What is an Integral Image?
   - How do we calculate it?
   - How do we use it?

Non-Local Denoising

Our Task

Noisy Image

Denoised Image



Noise Model

- Assumption: **Additive, Zero-Mean, ‘White’ Gaussian Noise**

Additive White Gaussian Noise

Noisy Image

$$\tilde{u} = u + n$$

Ideal (noiseless) Image

Noise Model

Additive White Gaussian Noise

Noisy Image

$$\tilde{u} = u + n$$

Ideal (noiseless) Image

$$\mathbb{P}(\tilde{u} \mid u) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}} e^{-\frac{\|u - \tilde{u}\|_2^2}{2\sigma^2}}$$

Pixels in Image

Noise Model **Conditional Distribution**

Noise Model

$$\tilde{u} = u + n$$

$$\mathbb{P}(\tilde{u} \mid u) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}} e^{-\frac{\|u - \tilde{u}\|_2^2}{2\sigma^2}}$$

We want to recover u from $\tilde{u}$

How?

- (Gaussian) Low-Pass?

How?

- (Gaussian) Low-Pass?
- Bilateral Filter?

How?

- (Gaussian) Low-Pass?
- Bilateral Filter?

- **Too much detail lost**
- **Blurry results**

Exploiting Self-Similarity

- *Idea:* Image shows **self-similarity** ➔ many similar PATCHES

Exploiting Self-Similarity

- <u>*Idea:* </u>Image shows **self-similarity** ➔ many similar PATCHES

Exploiting Self-Similarity

- *Idea:* Image shows **self-similarity** ➔ many similar PATCHES

Exploiting Self-Similarity

- *<u>Idea:</u>* Image shows **self-similarity** ➔ many similar PATCHES

Exploiting Self-Similarity

- ***Idea:*** Image shows **self-similarity** ➔ many similar PATCHES

Exploiting Self-Similarity

- *Idea:* Image shows **self-similarity** ➔ many similar PATCHES

Non-Local Means: Concept

“Restore current pixel as a weighted average of all pixels in the image”

Non-Local Means: Concept

<u>In Practise:</u>
Local neighbourhood
around pixel (e.g. 30x30)

"Restore current pixel as a weighted average of <u>*all*</u> pixels in the image"

Non-Local Means: Concept

In Practise:
Local neighbourhood
around pixel (e.g. 30x30)

"Restore current pixel as a *weighted* average of *all* pixels in the image"

Similarity determined by
patches around pixels

- Similar pixels count more
- Dissimilar pixels should not count

Non-Local Filters – General Structure

1. **Block-Matching**
    - *Procedure:* For every reference patch, centred at every pixel, find k most similar patches
        - For NL-Means: *All* patches in local neighbourhood

Non-Local Filters – General Structure

1. Block-Matching

- *Procedure:* For every reference patch, centred at every pixel, find k most similar patches
  - For NL-Means: *All* patches in local neighbourhood
- *Result:* 3d block of 2d patches for each reference patch

Note:
Does not need to be stored explicitly for NL-Means

Non-Local Filters – General Structure

1. **Block-Matching**
    - *Procedure:* For every reference patch, centred at every pixel, find k most similar patches
        - For NL-Means: A*ll* patches in local neighbourhood
    - *Result:* 3d block of 2d patches for each reference patch

2. **Filtering**
    - Use 3d blocks to derive denoised estimate

Non-Local Filters – General Structure

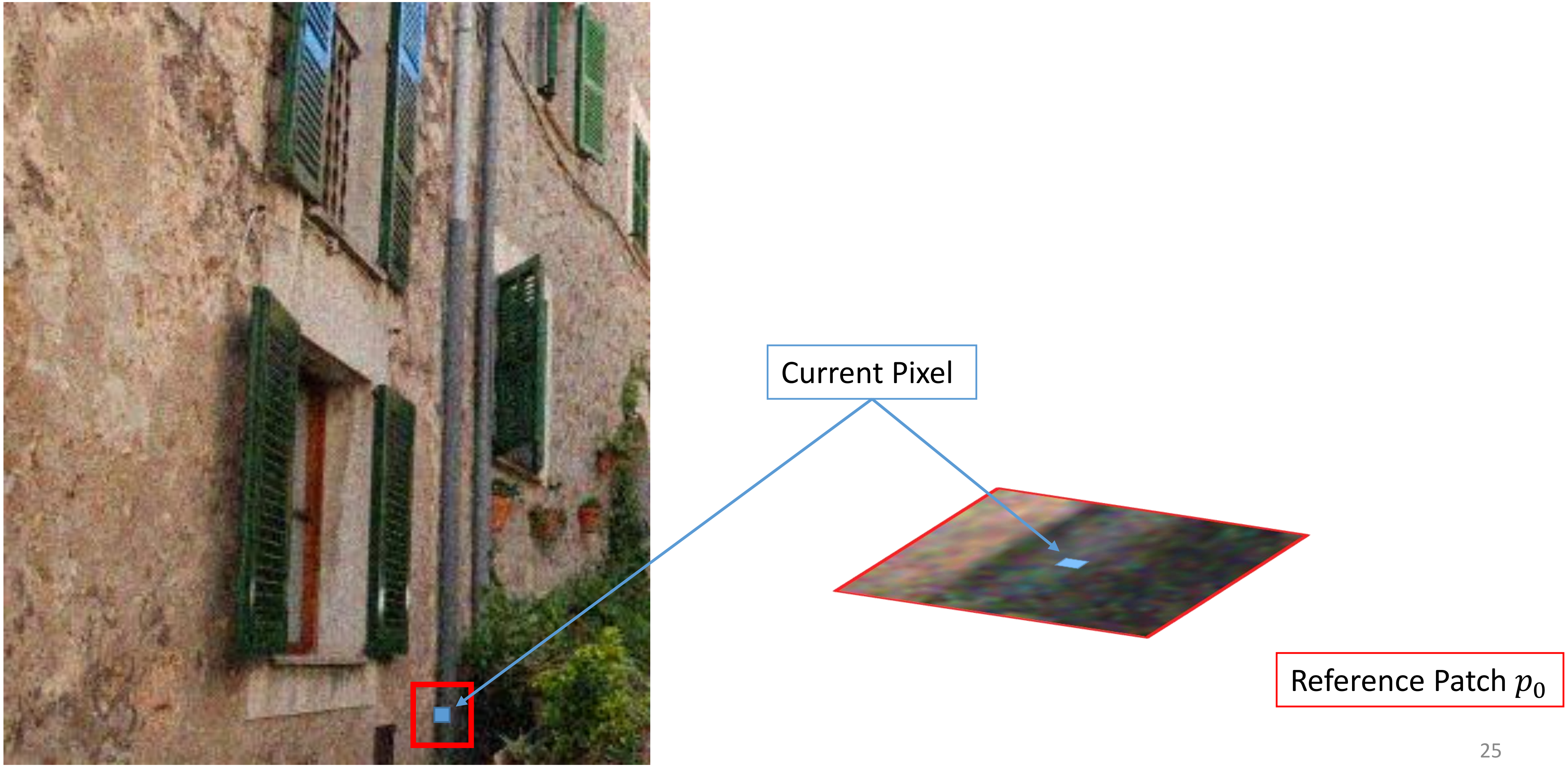
1. **Block-Matching**
   - *Procedure:* For every reference patch, centred at every pixel, find k most similar patches
     - For NL-Means: *All* patches in local neighbourhood
   - *Result:* 3d block of 2d patches for each reference patch

2. **Filtering**
   - Use 3d blocks to derive denoised estimate

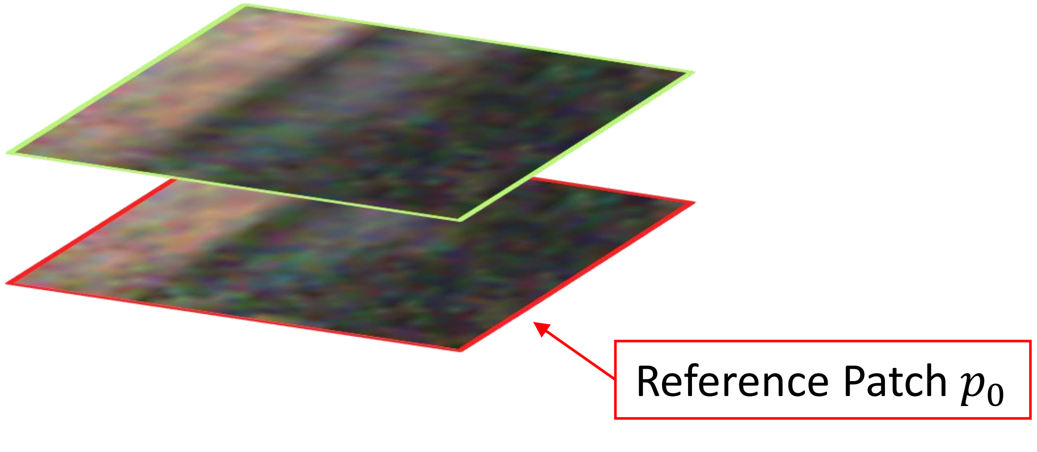
3. **(Aggregation)**
   - Combine (overlapping) patches to form final image

Note:
Only necessary for patch-wise implementation

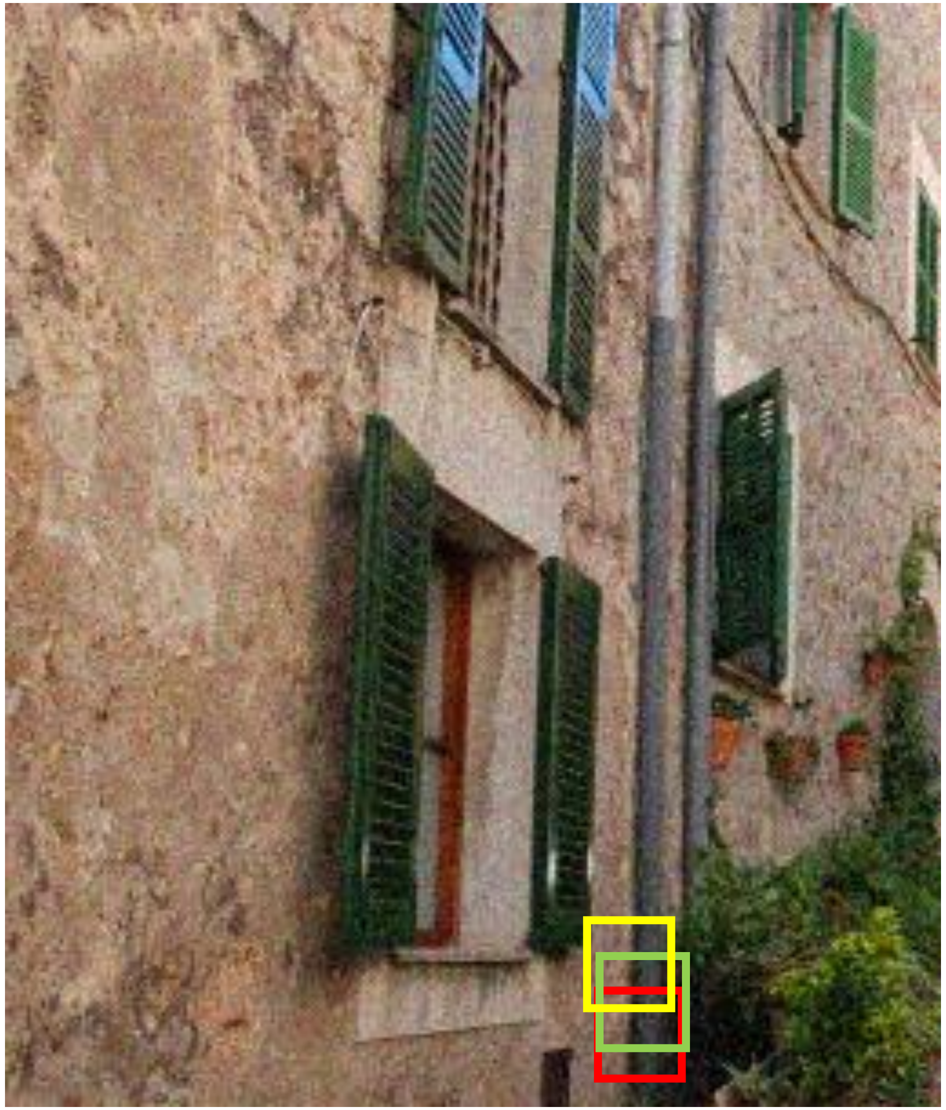
Building Blocks of 2d Patches

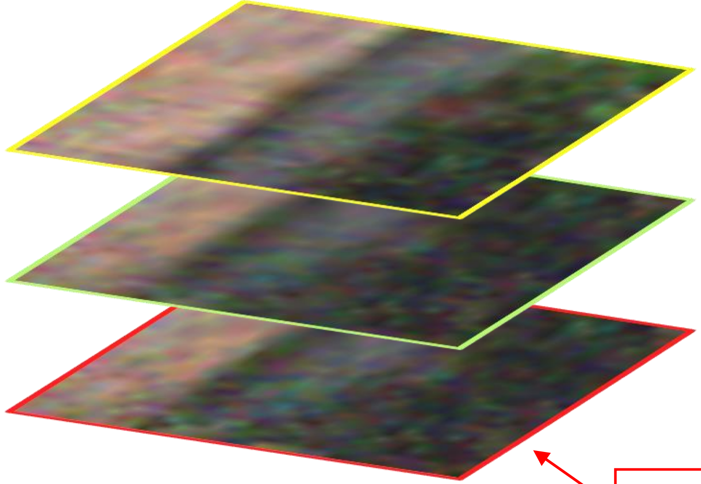


Building Blocks of 2d Patches



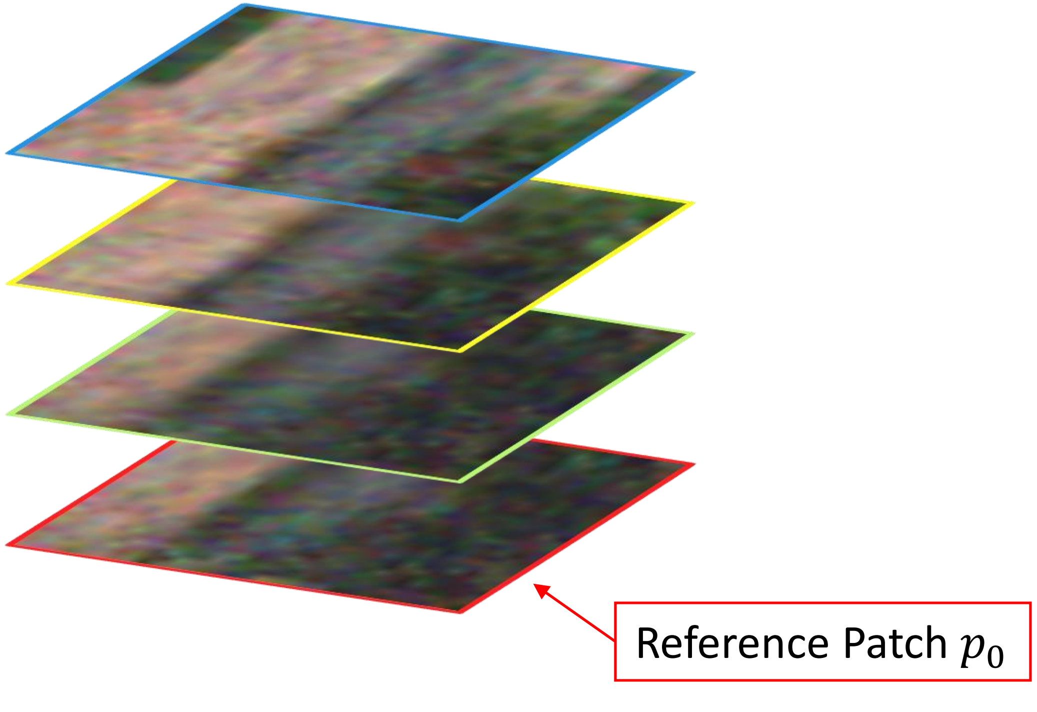
Building Blocks of 2d Patches



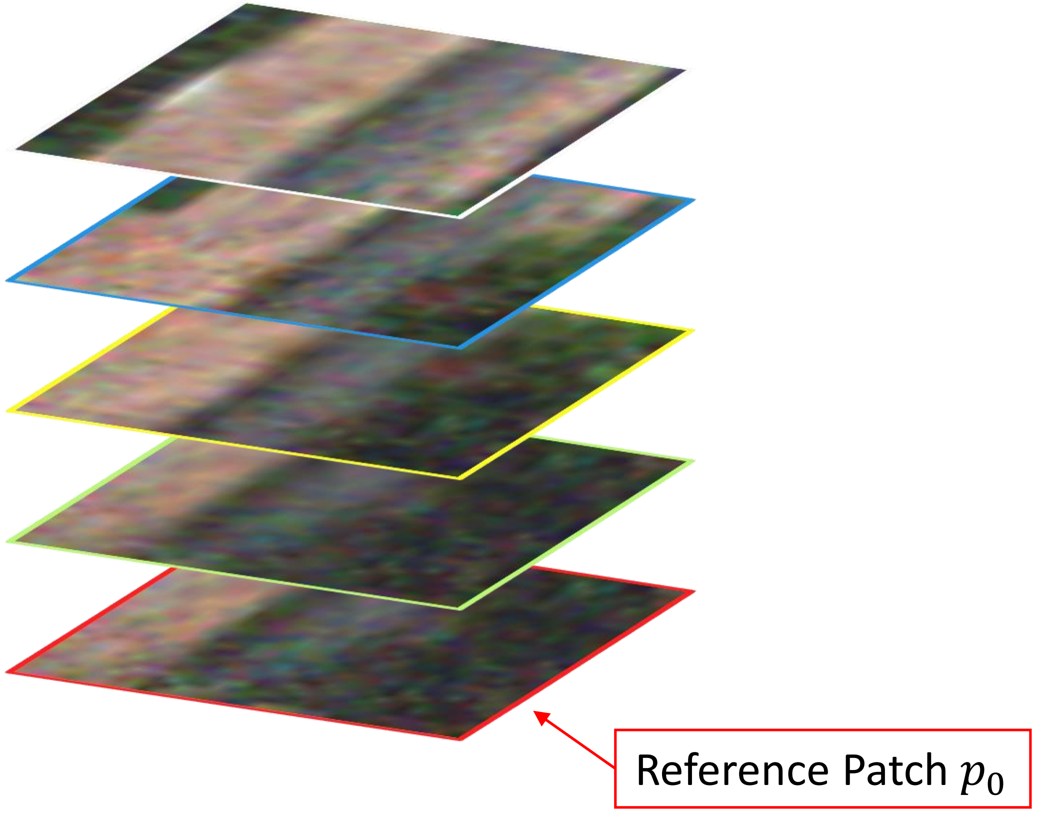


Reference Patch p_0

Building Blocks of 2d Patches

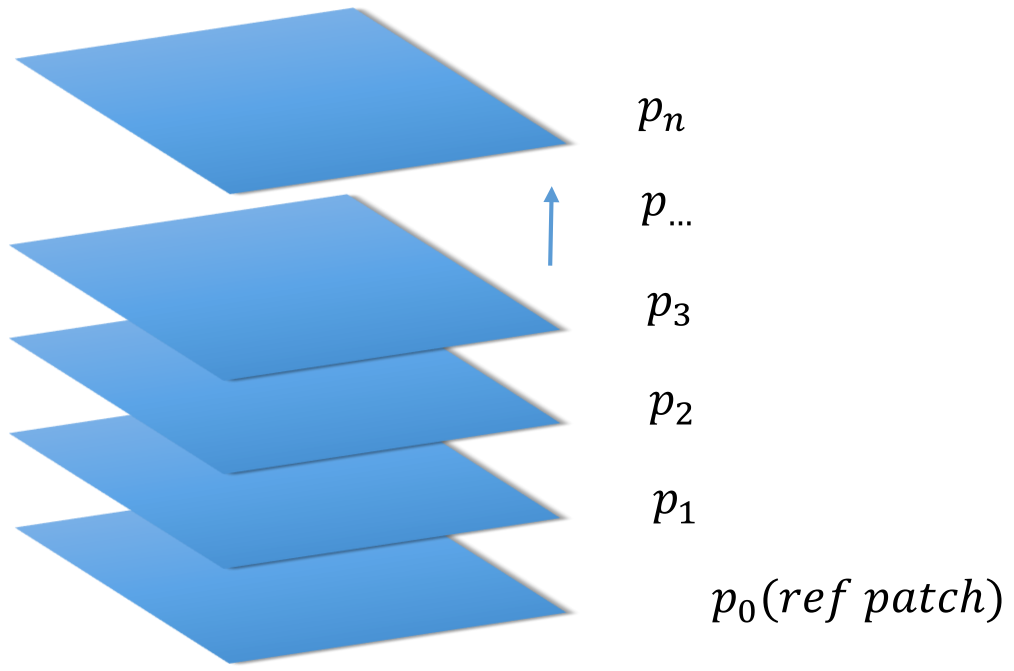


Building Blocks of 2d Patches



Building Blocks of 2d Patches

3D Block of Similar Patches



Two variants of NL-Means

Pixel-Wise

- Use *only pixels at the* ***centre*** *of each patch* for final estimate

Denoised pixel at location p

Noisy pixel at location q

$$\hat{u}(p) = \frac{1}{C(p)} \sum_{q \in B(p,r)} \tilde{u}(q)\, w(p,q)$$

Two variants of NL-Means

<u>Pixel-Wise</u>

- Use *only pixels at the <u>centre</u> of each patch* for final estimate

Noisy pixel at location q

Denoised pixel at location p

$$\hat{u}(p) = \frac{1}{C(p)} \sum_{q \in B(p,r)} \tilde{u}(q)\, w(p,q)$$

Neighbourhood centred at p (size $(2r+1)^2$)

Two variants of NL-Means

Pixel-Wise

- Use *only pixels at the* <u>*centre*</u> *of each patch* for final estimate

Denoised pixel at location p

Noisy pixel at location q

$$\hat{u}(p) = \frac{1}{C(p)} \sum_{q \in B(p,r)} \tilde{u}(q)\, w(p,q)$$

Neighbourhood centred at p (size $(2r+1)^2$)

Weight between patches centred at p (reference patch) and q

Two variants of NL-Means

<u>Pixel-Wise</u>

- Use *only pixels at the <u>centre</u> of each patch* for final estimate

Noisy pixel at location q

Denoised pixel at location p

$$\hat{u}(p) = \frac{1}{C(p)} \sum_{q \in B(p,r)} \tilde{u}(q)\, w(p,q)$$

Neighbourhood centred at p (size $(2r+1)^2$)

Weight between patches centred at p (reference patch) and q

$$C(p) = \sum_{q \in B(p,r)} w(p,q)$$

Two variants of NL-Means

Patch-Wise

- Use *all pixels of each patch* in the block for final estimate

➔Otherwise exactly the same as pixel-wise! ☺

Two variants of NL-Means

<u>Patch-Wise</u>

- Use *<u>all</u> pixels of each patch* in the block for final estimate

➔Otherwise exactly the same as pixel-wise! ☺

<u>Note:</u>

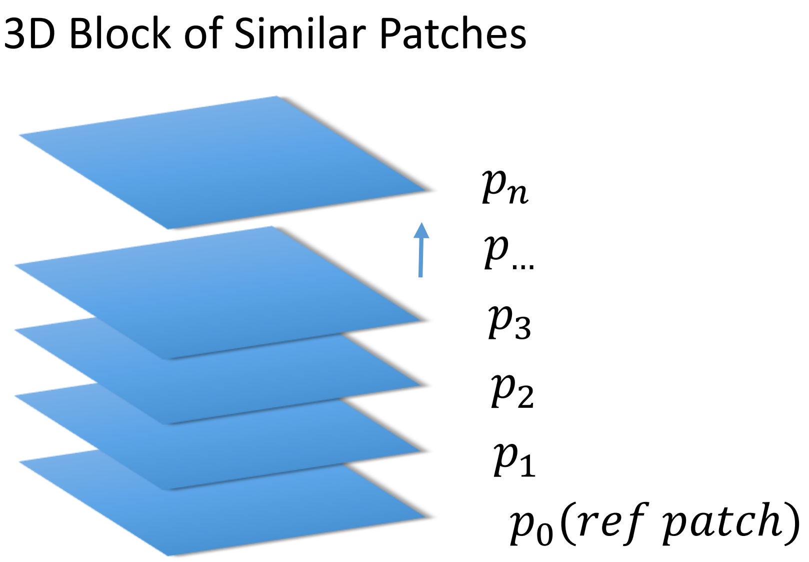
- We now get multiple estimates per pixel ➔ take average
- *Reason:* Patches centred at every pixel overlap

Filtering – Choosing the weights

- Average pixels across all patches in a block?

Filtering – Choosing the weights

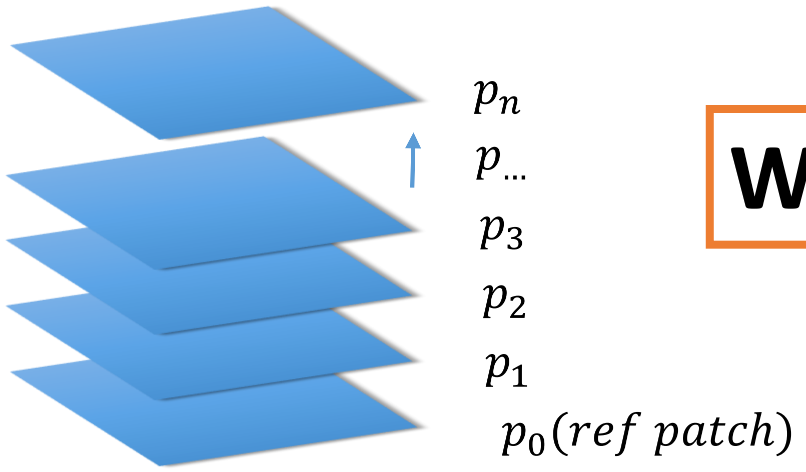
- Average pixels across all patches in a block?
- **Better:** Weight according to similarity to the reference patch



Filtering – Choosing the weights

- Average pixels across all patches in a block?
- **Better:** Weight according to similarity to the reference patch

3D Block of Similar Patches



We need a weight for each patch

Filtering – Weighting Function

- We want
  - Similar patches to count more
  - Allow more dissimilar patches to count in each block for strong noise
  - (Impose *additional* constraint on which patches to keep)

Filtering – Weighting Function

- We want
  - Similar patches to count more
  - Allow more dissimilar patches to count in each block for strong noise
  - (Impose *additional* constraint on which patches to keep)

- So far, we have assumed using *all* patches in a local neighbourhood
- However, we could exclude dissimilar patches from the start
  ➔ We keep only a *subset* of patches in $B(p, r)$
- *Note: Weighting function already does this to some degree (can be 0)*

Filtering – Weighting Function

- We want
  - Similar patches to count more
  - Allow more dissimilar patches for strong noise
  - (Impose *additional* constraint on which patches to keep)

Squared Patch Distance (SSD)

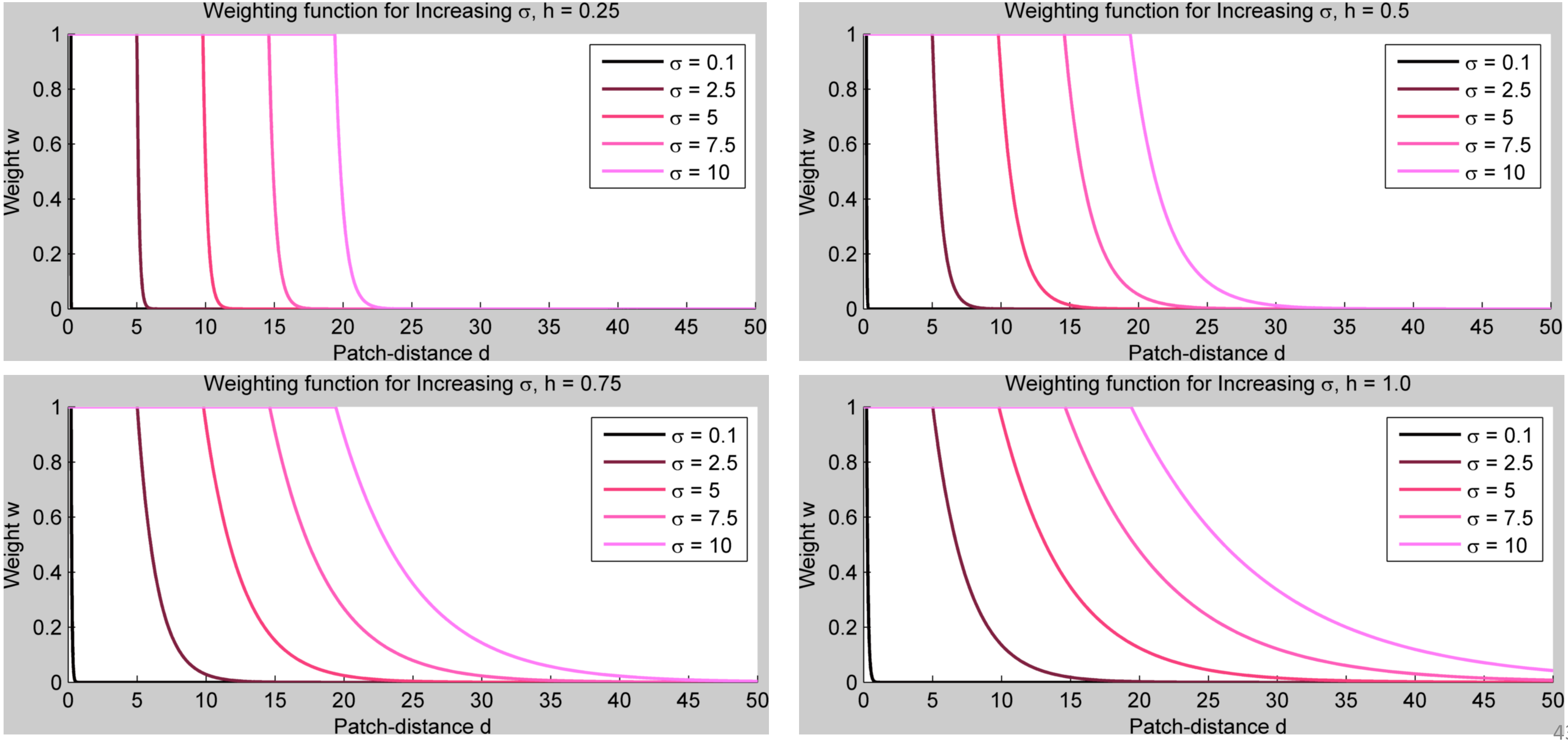
Noise Standard Deviation

$$\boxed{w(p,q) = e^{-\frac{\max(d^2 - 2\sigma^2, 0)}{h^2}}}$$

Weight between patches centred at p and q

Decay Parameter

Filtering – Weighting Function



Problems with Non-Local Methods

- Complexity (Can be quadratic or worse w.r.t. image size)
  - ➔ Takes a long time to run
  - ➔ In the real world, we deal with 2k, 4k, 6k Images

Problems with Non-Local Methods

- Complexity (Can be quadratic or worse w.r.t. image size)
  - ➔ Takes a long time to run
  - ➔ In the real world, we deal with 2k, 4k, 6k Images
- Not all images are highly self-similar everywhere

Problems with Non-Local Methods

- Complexity (Can be quadratic or worse w.r.t. image size)
  - ➔ Takes a long time to run
  - ➔ In the real world, we deal with 2k, 4k, 6k Images

- Not all images are highly self-similar everywhere

- Search **Database** of similar images
- Search **other frames** (if video)

Efficient Template Matching

The Problem

- Calculating SSD for every patch centred at each pixel with every other patch in local neighbourhood is *VERY* expensive

$$\boldsymbol{O(MNn^2)}$$

M = # offsets
N = # pixels in the image
n = patch size

Grow **quadratically** for square images / search regions as well!

The Problem

- Calculating SSD for every patch centred at each pixel with every other patch in local neighbourhood is *VERY* expensive

$$\boldsymbol{O}\left(\boldsymbol{MNn^2}\right)$$

$$\mathrm{M} = \#\ \text{offsets}$$
$$\mathrm{N} = \#\ \text{pixels in the image}$$
$$\mathrm{n} = \text{patch size}$$

Grow **quadratically** for square images / search regions as well!

Integral Images can reduce number of arithmetic operations required to:

$$\boldsymbol{O}(\boldsymbol{MN})$$

What is an Integral Image?

- The **Integral Image** $\boldsymbol{I_{int}}$ *for an Image* I stores at every pixel location,
  - The sum of all pixels in I to the left *of the current location*
  - And all pixels in I above *of the current location*
  - Including the pixel at the current location in I.

What is an Integral Image?

- The **Integral Image** $\boldsymbol{I_{int}}$ *for an Image* I stores at every pixel location,
  - The sum of all pixels in I to the left *of the current location*
  - And all pixels in I above *of the current location*
  - Including the pixel at the current location in I.

Integral Image location: x, y

$$I_{int}(x, y) = \sum_{p \leq x, q \leq y} I(p, q)$$

Image location: p, q

What is an Integral Image?

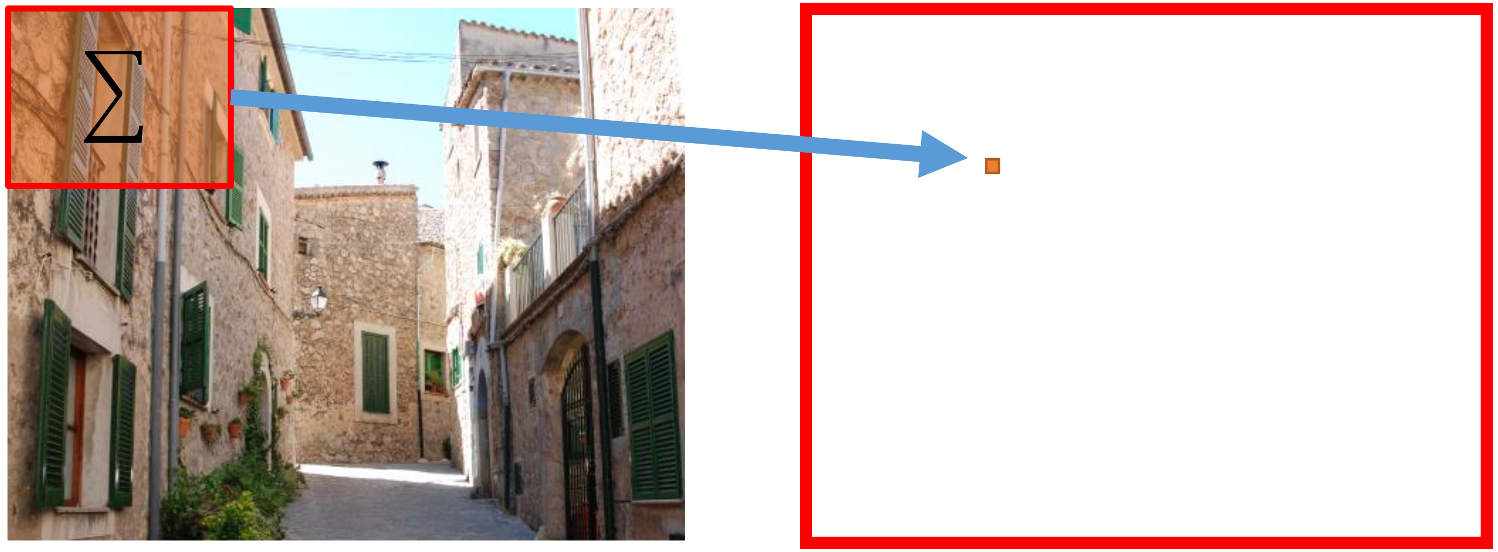
- The **Integral Image** $\boldsymbol{I_{int}}$ *for an Image* I stores at every pixel location,
  - The sum of all pixels in I to the left *of the current location*
  - And all pixels in I above *of the current location*
  - Including the pixel at the current location in I.

$\boldsymbol{I}$

$\boldsymbol{I_{int}}$

What is an Integral Image?

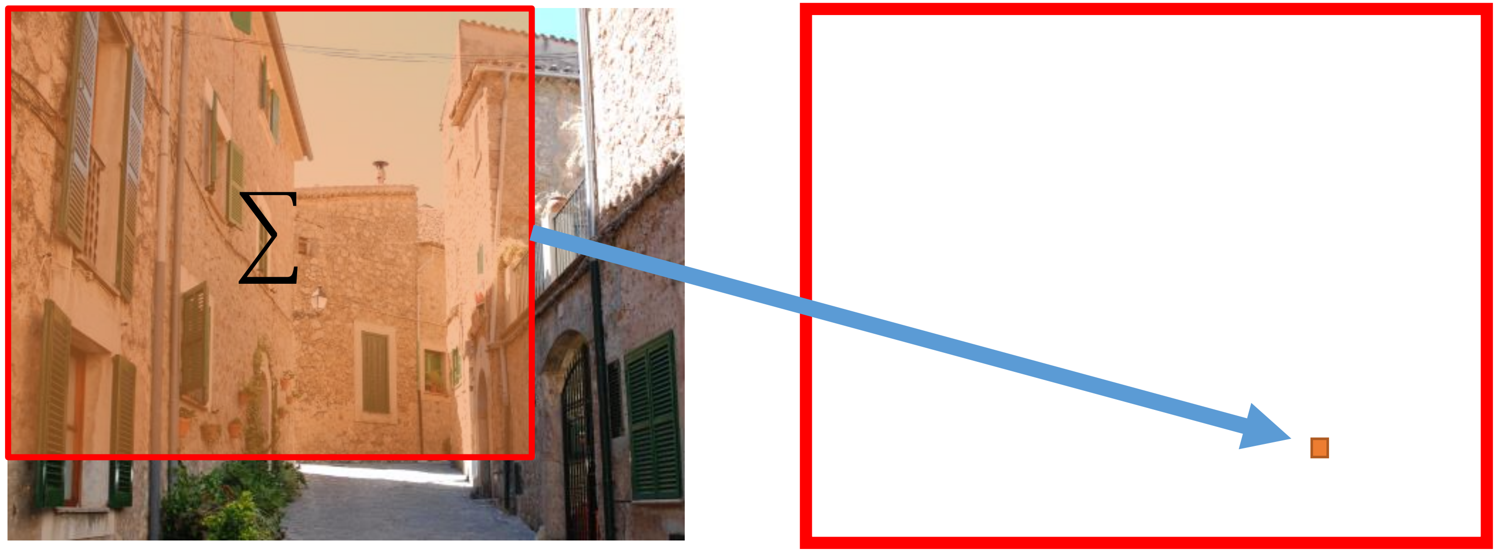
- The **Integral Image** $\boldsymbol{I_{int}}$ *for an Image* I stores at every pixel location,
  - The sum of all pixels in I to the left *of the current location*
  - And all pixels in I above *of the current location*
  - Including the pixel at the current location in I.



$\boldsymbol{I}$ $\boldsymbol{I_{int}}$

What is an Integral Image?

- The **Integral Image** $\boldsymbol{I_{int}}$ *for an Image* I stores at every pixel location,
  - The sum of all pixels in I to the left *of the current location*
  - And all pixels in I above *of the current location*
  - Including the pixel at the current location in I.



$\boldsymbol{I}$ $\boldsymbol{I_{int}}$

Computing it efficiently (serially)

- Remember:

Integral Image location: x, y

Image location: p, q

$$I_{int}(x,y) = \sum_{p \le x, q \le y} I(p,q)$$

Computing it efficiently (serially)

- Remember:

$$I_{int}(x, y) = \sum_{p \leq x, q \leq y} I(p, q)$$

- Use recurrence relation:

Cumulative sum (column-wise)

$$s(x, y) = s(x, y - 1) + I(x, y)$$

$$I_{int}(x, y) = I_{int}(x - 1, y) + s(x, y)$$

Computing it efficiently (serially)

- Use recurrence relation:

Cumulative sum (column-wise)

$$s(x, y) = s(x, y-1) + I(x, y)$$

$$I_{int}(x, y) = I_{int}(x-1, y) + s(x, y)$$

Note Starting Point:

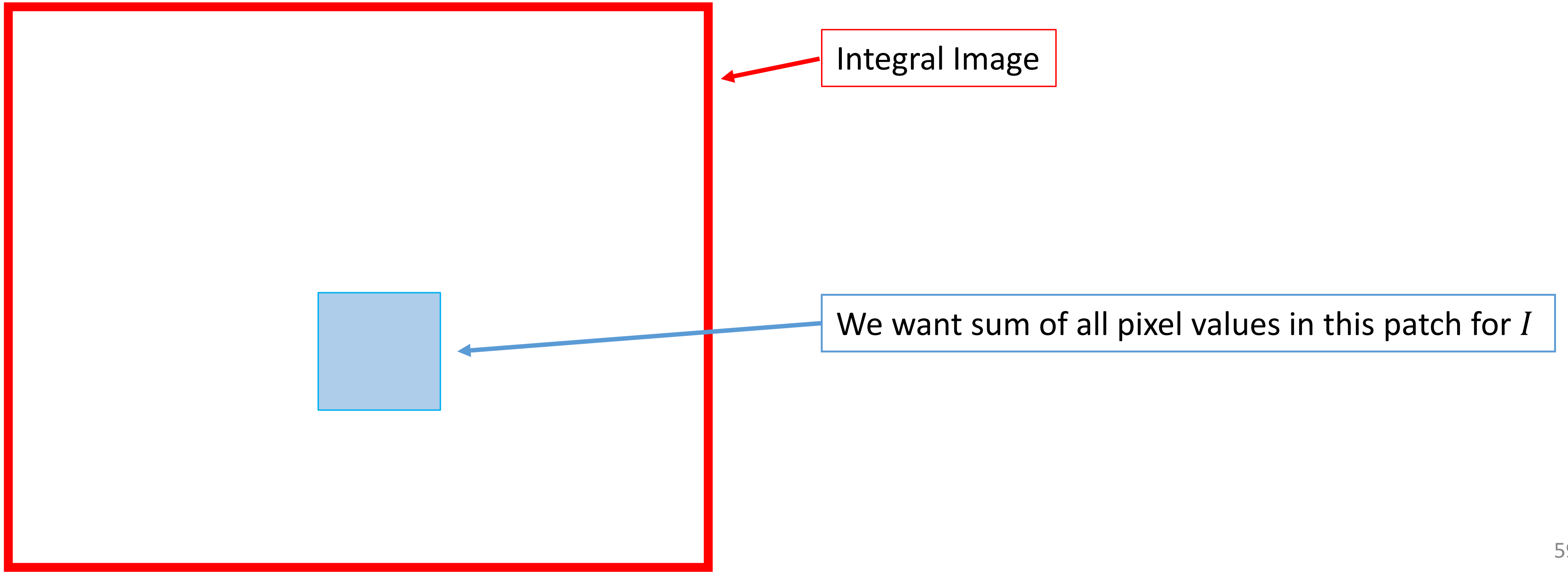
$$s(x, -1) = I_{int}(-1, y) = 0$$

How do we use the integral image?

- We can calculate the sum of all pixels in a patch in **constant time, regardless of patch size**!

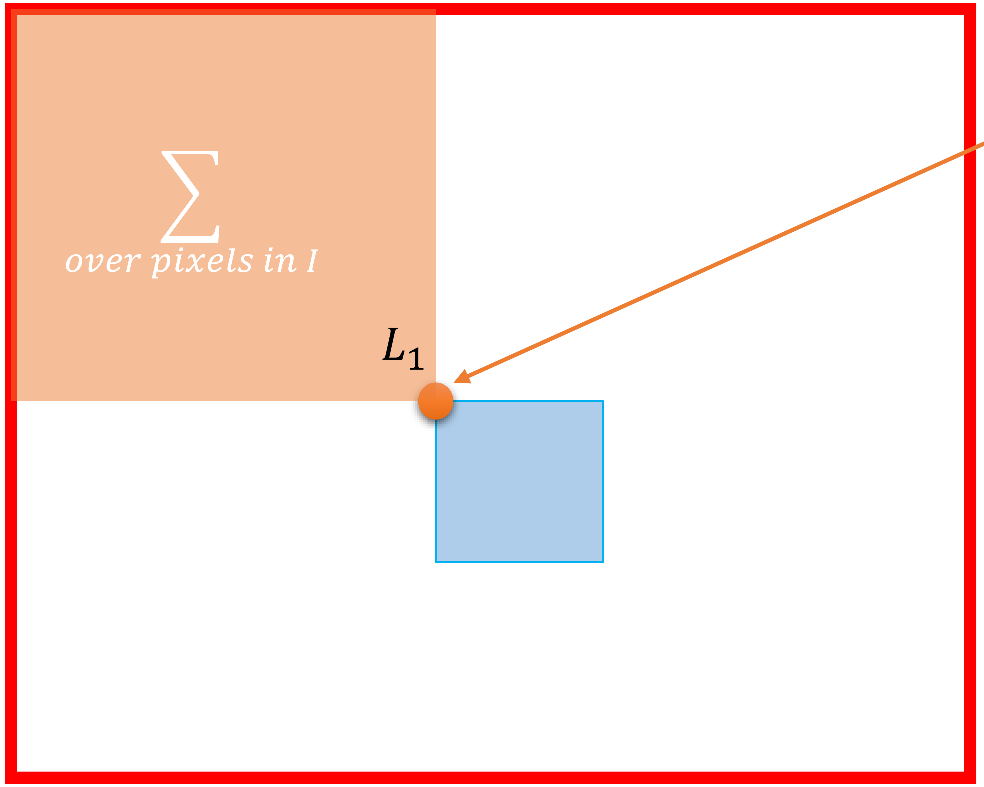
How do we use the integral image?

- We can calculate the sum of all pixels in a patch in **constant time, regardless of patch size**!



How do we use the integral image?

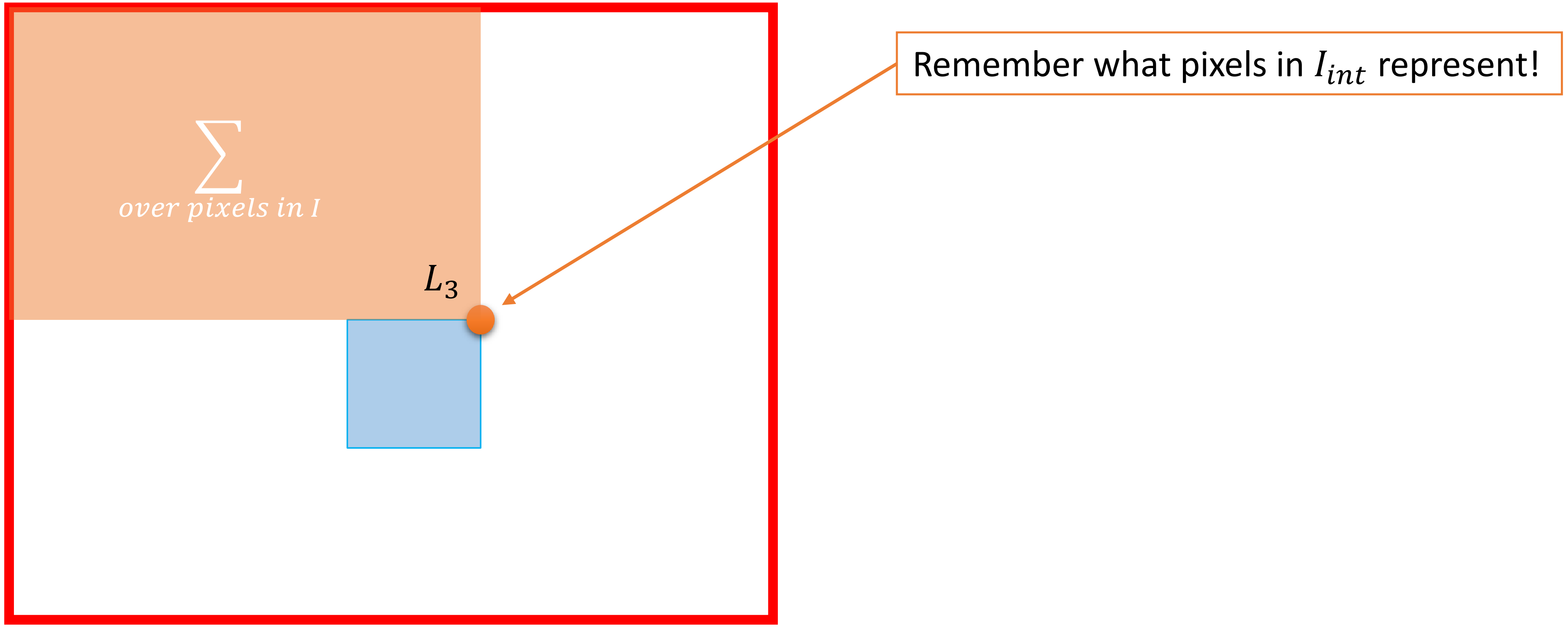
- We can calculate the sum of all pixels in a patch in **constant time, regardless of patch size**!



Remember what pixels in I_{int} represent!

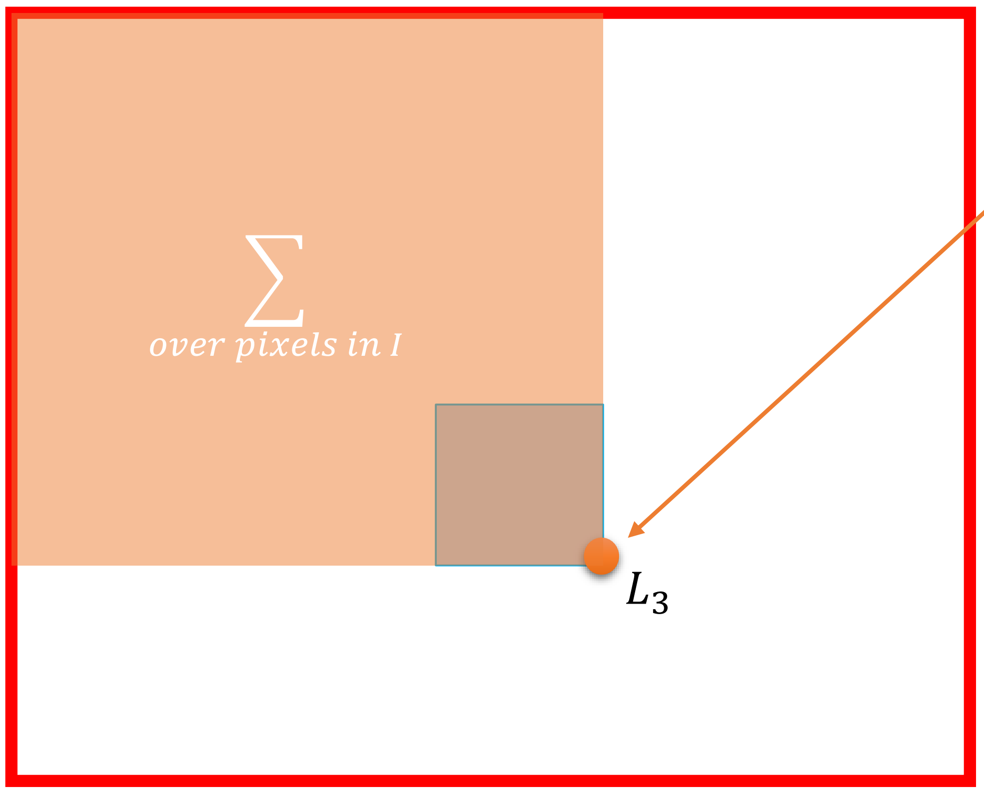
How do we use the integral image?

- We can calculate the sum of all pixels in a patch in **constant time, regardless of patch size**!



How do we use the integral image?

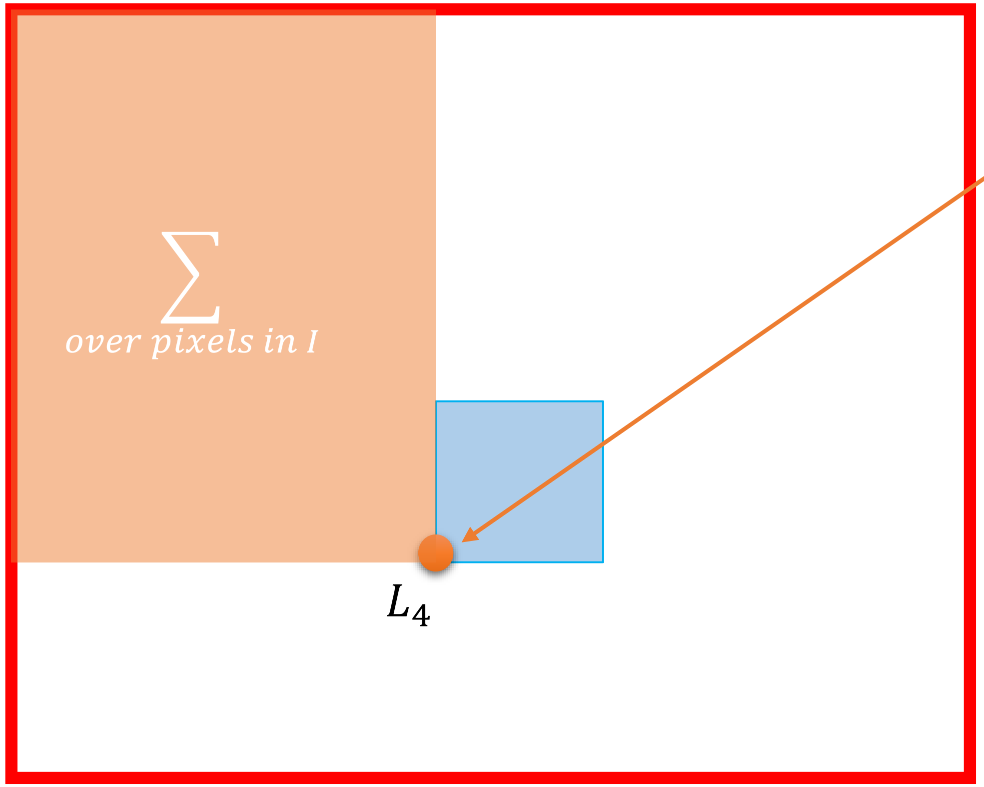
- We can calculate the sum of all pixels in a patch in **constant time, regardless of patch size**!



Remember what pixels in I_{int} represent!

How do we use the integral image?

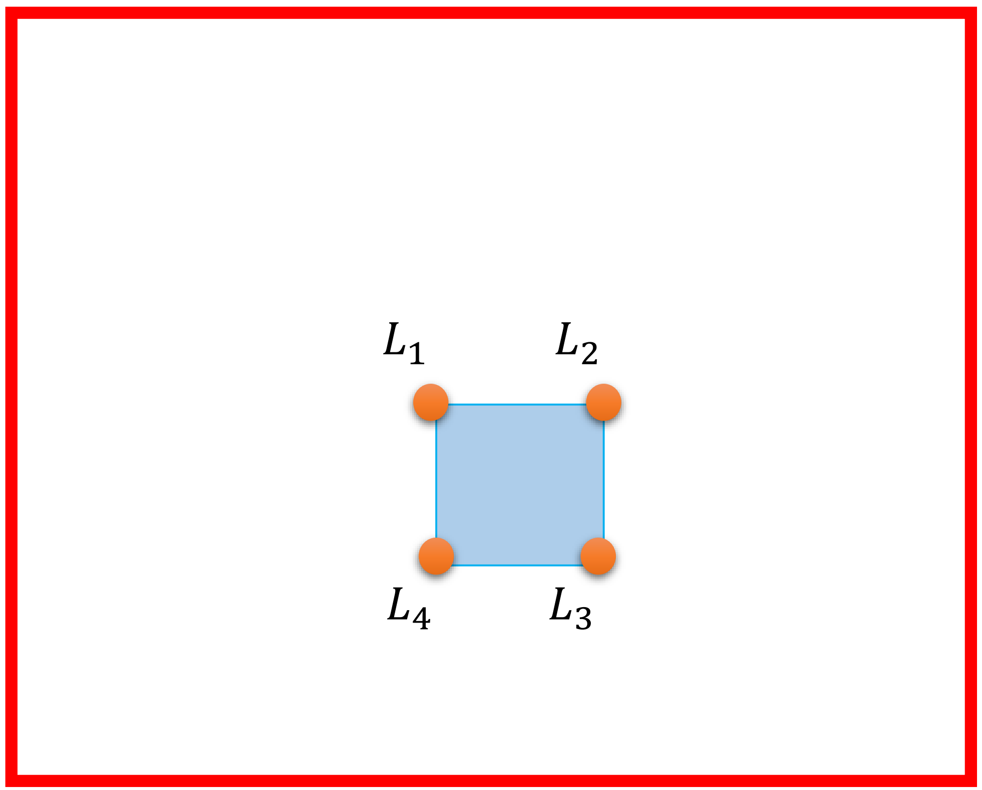
- We can calculate the sum of all pixels in a patch in **constant time, regardless of patch size**!



Remember what pixels in I_{int} represent!

How do we use the integral image?

- We can calculate the sum of all pixels in a patch in **constant time, regardless of patch size**!



Sum of patch is:

$$\boldsymbol{L_3 - L_2 - L_4 + L_1}$$

Necessary as pixels used to calculate $L_1 \subseteq L_2$ AND $L_1 \subseteq L_4$

How does this help with template matching?

- We can build an integral image of a *difference image*

Example: *Calculate SSD for offset (-10, -15)*

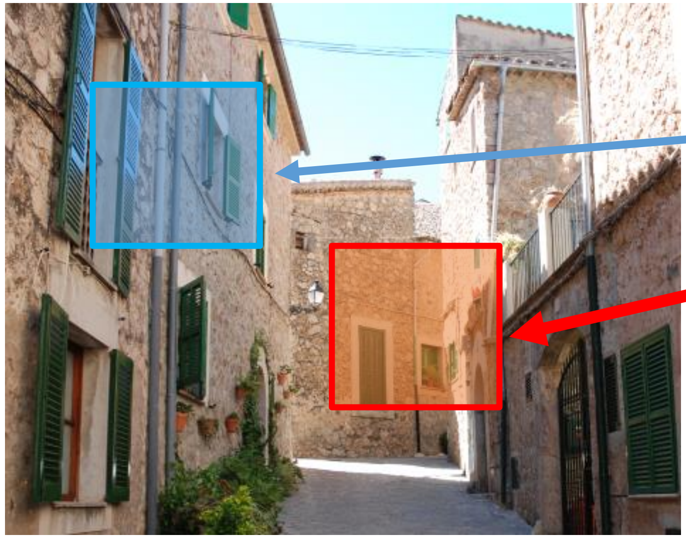
I.e. for a patch shifted -10 pixels down and -15 pixels to the right

How does this help with template matching?

- We can build an integral image of a *difference image*

<u>Example:</u> *Calculate SSD for offset (-10, -15)*

I.e. for a patch shifted -10 pixels down and -15 pixels to the right



Between this patch and this patch

How does this help with template matching?

- We can build an integral image of a *difference image*

<u>Example:</u> *Calculate SSD for offset (-10, -15)*

Between this patch and this patch

How does this help with template matching?

- We can build an integral image of a *difference image*

Example: *Calculate SSD for offset (-10, -15)*

Between this patch and this patch

How does this help with template matching?

- We can build an integral image of a *difference image*

<u>Example:</u> *Calculate SSD for offset (-10, -15)*

1. Calculate **<u>per-pixel</u> squared difference** *for the entire image, offset with itself*

➔ Difference Image

How does this help with template matching?

- We can build an integral image of a *difference image*

<u>Example:</u> *Calculate SSD for offset (-10, -15)*

1. Calculate **<u>per-pixel</u> squared difference** *for the entire image, offset with itself*

➔ Difference Image

2. Calculate Integral Image of the difference image

➔ We can now read SSD *for all patches* ***for this offset*** in constant time!

How does this help with template matching?

- We can build an integral image of a *difference image*

<u>Example:</u> *Calculate SSD for offset (-10, -15)*

1. Calculate **<u>per-pixel</u> squared difference** *for the entire image, offset with itself*

➔ Difference Image

2. Calculate Integral Image of the difference image

➔ We can now read SSD *for all patches* ***for this offset*** in constant time!

<u>Note:</u> There are as many offset as there are pixels in the search window!

Problems

- Memory Overhead?
- Parallel Computation of the Integral Image?
- No benefit for smaller patch-sizes
- Some Distance Metrics not supported

Results

Original

Results

Noisy

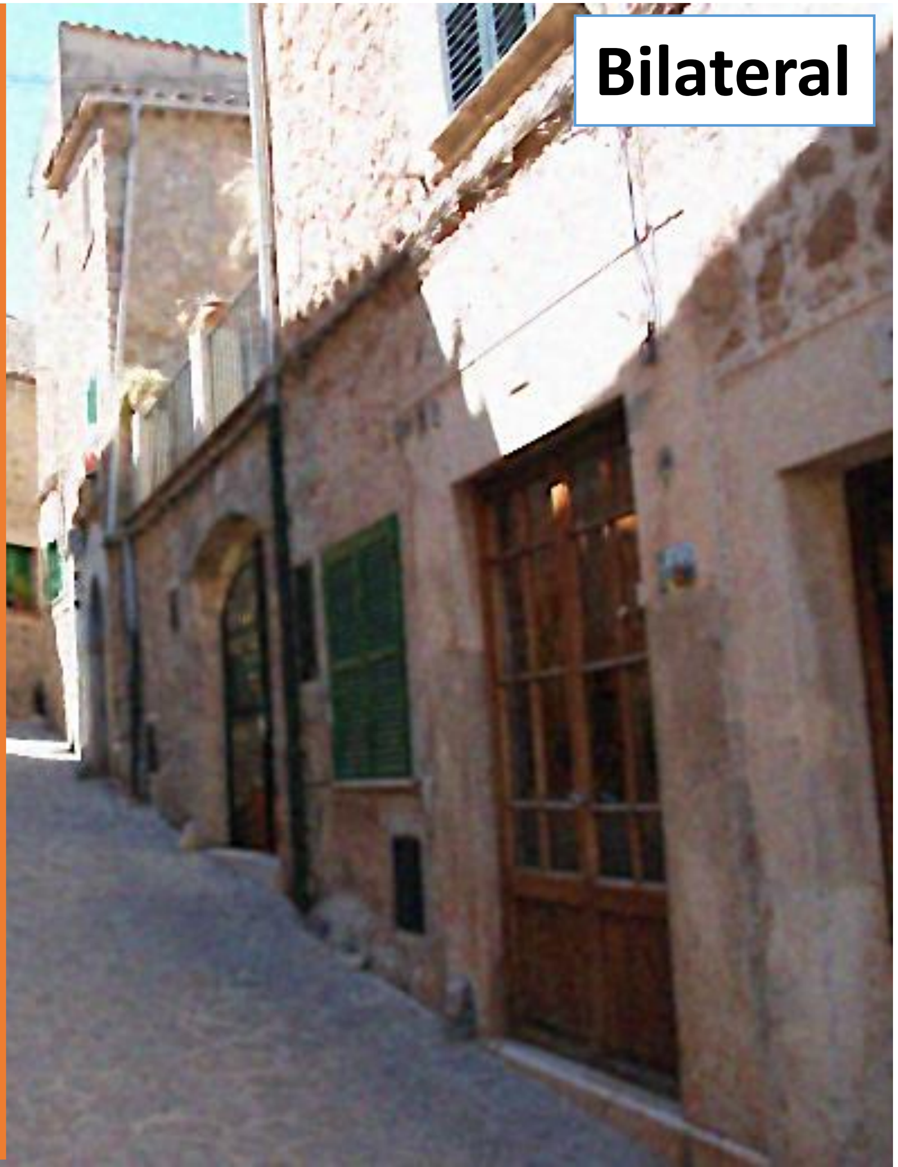
Results

Denoised

Results

Error

Results



Closing Remarks

- We have just implemented & understood the NL-Means filter:

$$\boldsymbol{NL}\big(\boldsymbol{u}(\boldsymbol{p})\big) = \frac{\boldsymbol{1}}{\boldsymbol{C}(\boldsymbol{p})} \int \boldsymbol{f}\Big(\boldsymbol{d}\big(\boldsymbol{B}(\boldsymbol{p}), \boldsymbol{B}(\boldsymbol{q})\big)\boldsymbol{u}(\boldsymbol{q})\Big)\, \boldsymbol{dq}$$

- Many other more complicated non-local methods:
  - BM3D/BM3D-SAPCA,
  - Non-Local Bayes,
  - PLOW, etc…

References

- Antoni Buades, Bartomeu Coll, and Jean-Michel Morel, *Non-Local Means Denoising*, Image Processing On Line, 1 (2011).http://dx.doi.org/10.5201/ipol.2011.bcm_nlm
- Gabriele Facciolo, Nicolas Limare, and Enric Meinhardt-Llopis, *Integral Images for Block Matching*, Image Processing On Line,4 (2014), pp. 344–369. http://dx.doi.org/10.5201/ipol.2014.57