ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
NGÀNH MÁY TÍNH VÀ KHOA HỌC THÔNG TIN


# ỨNG DỤNG XỬ LÝ SONG SONG TRONG NHẬN DIỆN ẢNH TRÙNG LẶP

**[ NHÓM 6 ]**

**Giảng viên hướng dẫn** : **GV. Nguyễn Hải Vinh**

**Sinh viên** : **Lê Huy Tấn Hoàng – 20001551**
**Lưu Quang Huy – 20001556**
**Nguyễn Quang Huy – 20001555**
**Phạm Minh Đức – 20001541**

**Môn học** : **Tính toán song song**

*Hà Nội, ngày 5 tháng 5 năm 2023*

# MỤC LỤC

# I. Giới thiệu khái quát

## *1. Đặt vấn đề*

Khi việc sử dụng di động ngày càng phổ biến, các tập tin hình ảnh trùng lặp không cần thiết ngày càng tăng trên thiết bị, trong mọi thư mục của điện thoại. Những hình ảnh trùng lặp chiếm nhiều bộ nhớ của điện thoại và cũng làm giảm tốc độ hoạt động của điện thoại. Sẽ rất khó để tìm và xóa chúng một cách thủ công.

Chắc chắn rằng chúng ta đều có rất nhiều hình ảnh trên máy tính, điện thoại và thậm chí cả lưu trữ đám mây. Vấn đề khi có quá nhiều ảnh là ta sẽ có thể có rất nhiều bức ảnh có nội dung gần hoặc hoàn toàn giống nhau. Mục tiêu của dự án là tạo ra một công cụ tìm kiếm hình ảnh trùng lặp để loại bỏ các bức ảnh tương tự và trùng lặp. Điều này sẽ giúp quản lý không gian một cách hiệu quả.

Vì lý do đó, dự án sẽ bao gồm giải pháp nhận diện hình ảnh trùng lặp bằng cách sử dụng xử lý song song từ đó ta có thể quét toàn bộ điện thoại và tìm thấy các tập tin hình ảnh trùng lặp. Đồng thời có thể tự đưa ra quyết định xem có muốn xóa chúng hay không bằng cách chọn chúng.

Mục đích chính của dự án là tạo ra một nền tảng đơn giản để phát hiện các hình ảnh trùng lặp từ một tập hình ảnh lớn trong thời gian ngắn. Việc thực hiện dự án này giúp người dùng di động giảm thiểu các tập tin hình ảnh trùng lặp không cần thiết trên thiết bị của họ, giúp tăng tốc độ xử lý và tiết kiệm bộ nhớ.

## *2. Tổng quan giải pháp*

Các hình ảnh tự nhiên có cấu trúc phức tạp thì các pixel của chúng có sự liên kết chặt chẽ, đặc biệt khi chúng ở gần nhau về mặt không gian và những sự liên kết này mang thông tin quan trọng về cấu trúc của các đối tượng trong hình ảnh trực quan. Trong báo cáo này, chúng tôi sẽ sử dụng mô hình SSIM để đơn giản hoá cấu trúc phức tạp của những hình ảnh đó.

Vậy SSIM hay Structural Similarity Index Measure là gì? Hiểu đơn giản thì nó là một phương pháp đánh giá độ tương đồng giữa hai hình ảnh. SSIM đo lường sự khác biệt giữa các đặc trưng trên ba tiêu chí của các hình ảnh. SSIM thường được sử dụng trong các ứng dụng

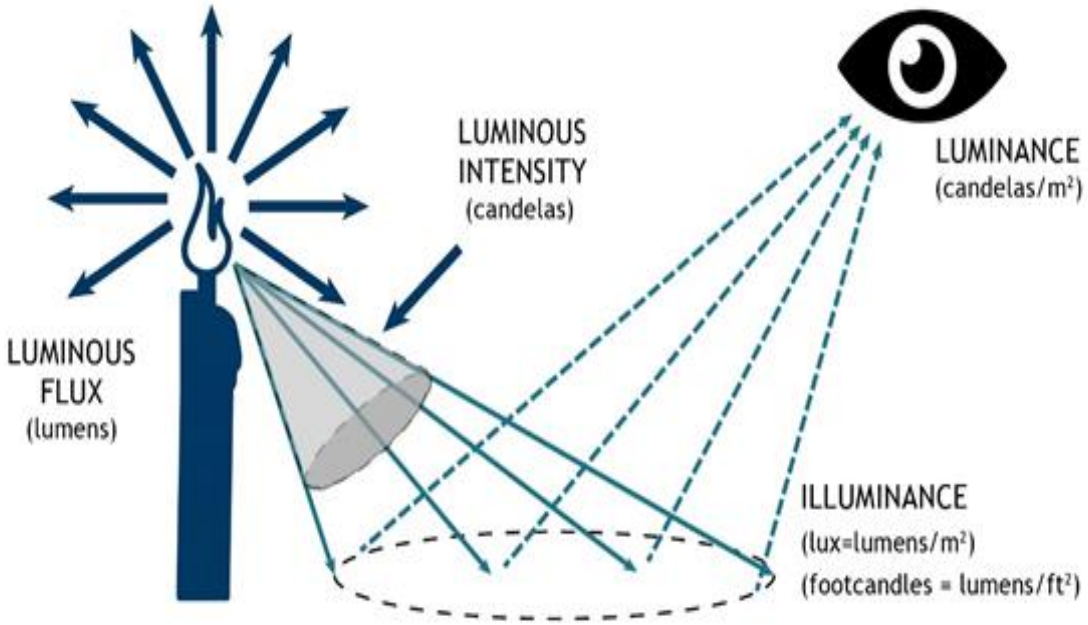
liên quan đến xử lý ảnh, video và nén dữ liệu. Các giá trị SSIM được tính toán trong phạm vi từ 0 đến 1, với 1 là giá trị tối đa cho sự tương đồng hoàn toàn giữa hai hình ảnh.

Ngoài ra để thuận tiện cho công việc tính toán, chúng tôi cũng sử dụng các cấu trúc song song đa luồng và đa lõi. Nhằm hỗ trợ cho việc tính toán được diễn ra nhanh và hiệu quả hơn. Cụ thể ở đây, chúng tôi sẽ sử dụng ngôn ngữ lập trình Python cùng với các thư viện OpenCV, MultiThreading và MultiProcessing để giải quyết bài toán này. Trong đó, OpenCV là một thư viện các chức năng lập trình chủ yếu dành cho thị giác máy tính thời gian thực. MultiThreading và MultiProcessing là các thư viện cung cấp các module và hỗ trợ các thao tác liên quan đến lập trình song song.

# II. Một số kiến thức nền tảng

## *1. Độ chói (Luminance)*

Độ chói là thước đo quang học của cường độ sáng trên một đơn vị diện tích ánh sáng truyền theo một hướng nhất định. Nó mô tả lượng ánh sáng đi qua, phát ra hoặc phản xạ từ một khu vực cụ thể và nằm trong một góc khối nhất định. Đơn vị cho độ chói là cd/m2.

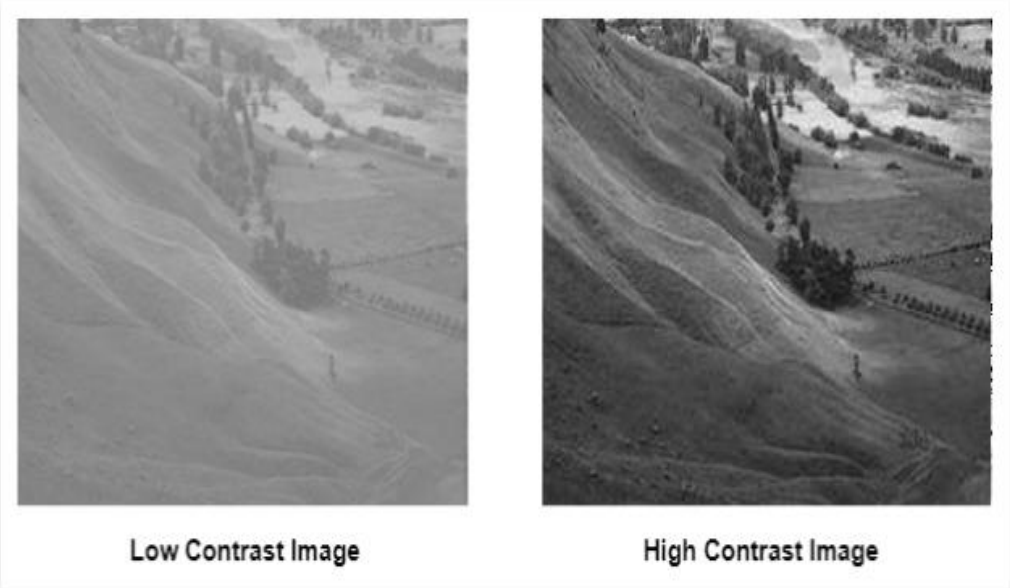


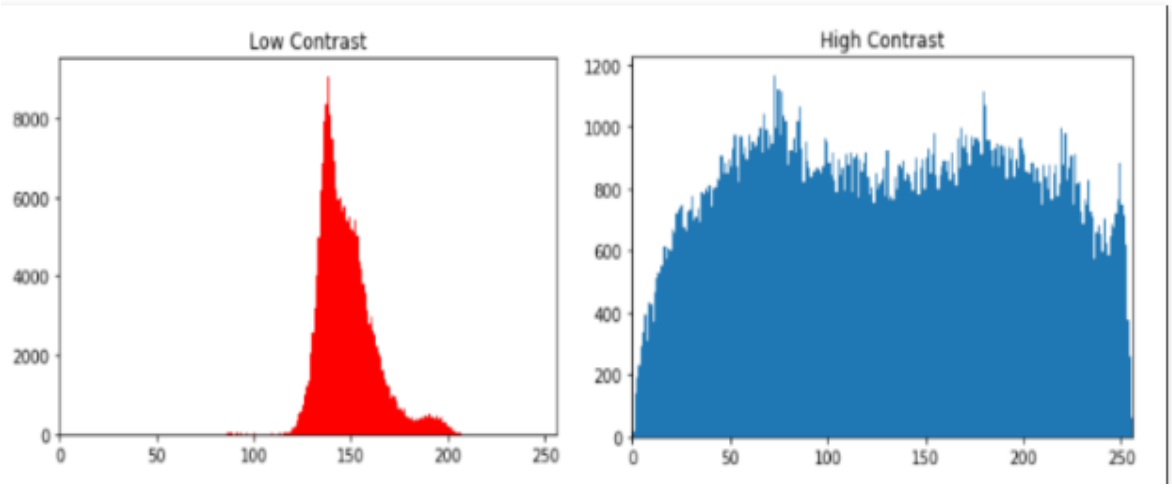

***<u>Hình 1: Minh hoa về độ chói.</u>***

Độ chói thường được sử dụng để mô tả sự phát xạ hoặc phản xạ từ các bề mặt phẳng, khuếch tán. Mức độ chói cho biết mức độ phát sáng mà mắt người có thể phát hiện được khi nhìn vào một bề mặt cụ thể từ một góc nhìn cụ thể.

## *2. Độ tương phản (Contrast)*

Độ tương phản là sự khác biệt về độ chói hoặc màu sắc làm cho một đối tượng có thể phân biệt được với các đối tượng khác trong cùng một khung hình.



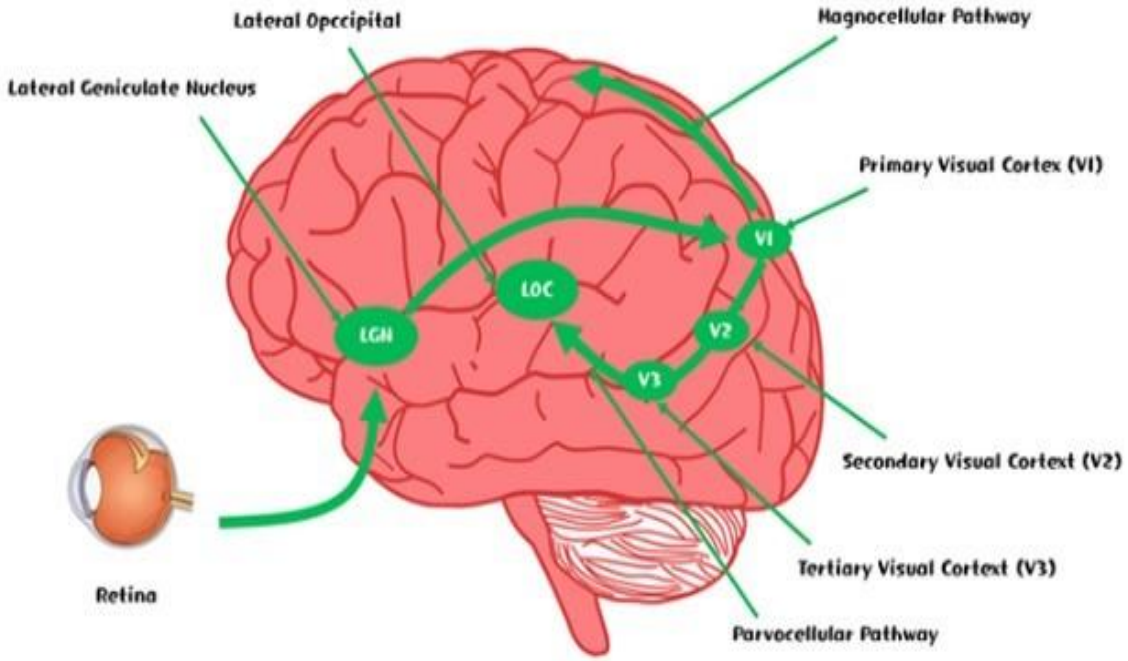

***Hình 2: So sánh giữa độ tương phản cao và độ tương phản thấp.***

Từ biểu đồ hình ảnh bên trái, chúng ta có thể thấy rằng các giá trị độ sáng (intesity) của hình ảnh bên trái nằm trong một phạm vi hẹp. Bởi vì rất khó để phân biệt các giá trị độ sáng gần giống nhau do đó hình ảnh bên trái có độ tương phản thấp. Hình ảnh bên phải làm tăng khoảng cách này giữa các giá trị độ sáng và các chi tiết trong hình ảnh giờ đây dễ nhận biết hơn đối với chúng ta và do đó mang lại hình ảnh có độ tương phản cao.

### *3. Hệ thống thị giác của con người (Human Visual System)*

Hệ thống thị giác bao gồm cả mắt và não. Ánh sáng đi vào mắt bạn nơi nó chạm vào võng mạc, kích hoạt các cơ quan tiếp nhận ánh sáng gửi tín hiệu điện qua dây thần kinh thị

giác của bạn, các dây thần kinh này truyền đến phía sau não, nơi diễn ra các giai đoạn đầu tiên của nhận thức thị giác. Sau đó, bộ não sẽ gửi các tín hiệu được lọc liên tục qua não cho đến khi bạn có thể "nhìn thấy" chúng và hành động.




***Hình 3: Cấu trúc hệ thống thị giác của con người.***

### *4. Độ chói nền (Background Luminance)*

Độ chói nền trong một bức ảnh là mức độ sáng tối của vùng nền hoặc phông nền của ảnh. Nó đo lường mức độ ánh sáng hoặc độ tối trong các vùng không có đối tượng chính hoặc vật thể nào trong ảnh.

Độ chói nền có thể ảnh hưởng đến cảm nhận của người xem về toàn bộ ảnh. Nếu độ chói nền quá sáng hoặc quá tối, nó có thể làm HSV cảm thấy khó chịu.

### *5. Mặt tương phản (Contrast Mask)*

Mặt tương phản trong xử lý ảnh là một kỹ thuật nhằm tạo ra một mặt dựa trên độ tương phản của các vùng trong ảnh. Kỹ thuật này cho phép người dùng tách các vùng của ảnh dựa trên độ tương phản của chúng, từ đó cho phép điều chỉnh các vùng này một cách độc lập để tạo ra một hình ảnh có độ tương phản tốt hơn.

### *6. Định luật Weber (Weber's Law)*

Định luật Weber phản ánh mối quan hệ giữa cảm giác và độ lớn của một kích thích vật lý đối với người. Theo đó, để nhận ra sự khác biệt giữa hai độ lớn kích thích, chúng ta cần một sự khác biệt tối thiểu trong độ mạnh của chúng.

Định luật Weber được sử dụng trong nhiều lĩnh vực khác nhau. Ví dụ, nó thường được sử dụng trong kinh doanh và nguồn nhân lực. Nhiều doanh nghiệp sử dụng các nguyên tắc của định luật Weber để điều chỉnh và tạo thương hiệu cũng như quảng cáo dễ nhớ và dễ nhận biết đối với người tiêu dùng.

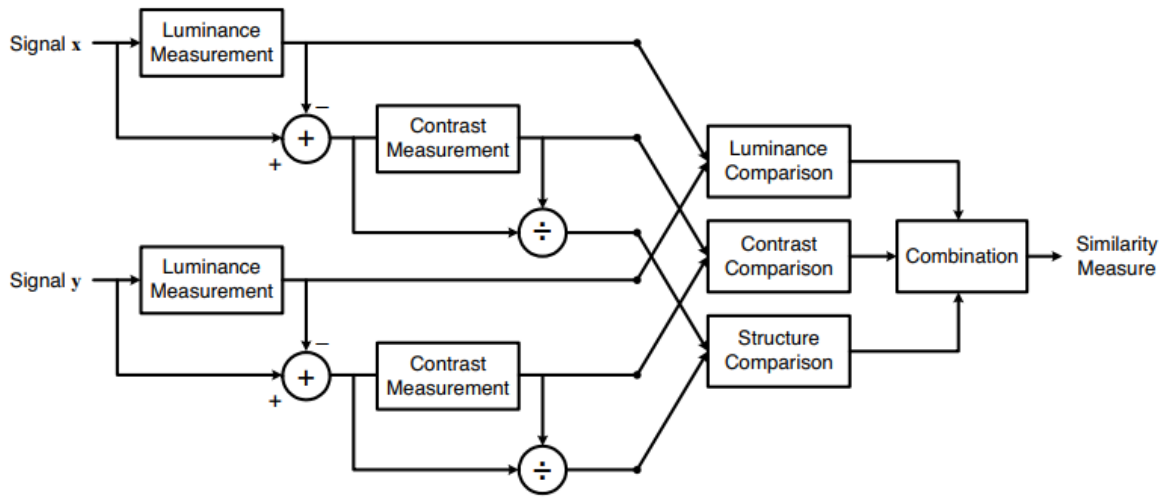
# III. Thuật toán Structural Similarity Index Measure (SSIM)

Giả sử rằng chúng ta có một bộ dữ liệu khổng lồ về hình ảnh và ta muốn lấy hai hình ảnh so sánh chúng để xác định xem chúng giống hệt nhau hay gần giống nhau theo một cách nào đó. Dưới đây chúng ta sẽ sử dụng thuật toán SSIM để thực hiện điều đó. Đầu tiên, chúng ta cần phải tìm hiểu vậy SSIM là gì?

## 1. *Định nghĩa*

SSIM hay thước đo chỉ số tương ứng trong cấu trúc là một phương pháp dự đoán chất lượng cảm nhận của truyền hình kỹ thuật số và hình ảnh điện ảnh, cũng như các loại hình ảnh và video kỹ thuật số khác. SSIM được sử dụng để đo lường sự giống nhau giữa hai hình ảnh.

SSIM là một mô hình dựa trên nhận thức, coi sự xuống cấp của hình ảnh là sự thay đổi được nhận thức trong thông tin cấu trúc, đồng thời kết hợp các hiện tượng nhận thức quan trọng, bao gồm cả thuật ngữ mặt chói và mặt phản. Sự khác biệt với các kỹ thuật khác như MSE hoặc PSNR là các phương pháp này ước tính sai số tuyệt đối. Thông tin cấu trúc là ý tưởng cho rằng các pixel có sự phụ thuộc lẫn nhau mạnh mẽ, đặc biệt là khi chúng ở gần nhau về mặt không gian. Những phụ thuộc này mang thông tin quan trọng về cấu trúc của các đối tượng trong cảnh trực quan. Mặt chói là hiện tượng biến dạng hình ảnh có xu hướng ít nhìn thấy hơn ở các vùng sáng, trong khi mặt nạ tương phản là hiện tượng biến dạng trở nên ít nhìn thấy hơn khi có hoạt động hoặc "kết cấu" quan trọng trong hình ảnh.




***Hình 4: Sơ đồ hệ thống SSIM.***

## 2. *Cách thức hoạt động*

Mỗi hình ảnh đều bao gồm rất nhiều đặc trưng riêng biệt. Nhưng trong thuật toán SSIM được sử dụng làm thước đo để đo mức độ giống nhau giữa hai hình ảnh đã cho. Ta sẽ chỉ sử dụng 3 đặc điểm chính từ mỗi hình ảnh bao gồm:

+ Độ chói

+ Sự tương phản
+ Cấu trúc

Việc so sánh giữa hai hình ảnh được thực hiện trên cơ sở 3 tính năng này.

Chỉ số SSIM giữa 2 hình ảnh có giá trị từ [-1; +1]. Giá trị +1 cho biết 2 hình ảnh đã cho rất giống nhau hoặc giống nhau trong khi giá trị -1 cho biết 2 hình ảnh đã cho khác nhau. Thông thường, các giá trị này được điều chỉnh trong phạm vi [0, 1].

### *3. Tính toán các đặc trưng*

Đầu tiên, độ chói của mỗi hình ảnh được tính bằng cách lấy trung bình tất cả các giá trị pixel hay ta có thể hiểu là giá trị cường độ trung bình.

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$

***Trong đó:*** $x_i$ là giá trị pixel thứ i của ảnh x.

N là tổng số giá trị pixel.

Tiếp theo là độ tương phản, ta sử dụng độ lệch chuẩn (căn bậc hai của phương sai tất cả các giá trị pixel) để ước tính độ tương phản của hình ảnh.

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}}$$

Và cuối cùng là cấu trúc, đầu tiên ta loại bỏ cường độ trung bình khỏi hình ảnh rồi sau đó chia cho độ lệch chuẩn của nó để kết quả có độ lệch chuẩn đơn vị cho phép so sánh tốt hơn. Với X là hình ảnh đầu vào.

$$\frac{x - \mu_x}{\sigma_x}$$

### *4. Các hàm so sánh*

Những gì chúng ta thiếu bây giờ là các hàm so sánh có thể so sánh hai hình ảnh đã cho trên các tham số này và cuối cùng là một hàm kết hợp kết hợp tất cả chúng. Phần này, ta xác định các hàm so sánh và cuối cùng là hàm kết hợp mang lại giá trị chỉ số.

$$S(x,y) = f(l(x,y), c(x,y), s(x,y))$$

Và có một điểm cần lưu ý ở đây là ta cần chứng minh rằng 3 thành phần này phải độc lập với nhau. Nghĩa là sự thay đổi của độ chói hoặc/và độ tương phản không có tác động đến thành phần cấu trúc.

Để hoàn thiện định nghĩa phép đo độ tương ứng, chúng ta cần xác định ba hàm $l(x,y)$, $c(x,y)$, và $s(x,y)$, cũng như hàm hợp. Ta cũng muốn phép đo độ tương ứng để đáp ứng các điều kiện sau đây.

1. ***Tính đối xứng:*** $S(x,y) = S(y,x)$.
2. ***Giới hạn:*** $S(x,y) \leq 1$.
3. ***Cực đại duy nhất:*** $S(x,y) = 1$ khi và chỉ khi $x = y$ (trong các biểu diễn rời rạc, $x_i = y_i$ với mọi $i = 1,2,\dots,N$).

Bắt đầu với hàm so sánh độ chói $l(x,y)$, là một hàm phụ thuộc vào 2 biến $\mu_x$ và $\mu_y$ (hay giá trị cường độ trung bình của 2 hình ảnh được so sánh $x$ và $y$). Dễ nhận thấy, hàm $l(x,y)$ đều thoả mãn 3 ràng buộc được nhắc đến ở trên.

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{{\mu_x}^2 + {\mu_y}^2 + C_1}$$

Trong đó, $C_1$ là hằng số để tránh cho việc mẫu của bất đẳng thức trên dần bằng 0. Thông qua các thực nghiệm, ta sẽ chọn $C_1 = (K_1 L)^2$ với $L$ là phạm vi động cho các giá trị pixel (ta đặt nó là 255 vì ta đang xử lý hình ảnh 8 bit tiêu chuẩn). $K1 \ll 1$ là một hằng số nhỏ.

Phương trình trên cũng phù hợp về mặt định tính với định luật Weber, được sử dụng rộng rãi để mô hình hóa sự thích ứng ánh sáng trong HVS. Theo định luật Weber, độ chênh lệch của độ chói đáng chú ý $\Delta I$ gần như tỷ lệ thuận với độ chói nền I đối với một loạt các giá trị độ

chói. Nói cách khác, HVS nhạy cảm với độ chênh lệch của độ chói tương đối chứ không phải thay đổi độ chói tuyệt đối. Để đại diện cho kích thước của độ chênh lệch của độ chói so với độ chói nền, ta viết lại độ chói của tín hiệu bị méo thành $\mu_y = (1 + R)\mu_x$ nên thay vào biểu thức ta có:

$$l(x, y) = \frac{2(1 + R)}{1 + (1 + R)^2 + \frac{C_1}{{\mu_x}^2}}$$

Nếu chúng ta giả sử $C_1$ là nhỏ đủ với ${\mu_x}^2$ khi đó hàm $l(x, y)$ sẽ chỉ phụ thuộc vào biến $R$.

Tiếp theo, hàm so sánh độ tương phản hay $c(x, y)$ là hàm phụ thuộc $\sigma_x, \sigma_y$ biểu thị độ lệch chuẩn của hình ảnh $x$ và $y$. Với $C_2 = (K_2 L)^2$ và $K_2 \ll 1$. Và hàm $c(x, y)$ cũng thoả mãn 3 điều kiện ở trên.

$$c(x, y) = \frac{2\sigma_x \sigma_y + C_2}{{\sigma_x}^2 + {\sigma_y}^2 + C_2}$$

Thành phần thứ 3, hàm so sánh cấu trúc được thực hiện sau khi loại bỏ độ chói và chuẩn hóa phương sai. Cụ thể chúng ta liên kết 2 vector đơn vị $\frac{x - \mu_x}{\sigma_x}$ và $\frac{y - \mu_y}{\sigma_y}$. Mối tương quan giữa chúng là một thước đo đơn giản và hiệu quả để định lượng sự tương đồng về cấu trúc. Chú ý rằng mối tương quan giữa $\frac{x - \mu_x}{\sigma_x}$ và $\frac{y - \mu_y}{\sigma_y}$ là tương đương với hệ số tương quan giữa $x$ và $y$. Như vậy ta định nghĩa hàm so sánh cấu trúc như sau:

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3}$$

Tương tự như trong các phép đo độ chói và độ tương phản, chúng ta đã giới thiệu một hằng số nhỏ ở cả mẫu số, tử số và ở đây chúng ta sẽ sử dụng hằng số $C_3$. Vậy hàm so sánh cấu trúc $\sigma_{xy}$ có thể được tính như sau:

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

⇒ ***Tính SSIM:*** Bước cuối cùng chúng ta hợp nhất 3 phép so sánh đã tính toán ở trên để tính ra SSIM:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma}$$

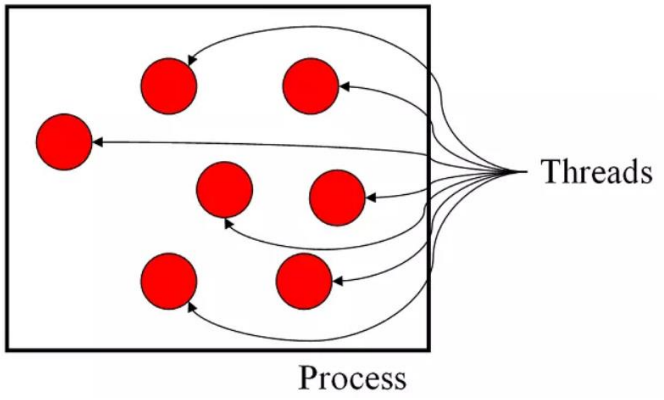
Trong đó $\alpha > 0, \beta > 0, \gamma > 0$ biểu thị tầm quan trọng tương đối của từng thành phần. Để đơn giản hóa biểu thức, nếu chúng ta giả sử, $\alpha = \beta = \gamma = 1$ và $C_3 = \frac{C_2}{2}$, thì ta có thể thu được kết quả cuối cùng có dạng như dưới đây:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_x\sigma_y + C_2)}{({\mu_x}^2 + {\mu_y}^2 + C_1)({\sigma_x}^2 + {\sigma_y}^2 + C_2)}$$

Giả định trong trường hợp đặc biệt khi $C_1 = C_2 = 0$, khi đó chỉ cần $({\mu_x}^2 + {\mu_y}^2)$ hoặc $({\sigma_x}^2 + {\sigma_y}^2)$ có xu hướng dần về 0 hay trong trường xấu nhất là cả hai đều dần về 0 thì chỉ số SSIM thu được cuối cùng sẽ là một kết quả không ổn định. Nên ta cần lưu ý để tránh trường hợp trên xảy ra.

# *IV. Đa luồng trong Python*

## *1. Phân biệt giữa Process và Threads*



***Hình 5: Sự khác nhau giữa Process và Thread.***

Tiến trình (Process) là 1 thực thể (Instance) của chương trình máy tính. Tiến trình sinh ra các luồng (Threads) hay còn gọi là sub-processing để xử lý các nhiệm vụ phụ. Threads tồn tại bên trong tiến trình và chia sẻ cùng không gian bộ nhớ của Process.

Trong Cpython, Global Interpreter Lock – GIL giới hạn chỉ có một thread được chạy tại một thời điểm, nên về cơ bản Cpython sẽ không hoàn toàn hỗ trợ đa luồng, nhưng ta có thể sử dụng các Interpreter khác (Jython và IronPython) không sử dụng GIL vì vậy có thể chạy đa luồng.

### *2. Multiprocessing*

Multiprocessing là khả năng của một hệ thống hỗ trợ nhiều bộ vi xử lý processor cùng một lúc. Các ứng dụng trong hệ thống đa xử lý được chia thành nhiều quy trình nhỏ và chạy độc lập, và hệ điều hành sẽ phân bổ các luồng này cho bộ vi xử lý để cải thiện hiệu suất của hệ thống.

Multiprocessing trong Python là một module hỗ trợ lập trình viên có thể phân chia công việc theo nhiều quy trình. Bằng cách thông qua những phương thức (API) mà module cung cấp sẵn, chúng ta có thể quản lý được các task một cách dễ dàng.

Module multiprocessing cung cấp các lớp và hàm để tạo và quản lý các tiến trình trong Python. Các tiến trình này có thể chạy độc lập với nhau và có thể chia sẻ dữ liệu với nhau thông qua một số cơ chế như pipes, queues, hoặc shared memory.

### *3. Multithreading*

Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời và mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm, để sử dụng tốt nhất các nguồn có sẵn, đặc biệt khi máy tính của bạn có nhiều CPU.

Python cung cấp thread Module và threading Module để bạn có thể bắt đầu một thread mới cũng như một số tác vụ khác trong khi lập trình đa luồng. Mỗi một Thread đều có vòng đời chung là bắt đầu, chạy và kết thúc. Một Thread có thể bị ngắt (interrupt), hoặc tạm thời bị dừng (sleeping) trong khi các Thread khác đang chạy – được gọi là yielding.

+ Trong dự án này, ta sử dụng "ThreadPoolExecutor" để tạo ra một thread pool và submit các task kiểm tra đồng bộ hóa cho các cặp ảnh này vào thread pool. Thread Pool Executor sẽ quản lý việc chạy các task trên các thread riêng biệt và tự động phân chia các task cho các thread để tối đa hóa tốc độ thực thi.
+ Với mỗi task được submit vào thread pool, chúng ta lưu trữ kết quả trả về (nếu có) vào danh sách các bức ảnh trùng lặp.
+ Sau khi các task đã hoàn thành, chúng ta in ra danh sách các bức ảnh trùng lặp và thời gian thực hiện của chương trình.

Bằng cách sử dụng multithreading, chúng ta có thể thực hiện so sánh các cặp ảnh đồng thời trên nhiều thread khác nhau, giúp tăng tốc độ thực thi chương trình.

# V. *Lập trình*

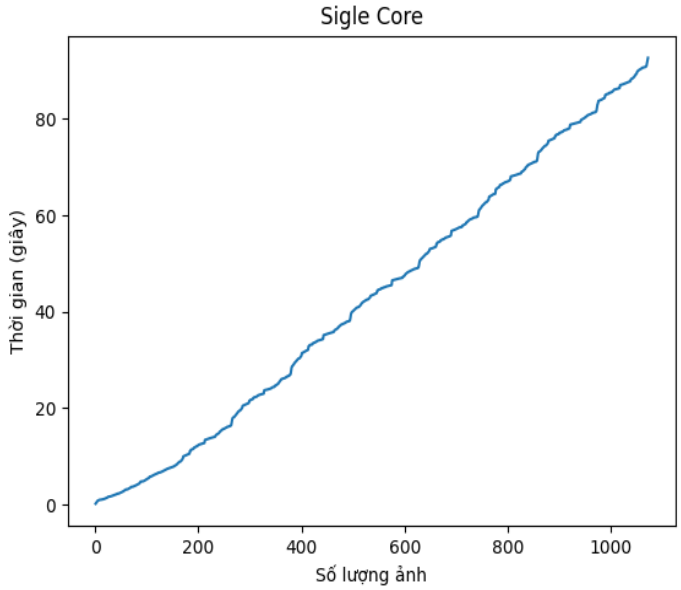
Chúng ta sử dụng bộ dữ liệu từ kaggle để test chương trình

## 1. *Chương trình đơn Single_Core.py*

```
import os
import cv2
from skimage.metrics import structural_similarity as ssim
import time

# Đường dẫn đến thư mục chứa các bức ảnh
path = "./images_in"

# Lấy danh sách các tệp tin ảnh trong thư mục
image_files = [os.path.join(path, f) for f in os.listdir(path) if
               os.path.isfile(os.path.join(path, f)) and f.endswith(('.png',
'.jpg', '.jpeg', '.bmp'))]

# Danh sách các bức ảnh trùng lặp
duplicates = []
```

```
start_time = time.time()  # Lấy thời điểm bắt đầu chạy code

# So sánh các cặp ảnh trong danh sách
for i in range(len(image_files)):
    for j in range(i + 1, len(image_files)):
        # Đọc ảnh từ file
        img1 = cv2.imread(image_files[i])
        img2 = cv2.imread(image_files[j])

        # Chuyển ảnh sang độ xám
        img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
        img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

        # Resize ảnh cùng kích thước
        img1_gray_resized = cv2.resize(img1_gray, (img2_gray.shape[1],
img2_gray.shape[0]))

        # Tính toán chỉ số SSIM
        ssim_score = ssim(img1_gray_resized, img2_gray)

        # Nếu chỉ số SSIM > 0.95, coi như 2 ảnh giống nhau và lưu vào danh sách các bức ảnh trùng lặp
        if ssim_score > 0.95:
            duplicates.append((image_files[i], image_files[j]))

end_time = time.time()  # Lấy thời điểm kết thúc chạy code
elapsed_time = end_time - start_time  # Tính thời gian chạy code

# In ra danh sách các bức ảnh trùng lặp
if duplicates:
    print("Các bức ảnh trùng lặp:")
    for dup in duplicates:
        print(dup[0], dup[1])
else:
    print("Không có bức ảnh trùng lặp trong thư mục này.")

print("Thời gian chạy code: %.2f giây" % elapsed_time)
```

Đánh giá về mặt thời gian:




***Hình 6: Biểu đồ thời gian khi chạy chương trình bằng Sigle Core.***

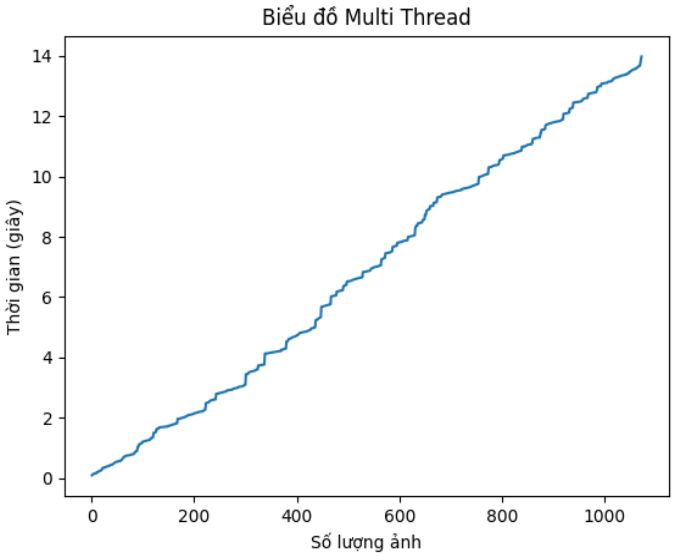
Qua số liệu được thống kê từ chương trình chúng ta thấy biểu đồ về mặt thời gian tăng dần tỉ lệ với số lượng ảnh tăng lên.

### 2. *Chương trình song song detect_multithreading.py*

```
import cv2, os
import concurrent.futures
from skimage.metrics import structural_similarity as ssim
import time

# Đường dẫn đến thư mục chứa các bức ảnh
path = "./images_in"

# Lấy danh sách các tệp tin ảnh trong thư mục
image_files = [os.path.join(path, f) for f in os.listdir(path) if
               os.path.isfile(os.path.join(path, f)) and f.endswith(('.png',
'.jpg', '.jpeg', '.bmp'))]

# Hàm kiểm tra đồng bộ hóa so sánh 2 bức ảnh
def check_duplicate(img_path1, img_path2):
    # Đọc ảnh từ file
    img1 = cv2.imread(img_path1)
    img2 = cv2.imread(img_path2)

    # Chuyển ảnh sang độ xám
    img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Resize ảnh cùng kích thước
    img1_gray_resized = cv2.resize(img1_gray, (img2_gray.shape[1],
img2_gray.shape[0]))
```

```
    # Tính toán chỉ số SSIM
    ssim_score = ssim(img1_gray_resized, img2_gray)

    # Nếu chỉ số SSIM > 0.95, coi như 2 ảnh giống nhau và trả về kết quả
    if ssim_score > 0.95:
        return (img_path1, img_path2)

# Tạo các cặp bức ảnh để so sánh
pairs = []
for i in range(len(image_files)):
    for j in range(i + 1, len(image_files)):
        pairs.append((image_files[i], image_files[j]))

# Sử dụng multithreading để so sánh các cặp ảnh
duplicates = []
start_time = time.time()
with concurrent.futures.ThreadPoolExecutor() as executor:
    # Tạo các task và submit chúng vào thread pool
    futures = [executor.submit(check_duplicate, pair[0], pair[1]) for pair
in pairs]

    # Lấy kết quả trả về từ các task đã hoàn thành
    for future in concurrent.futures.as_completed(futures):
        result = future.result()
        if result is not None:
            duplicates.append(result)
end_time = time.time()

# In ra danh sách các bức ảnh trùng lặp
if duplicates:
    print("Các bức ảnh trùng lặp:")
    for dup in duplicates:
        print(dup[0], dup[1])
else:
    print("Không có bức ảnh trùng lặp trong thư mục này.")

print("Thời gian thực hiện: ", end_time - start_time, "giây")
```

Đánh giá về mặt thời gian:



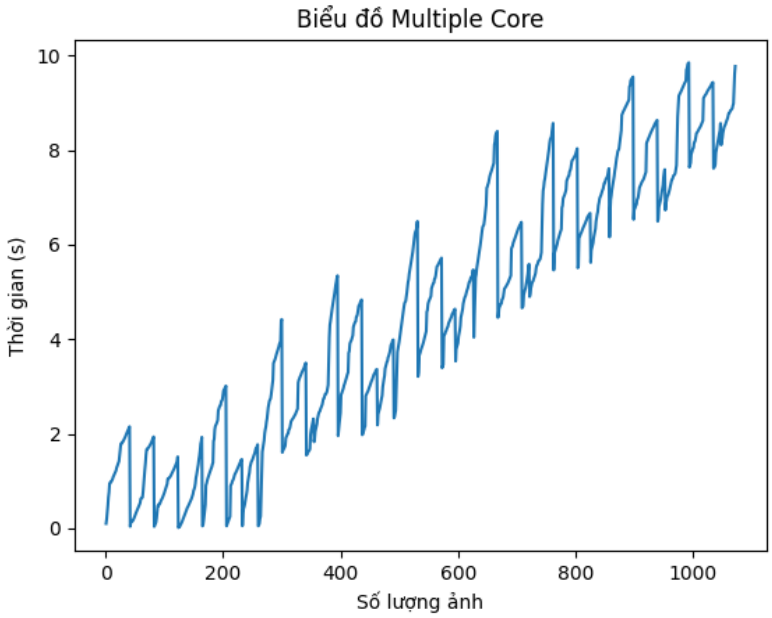

***Hình 7: Biểu đồ thời gian khi chạy chương trình bằng phương pháp Multi Thread.***

### 3. *Chương trình song song detect_multiprocessing.py*

```
import cv2, os
from skimage.metrics import structural_similarity as ssim
import time
from multiprocessing import Pool, cpu_count

# Đường dẫn đến thư mục chứa các bức ảnh
path = "./images_in"
# Lấy danh sách các tệp tin ảnh trong thư mục
image_files = [os.path.join(path, f) for f in os.listdir(path) if
               os.path.isfile(os.path.join(path, f)) and f.endswith(('.png',
'.jpg', '.jpeg', '.bmp'))]
# Danh sách các bức ảnh trùng lặp
duplicates = []

def compare_images(pair):
    i, j = pair
    # Đọc ảnh từ file
    img1 = cv2.imread(image_files[i])
    img2 = cv2.imread(image_files[j])

    # Chuyển ảnh sang độ xám
    img1_gray = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

    # Resize ảnh cùng kích thước
    img1_gray_resized = cv2.resize(img1_gray, (img2_gray.shape[1],
img2_gray.shape[0]))

    # Tính toán chỉ số SSIM
    ssim_score = ssim(img1_gray_resized, img2_gray)

    # Nếu chỉ số SSIM > 0.95, coi như 2 ảnh giống nhau và lưu vào danh sách các bức ảnh trùng lặp
```

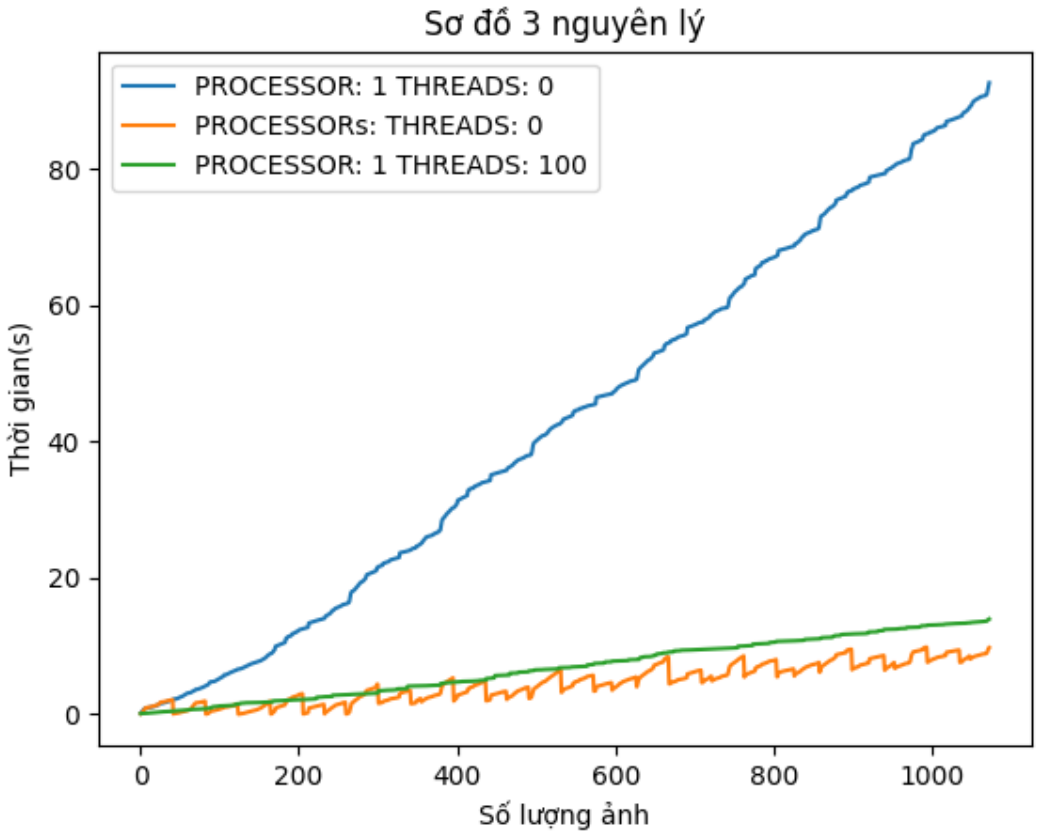
```
    if ssim_score > 0.95:
        return (image_files[i], image_files[j])
    return None

if __name__ == '__main__':
    start_time = time.time()  # Lấy thời điểm bắt đầu chạy code
    pairs = [(i, j) for i in range(len(image_files)) for j in range(i + 1,
len(image_files))]
    with Pool(processes=cpu_count()) as pool:
        results = pool.map(compare_images, pairs)
    duplicates = [res for res in results if res is not None]

    end_time = time.time()  # Lấy thời điểm kết thúc chạy code
    elapsed_time = end_time - start_time  # Tính thời gian chạy code

  # In ra danh sách các bức ảnh trùng lặp
    if duplicates:
        print("Các bức ảnh trùng lặp:")
        for dup in duplicates:
            print(dup[0], dup[1])
    else:
        print("Không có bức ảnh trùng lặp trong thư mục này.")

    print("Thời gian chạy code: %.2f giây" % elapsed_time)
```

Đánh giá về mặt thời gian:




***Hình 8: Biểu đồ thời gian khi chạy chương trình bằng phương pháp Multiple Core.***

### *4. Chương trình song song detect_multiprocessing.py*




***Hình 9: Biểu đồ thời gian để so sánh giữa ba phương pháp.***

Chúng ta có thể quan sát rõ ràng ở đây rằng thời gian dành cho ứng dụng đơn lõi, đơn luồng tăng theo cấp số nhân theo thời gian. Nhưng đối với MultiProcessing và MultiThreading, nó tăng với một giá trị độ dốc nhỏ. Ngoài quan sát, thực tế là thư viện MultiProcessing đang đánh bại luồng khi dữ liệu được tăng lên, điều này là do MultiProcessing chiếm nhiều tài nguyên và do đó mất nhiều thời gian hơn để bắt đầu xử lý. Nhưng sau khi đã bắt đầu xử lý thì quá trình xử lý sẽ diễn ra rất nhanh chóng và sớm đưa ra kết quả. Trong khi đó, Threads có các điểm ngắt nhằm giảm hiệu suất của ứng dụng. Do đó, nó không thể hoạt động một cách trơn tru.

# *VI. Tài liệu tham khảo*


1) Z. Wang, A. C. Bovik, and L. Lu, “Why is image quality assessment so difficult,” in Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, Orlando, FL, May 2002.
2) Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh and Eero P. Simoncelli, “Image Quality Assessment: From Error Visibility to Structural Similarity”, IEEE, April 2004.
3) A. M. Eskicioglu and P. S. Fisher, “Image quality measures and their performance,” IEEE Trans. Commun, Dec. 1995.
4) Roman Trobec, Boštjan Slivnik, Patricio Bulić, Borut Robič, “Introduction to Parallel Computing”, From Algorithms to Programming on State-of-the-Art Platforms, AG 2018.