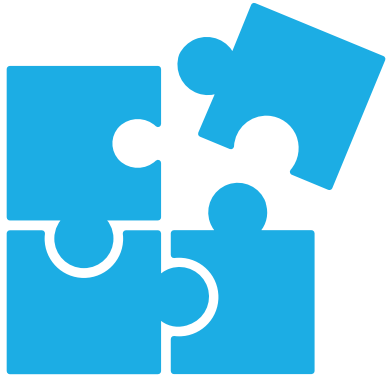
School of Computer Science



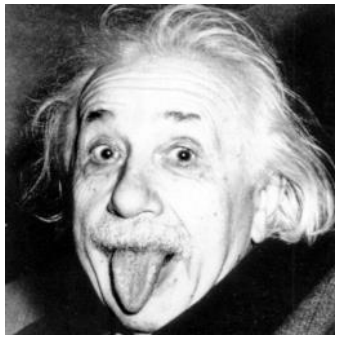
Backtracking

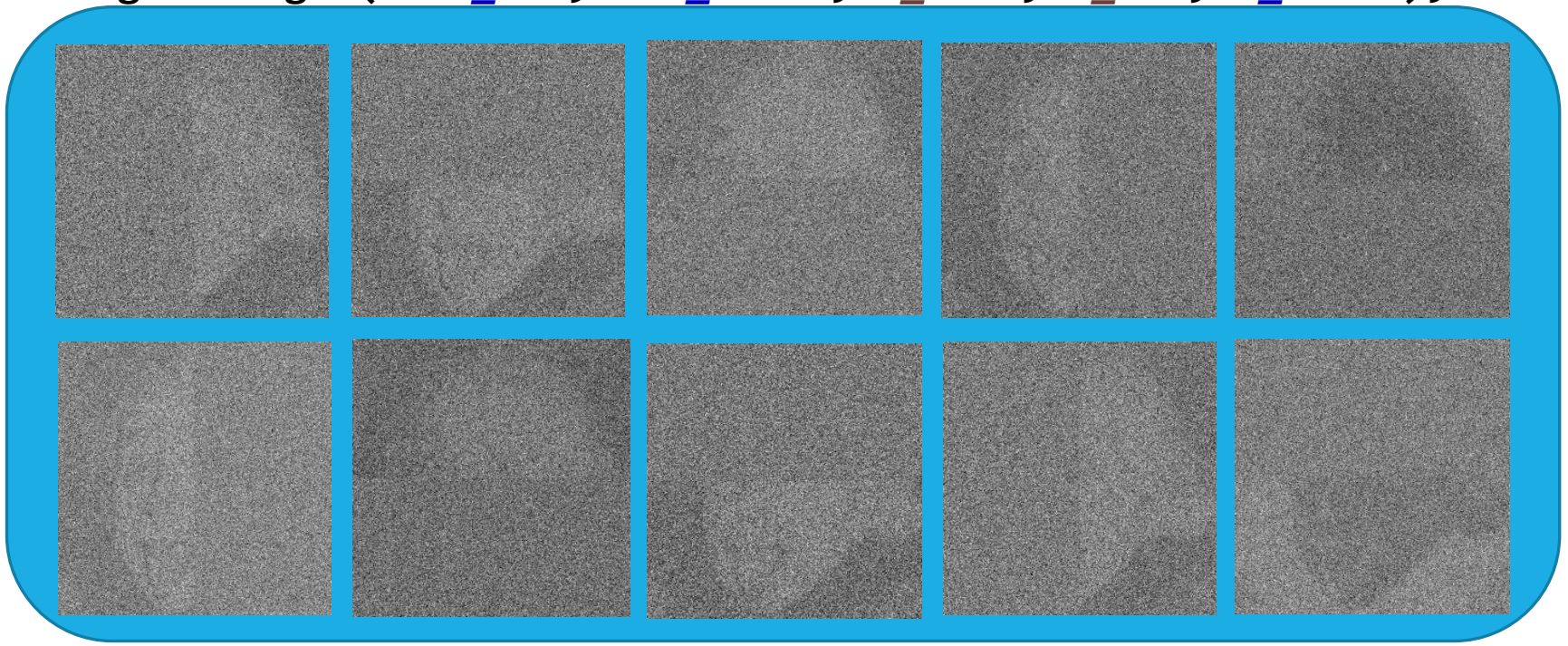
ALGORITHMICS

Generation of images

- We will generate synthetically the image dataset
  - For example with `N_IMGS` = 10 and `PERCENTAGE_MISALIGNED` = 25 we will have 8 “good” pictures and 2 “bad” pictures

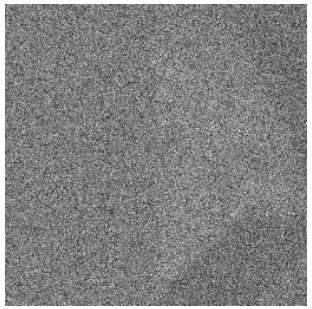
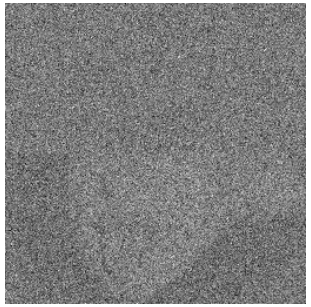
```
img_avger = new ImageAverager(REAL_IMG, BAD_IMAGE, n_real, n_bad, S_NOISE);
```

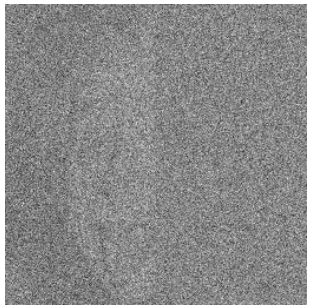
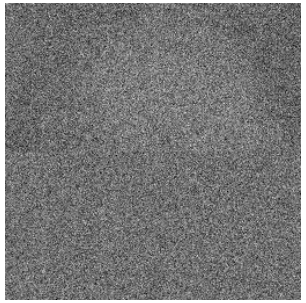




Good images

- We simulate a process in which we cannot get complete images

left   up

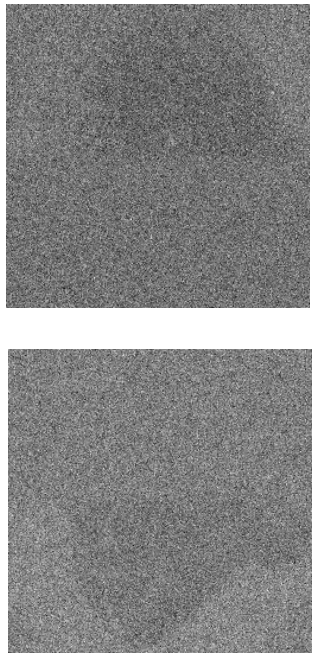
right   down

```
hold_img = new Image(this.width, this.height);
hold_img.addSignal(this.real_img);
hold_img.suppressRegion(region);
hold_img.addNoise(s_noise); //additive Gaussian
                                noise
```

Additive white Gaussian noise is a basic noise model used in computer science to mimic the effect of many random processes that occur in nature

Bad images

- We simulate a process in which we got a image that is not correct



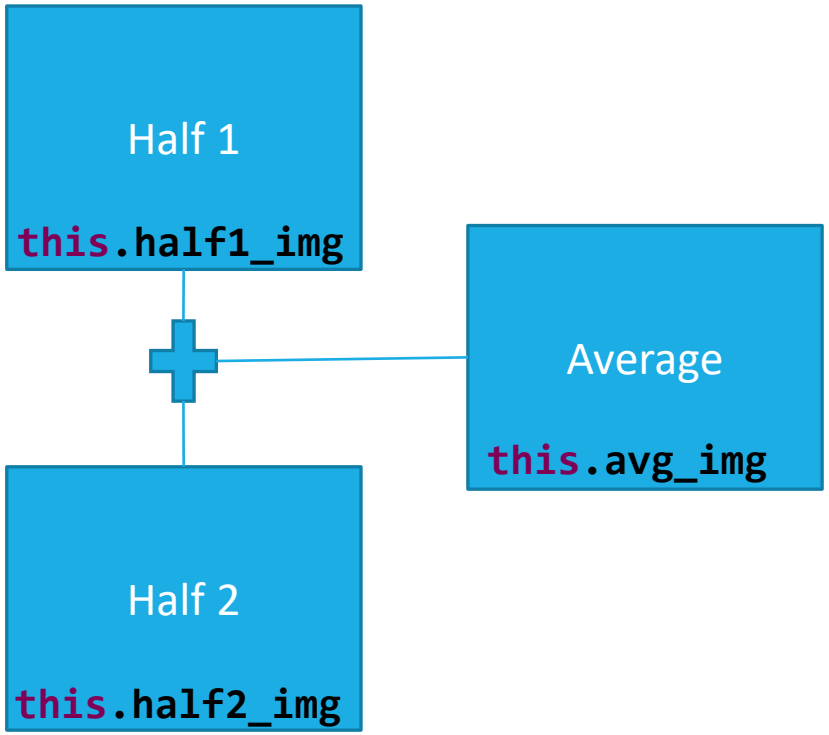
```
hold_img = new Image(this.width, this.height);
hold_img.addSignal(this.bad_img);
hold_img.invertSignal(); //corrupt the image
hold_img.suppressRegion(region);
hold_img.addNoise(s_noise); //additive Gaussian noise
```

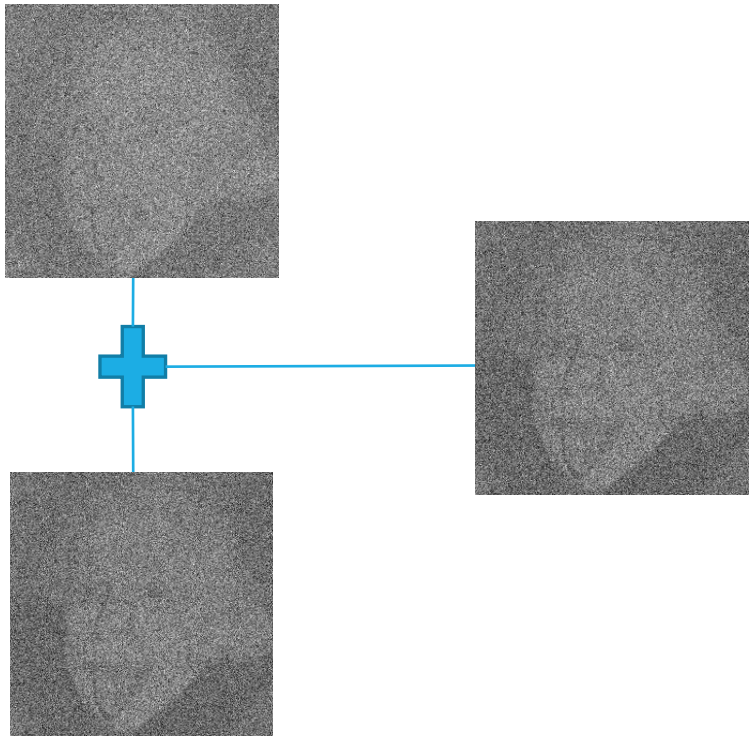
Our goal

- We need to recreate the original image considering that we don't know which one is the original image
- We also don't know which parts of the generated images are good or bad
- We only know that the percentage of good images is greater than the percentage of bad images
  - So, if we **compare the similarity** between groups of images, we will find more matches between good ones than between bad ones

Idea

- We create two groups of images and we calculate the similarity





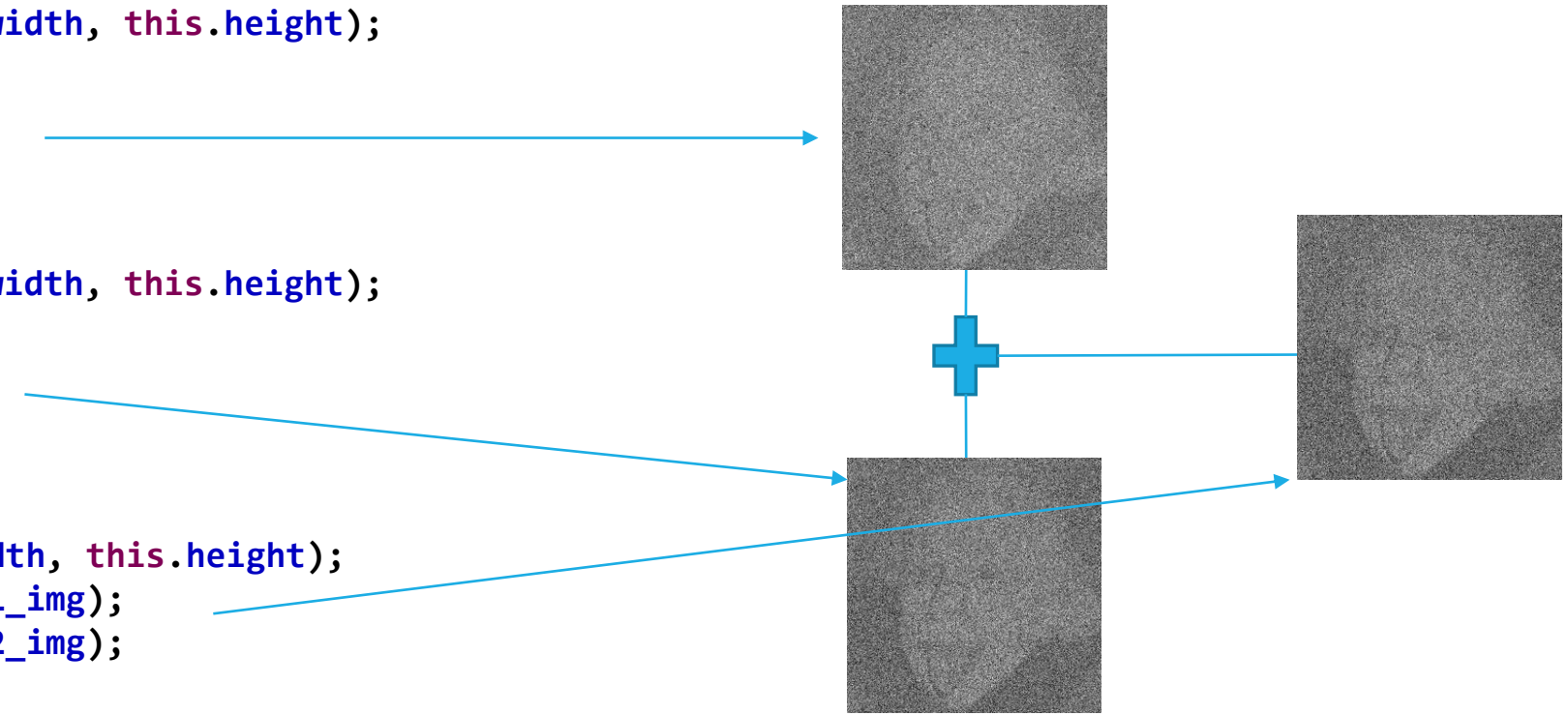
How to combine images?

- To create **groups of images** or to have the **average between two images** it is possible to:

```
this.half1_img = new Image(this.width, this.height);
this.half1_img.addSignal(image1);
this.half1_img.addSignal(image3);
this.half1_img.addSignal(image4);
this.half1_img.addSignal(image5);

this.half2_img = new Image(this.width, this.height);
this.half2_img.addSignal(image2);
this.half2_img.addSignal(image6);
this.half2_img.addSignal(image7);
this.half2_img.addSignal(image8);

this.avg_img = new Image(this.width, this.height);
this.avg_img.addSignal(this.half1_img);
this.avg_img.addSignal(this.half2_img);
```



Zero Mean Cross-Correlation

- **Similarity** between **two** gray valued **images**

$$ZNCC = \frac{\sum_{x,y}(I_1(x,y) - \mu_1)(I_2(x,y) - \mu_2)}{N \cdot \sigma_1 \cdot \sigma_2}$$

- Where:
  - I_1 and I_2 are two images of the same size
  - I(i, j) is the value of the pixel in the position i and j of any of the images
  - N is the number of pixels of any of the images
  - μ_1 is the mean value of pixels for I_1 and μ_2 is the mean value of pixels for I_2
  - σ_1 is the standard deviation of the pixels for I_1 and σ_2 is the standard deviation of the pixels for I_2

- Values we can obtain:
  - If one of the pictures has all the values to 0 → ZNCC = 0
  - If one of the pictures is the negative of the other → ZNCC = -1
  - If both images are exactly equals → ZNCC = 1

How can we express the solution?

- Like a vector $X = \{x_1, x_2, x_3, \ldots, x_n\}$ with the highest ZNCC

- Values can be:
  - If we don't use the image → $X_i = 0$
  - If we use the image in the first half → $X_i = 1$
  - If we use the image in the second half → $X_i = 2$

Greedy algorithm

- Method with a loop that randomly assign each of the images to different groups:
  - The image can be part of half 1
  - The image can be part of half 2
  - The image can be ignored

- `N_TRIES_GREEDY` is the number of times we execute that loop to make an attempt

```
System.out.print("TESTING GREEDY:\n");
img_avger.splitSubsetsGreedy(N_TRIES_GREEDY);
System.out.printf("  -ZNCC: %f\n",  img_avger.zncc());
System.out.printf("  -Counter: %d\n",  img_avger.getCounter());
img_avger.saveResults(OUT_DIR_G);
```

Bactracking algorithm

- Method that tries all the alternatives including an image in half 1, in half 2 and in no group

```
System.out.print("TESTING BACKTRACKING:\n");
img_avger.splitSubsetsBacktracking();
System.out.printf("  -ZNCC: %f\n",  img_avger.zncc());
System.out.printf("  -Counter: %d\n",  img_avger.getCounter());
img_avger.saveResults(OUT_DIR_B);
```

Bactracking algorithm with balancing condition

- Method that tries all the alternatives including an image in half 1, in half 2 and in no group
- `MAX_UNBALANCING` is the maximum difference that can be between the number of images in half 1 and in half 2

```
System.out.print("TESTING BACKTRACKING BALANCED:\n");
img_avger.splitSubsetsBacktracking(MAX_UNBALANCING);
System.out.printf("  -ZNCC: %f\n",  img_avger.zncc());
System.out.printf("  -Counter: %d\n",  img_avger.getCounter());
img_avger.saveResults(OUT_DIR_B);
```