



★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



All about Structural Similarity Index (SSIM): Theory + Code in PyTorch



Pranjal Datta · [Follow](#)

Published in SRM MIC · 12 min read · Sep 3, 2020



422



6



Recently, while implementing a depth estimation paper, I came across the term **Structural Similarity Index(SSIM)**. SSIM is used as a metric to measure the *similarity* between two given images. As this technique has been around since 2004, a lot of material exists explaining the theory behind SSIM but very few resources go deep into the details, that too specifically for a gradient-based implementation as SSIM is often used as a loss function. Hence, this article is my humble attempt to plug this gap!

The objective of this article is two-fold,

- To explain the theory and intuition behind SSIM and explore some of its application in current cutting edge Deep Learning.
- Go deep into a PyTorch implementation. You can skip to the code [here](#). The full implementation can be found as a standalone notebook [here](#).

Just click on the “Open in Colab” link to start running the code!

So let's begin!

The Theory

SSIM was first introduced in the 2004 IEEE paper, *Image Quality Assessment: From Error Visibility to Structural Similarity*. The abstract provides a good intuition into the idea behind the system proposed,

Objective methods for assessing perceptual image quality traditionally attempted to quantify the visibility of errors (differences) between a distorted image and a reference image using a variety of known properties of the human visual system. Under the assumption that human visual perception is highly adapted for extracting structural information from a scene, we introduce an alternative complementary framework for quality assessment based on the degradation of structural information.

Summary: The authors make 2 essential points,

- Most Image quality assessment techniques rely on quantifying errors between a reference and a sample image. A common metric is to quantify the difference in the values of *each of the corresponding* pixels between the sample and the reference images (By using, for example, *Mean Squared Error*).
- The *Human visual perception system* is highly capable of identifying structural information from a scene and hence identifying the *differences between the information* extracted from a reference and a sample scene.

Hence, a metric that replicates this behavior will perform better on tasks that involve differentiating between a sample and a reference image.

The Structural Similarity Index (SSIM) metric extracts 3 key *features* from an image:

- **Luminance**
- **Contrast**
- **Structure**

The comparison between the two images is performed on the basis of these 3 features.

Fig 1 given below shows the arrangement and flow of the Structural Similarity Measurement system. *Signal X* and *Signal Y* refer to the Reference and Sample Images.

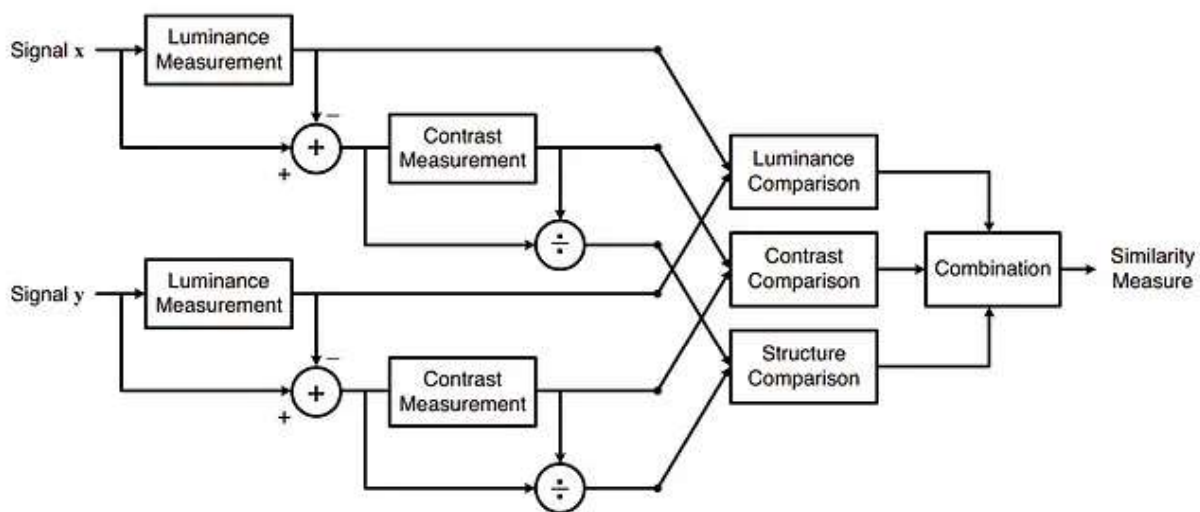


Fig 1: The Structural Similarity Measurement System. Source: <https://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf>

But what does this metric calculate?

This system calculates the *Structural Similarity Index* between 2 given images which is a value between -1 and +1. A *value of +1* indicates that the 2 given images are **very similar or the same** while a *value of -1* indicates the 2 given images are **very different**. Often these values are adjusted to be in the range [0, 1], where the extremes hold the same meaning.

Now, let's explore briefly, how these features are represented mathematically, and how they contribute to the final SSIM score.

- **Luminance:** Luminance is measured by *averaging* over all the pixel values. Its denoted by μ (Mu) and the formula is given below,

$$\mu_x = \frac{1}{N} \sum_{i=1}^N x_i. \quad (2)$$

The luminance comparison function $l(\mathbf{x}, \mathbf{y})$ is then a function of μ_x and μ_y .

where x_i is the i th pixel value of the image x . N is the total number of pixel values. Source:
<https://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf>

- **Contrast:** It is measured by taking the *standard deviation (square root of variance)* of all the pixel values. It is denoted by σ (sigma) and represented by the formula below,

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)^2 \right)^{\frac{1}{2}}. \quad (4)$$

The contrast comparison $c(\mathbf{x}, \mathbf{y})$ is then the comparison of σ_x and σ_y .

Where x and y are the two images and mu is the mean of the pixel values of the image. Source:

<https://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf>

- **Structure:** The structural comparison is done by using a consolidated formula (more on that later) but in essence, we divide the input signal with its *standard deviation* so that the result has unit standard deviation which allows for a more robust comparison.

$$(\mathbf{x} - \mu_x) / \sigma_x$$

where x is the Input Image

So now we have established the mathematical intuition behind the three parameters. But hold on! We are not yet done with the math, a little bit more. What we lack now, are **comparison functions** that can *compare* the two given images on these parameters, and finally, a **combination function** that combines them all. Here, we define the comparison functions and finally the combination function that yields the *similarity index value*

- **Luminance comparison function:** It is defined by a function, $l(x, y)$ which is shown below. μ (mu) represents the mean of a given image. x and y are the two images being compared.

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

where C_1 is a constant to ensure stability when the denominator becomes 0. C_1 is given by,

$$C_1 = (K_1 L)^2$$

Update: Throughout the article, we had not explored what the K and L constants in this equation are. Thankfully a reader pointed that out in the comments, and so in the interest of making this article a bit more helpful, I'll just define them here.

L is the dynamic range for pixel values (we set it as 255 since we are dealing with standard 8-bit images). You can read more about what are the different image types and what they mean, [here](#).

K_1 , K_2 are just normal constants, nothing much there!

- **Contrast comparison function:** It is defined by a function $c(x, y)$ which is shown below. σ denotes the standard deviation of a given image. x and y are the two images being compared.

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

where C_2 is given by,

$$C_2 = (K_2L)^2$$

- **Structure comparison function:** It is defined by the function $s(x, y)$ which is shown below. σ denotes the standard deviation of a given image. x and y are the two images being compared.

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

where $\sigma(xy)$ is defined as,

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

And finally, the SSIM score is given by,

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

where $\alpha > 0$, $\beta > 0$, $\gamma > 0$ denote the relative importance of each of the metrics. To simplify the expression, if we assume, $\alpha = \beta = \gamma = 1$ and $C_3 = C_2/2$, we can get,

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

But there's a plot twist!

While you would be able to implement SSIM using the above formulas, chances are it won't be as good as the ready-to-use implementations available, as the authors explain that,

For image quality assessment, it is useful to apply the SSIM index locally rather than globally. First, image statistical features are usually highly spatially nonstationary. Second, image distortions, which may or may not depend on the local image statistics, may also be space-variant. Third, at typical viewing distances, only a local area in the image can be perceived with high resolution by the human observer at one time instance (because of the foveation feature of the HVS [49], [50]). And finally, localized quality measurement can provide a spatially varying quality map of the image, which delivers more information about the quality degradation of the image and may be useful in some applications.

Summary: Instead of applying the above metrics *globally* (i.e. all over the image at once) it's better to apply the metrics *regionally* (i.e. in small sections of the image and taking the mean overall).

This method is often referred to as the *Mean Structural Similarity Index*.

Due to this change in approach, our formulas also deserve modifications to reflect the same (it should be noted that this approach is more common and will be used to explain the code).

(**Note:** If the content below seems a bit overwhelming, no worries! If you get the gist of it, then going through the code will give you a much clearer idea.)

The authors use an 11x11 circular-symmetric Gaussian Weighing function (*basically*, an 11x11 matrix whose values are derived from a gaussian distribution) which moves pixel-by-pixel over the entire image. At each step, the local statistics and SSIM index are calculated within the local window. Since we are now calculating the metrics locally, our formulas are revised as,

$$\begin{aligned}\mu_x &= \sum_{i=1}^N w_i x_i \\ \sigma_x &= \left(\sum_{i=1}^N w_i (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \\ \sigma_{xy} &= \sum_{i=1}^N w_i (x_i - \mu_x)(y_i - \mu_y).\end{aligned}$$

Where w_i is the gaussian weighting function.

If you found this a bit unintuitive, no worries! It suffices to imagine w_i as a *multiplicand that is used to calculate the required values with the help of some mathematical tricks*.

Once computations are performed all over the image, we simply take the *mean of all the local SSIM values* and arrive at the **global** SSIM value.

$$\text{MSSIM}(\mathbf{X}, \mathbf{Y}) = \frac{1}{M} \sum_{j=1}^M \text{SSIM}(\mathbf{x}_j, \mathbf{y}_j)$$

Finally done with the theory! Now onto the code!

The Code

Before we plunge into the code, it's important to note that we won't be going through **every** line but we will explore in-depth the essential ones. Let's get started!

The full code can be found as a standalone notebook [here](#). Just click on the “Open in Colab” button to start running the code! The explanation in this section will be referring to the notebook mentioned above.

First, let's explore some utility functions that perform some essential tasks.

Function #1: **gaussian(window_size, sigma)**

This function *essentially* generates a list of numbers (of *length equal to window_size*) sampled from a **gaussian** distribution. The sum of all the elements is equal to 1 and the values are normalized. *Sigma* is the standard deviation of the gaussian distribution.

Note: This is used to generate the 11x11 gaussian window mentioned above.

Example:

Code:

```
gauss_dis = gaussian(11, 1.5)
print("Distribution: ", gauss_dis)
print("Sum of Gauss Distribution:", torch.sum(gauss_dis))
```

Output:

```
Distribution:  tensor([0.0010, 0.0076, 0.0360, 0.1094, 0.2130,
0.2660, 0.2130, 0.1094, 0.0360,0.0076, 0.0010])
```

```
Sum of Gauss Distribution: tensor(1.)
```

Function #2: create_window(window_size, channel)

While we generated a 1D tensor of gaussian values, the 1D tensor itself is of no use to us. Hence we gotta convert it to a 2D tensor (the 11x11 Tensor we talked about earlier). The steps taken in this function are as follows,

- Generate the 1D tensor using the *gaussian function*.
- Convert it to a 2D tensor by cross-multiplying with its transpose (this preserves the gaussian character).
- Add two extra dimensions to convert it to 4D. (This is only when SSIM is used as a loss function in computer vision)
- Reshape to adhere to PyTorch weight's format.

Code:

```
window = create_window(11, 3)
print(window.shape)
```

Output:

```
torch.Size([3, 1, 11, 11])
```

Now that we have explored the two utility functions, let's go through the main code! The core SSIM is implemented through the *ssim()* function which is explored below.

Function #3: *ssim(img1, img2, val_range, window_size=11, window=None, size_average=True, full=False)*

Before we move onto the essentials, let us explore what happens in the function *before* the ssim metrics are calculated,

- We set the maximum value of the normalized pixels (implementation detail; needn't worry)
- We initialize the gaussian window by means of the *create_window()* function IF a window was not provided during the function call.

Once these steps are completed, we go about calculating the various values (the sigmas and the mus of the world) which are needed to arrive at the final SSIM score.

Note: Since we are calculating local statistics *and* we need to make it computationally efficient, the formulas used are *slightly* different (They are just permutations of the formulas discussed above. Relevant mathematical materials are provided in the appendix.)

- We first calculate $\mu(x)$, $\mu(y)$, their squares, and $\mu(xy)$. *channels* here store the number of color channels of the input image. The *groups* parameter is

used to apply a convolution filter to **all** the input channels. More information regarding *groups* can be found [here](#).

```
channels, height, width = img1.size()

mu1 = F.conv2d(img1, window, padding=pad, groups=channels)
mu2 = F.conv2d(img2, window, padding=pad, groups=channels)

mu1_sq = mu1 ** 2
mu2_sq = mu2 ** 2

mu12 = mu1 * mu2
```

- We then go on to calculate the squares of $\sigma(x)$, $\sigma(y)$, and $\sigma(xy)$. For more math, check Appendix [1.1](#).

```
sigma1_sq = F.conv2d(img1 * img1, window, padding=pad,
groups=channels) - mu1_sq

sigma2_sq = F.conv2d(img2 * img2, window, padding=pad,
groups=channels) - mu2_s

sigma12 = F.conv2d(img1 * img2, window, padding=pad,
groups=channels) - mu12
```

- Thirdly, we calculate the contrast metric according to the formula mentioned [here](#),

```
contrast_metric = (2.0 * sigma12 + C2) / (sigma1_sq + sigma2_sq + C2)
contrast_metric = torch.mean(contrast_metric)
```

- Finally, we calculate the SSIM score and return the mean according to the formula mentioned [here](#).

```
numerator1 = 2 * mu12 + C1
numerator2 = 2 * sigma12 + C2
denominator1 = mu1_sq + mu2_sq + C1
denominator2 = sigma1_sq + sigma2_sq + C2

ssim_score = (numerator1 * numerator2) / (denominator1 *
denominator2)

return ssim_score.mean()
```

That was a lot! Now let's see how the code performs!

We are going to test the code in three cases to check how does it perform.
Let's get going!

- **Case #1: True Image vs False Image**

In the first scenario, we are going to run 2 **very** different Images through SSIM. One of them is considered the *True* Image while the other is considered the *False* Image. (Since we are measuring the *difference*, the Truth and Falsity labels are essentially interchangeable; They are being used only as reference points.)

The images are,



False Image (left) True Image (Right)

The code below is for representation purposes only although not much different from the code in the notebook. For more detail and visualization, check the [notebook](#).

Code:

```
img1 = load_images("img1.jpg") # helper function to load images
img2 = load_images("img2.jpg")

_img1 = tensorify(img1) # helper function to convert cv2 image to
tensors

_img2 = tensorify(img2)

ssim_score = ssim(_img1, _img2, 225)
print(True vs False Image SSIM Score: ", ssim_score)
```

Output:

True vs False Image SSIM Score: tensor(0.3385)

- **Case #2: True Image vs True Image with Gaussian Noise**

In this scenario, we compare the true image and a heavily noised version of it. The images are shown below,



The noised True Image (left), The true Image (Right)

On running the same piece of code as above we get,

Code:

```
noise = np.random.randint(0, 255, (640, 480, 3)).astype(np.float32)
noisy_img = img1 + noise

_img1 = tensorify(img1)
_img2 = tensorify(noisy_img)

true_vs_false = ssim(_img1, _img2, val_range=255)
print("True vs Noised True Image SSIM Score:", true_vs_false)
```

Output:

True vs Noised True Image SSIM Score: `tensor(0.0185)`

- **Case #3: True Image vs True Image**

In the final case, we compare the True Image against itself. Hence, the image shown below is compared to itself. If our SSIM code is working perfectly, the **score** should be **one**.



The True Image

On running the piece of code shown below, we can confirm that the SSIM score for this given scenario is indeed **one**.

Code:

```
_img1 = tensorify(img1)
true_vs_false = ssim(_img1, _img1, val_range=255)
```

```
print("True vs True Image SSIM Score:", true_vs_false)
```

Output:

```
True vs True Image SSIM Score: tensor(1.)
```

Conclusion

Finally, we are here! In this article, we covered the theory behind SSIM and the code that goes into implementing it. In the References, some additional materials are provided including links to Computer Vision literature where SSIM is used in some form.

Hope understanding SSIM was much easier for you than it was for me :). I tried to focus on the areas that I personally found complicated and difficult to understand, hoping to not only consolidate my learnings but also in the process, help somebody else stumbling along the same path ;).

Would really appreciate feedback positive or negative. You can drop it in the comments section or reach out to me at pranjaldatta99@gmail.com.

References

1. *The Original Paper*: <https://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf>
2. *High-Quality Monocular Depth Estimation via Transfer Learning*: The depth estimation paper where SSIM is used as one of the loss functions. The paper can be found [here](#). You can also check my PyTorch Implementation of this paper [here](#).

3. More on SSIM Applications: <https://www.imatest.com/docs/ssim/>

Appendix

1.0: Since using local statistics, the formula for the *mean* (μ), changes from

$$\sum_i^N \mathbf{x}_i \rightarrow \sum_i^N \mathbf{w}_i \mathbf{x}_i$$

1.1: The variance (square of standard deviation) formula used in the Python code can be derived as,

$$\sigma^2 = \frac{\sum (X - \mu)^2}{N} \quad (1)$$

$$= \frac{\sum (X^2 - 2\mu X + \mu^2)}{N} \quad (2)$$

$$= \frac{\sum X^2}{N} - \frac{2\mu \sum X}{N} + \frac{N\mu^2}{N} \quad (3)$$

$$= \frac{\sum X^2}{N} - 2\mu^2 + \mu^2 \quad (4)$$

$$= \frac{\sum X^2}{N} - \mu^2 \quad (5)$$

While the derivation above shows the general method, in our context the final formula becomes,

$$\sigma^2 = \sum_i^N w_i x_i^2 - \mu^2$$

Computer Vision

Image Processing

Pytorch

Image Similarity

Image Comparison



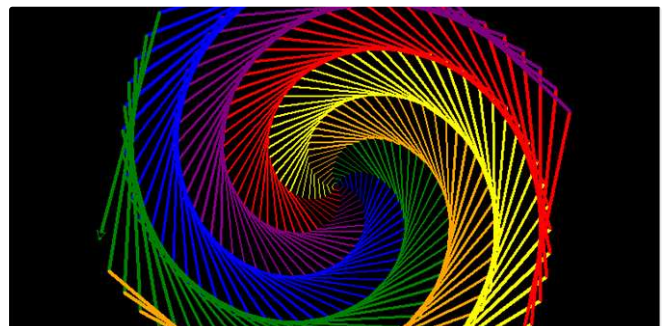
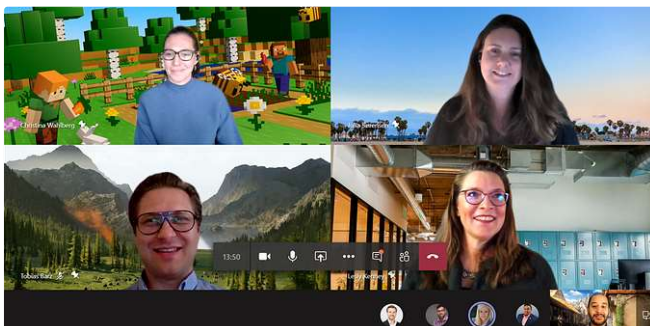
Written by Pranjal Datta

Follow

53 Followers · Writer for SRM MIC

Doing MLE, Backend, and Infra software things. Trying to get better at writing too. ML
@PixxelSpace

More from Pranjal Datta and SRM MIC





Pranjal Datta in SRM MIC

Using PSPNet to change your background in under 40 lines of...

We attempt to use an advanced segmentation technique to deploy the pipeline in a few line...

10 min read · Jul 23, 2020



156



Swarnabha Das in SRM MIC

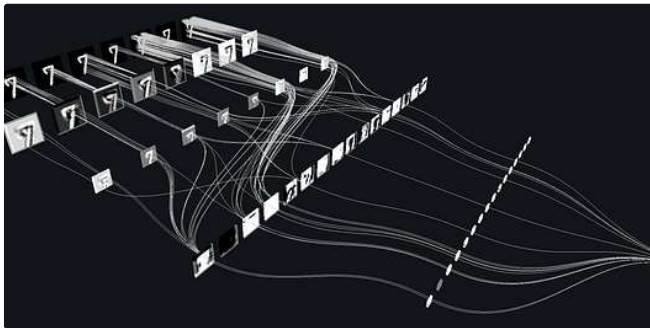
25 Lesser-known Python Modules that can Improve the Way we Code.

Python is a beautiful programming language. It has everything you will probably ever need...

14 min read · May 19, 2020



129



Aryan Kargwal in SRM MIC

ConvNet Architectures for beginners Part I

Beginner's guide to Convolutional Neural Network architectures...

5 min read · Jul 21, 2020



83



Pranjal Datta

Why college isn't irrelevant

(Originally published at <https://pranjaldata.github.io/2022/06/11/col...>)

10 min read · Dec 30, 2022



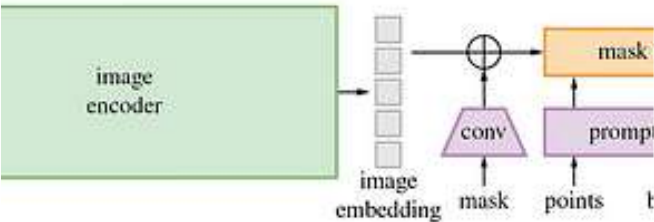
2




See all from Pranjal Datta

See all from SRM MIC

Recommended from Medium



 Skann.ai

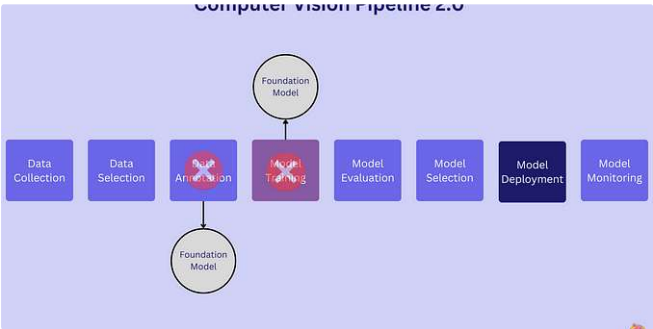
Breaking Boundaries in Computer Vision: Segment Anything Model...


Breaking Boundaries in Computer Vision: Segment Anything Model (SAM)

7 min read · Sep 21, 2023

 3 



 The Tenyks Blogger

Computer Vision Pipeline v2.0

How Foundation Models are transforming the Computer Vision pipeline.

9 min read · Jan 12

 540  2

Lists



Natural Language Processing

1118 stories · 591 saves



Felipe Bandeira in Python in Plain English

Transformers vs. OCR: who can actually read better?

A comparison between OCR-based and OCR-free approaches for Information Extraction

15 min read · Sep 1, 2023

 396

4



 Jason Corso

Annotation is dead

Human annotation is largely responsible for the current AI boom, but is the need for...

11 min read · Jan 12

624

Q



 Vasista Reddy in ScrapeHero

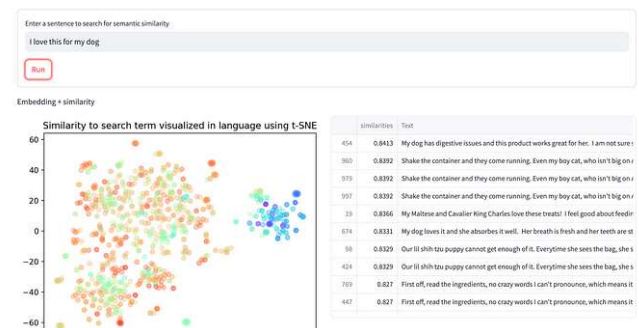
Exploring Image Similarity Approaches in Python

In a world inundated with images, the ability to measure and quantify the similarity...

5 min read · Sep 5, 2023

 77

Q 1



 Daniel Avila in LatinXinAI

Visualizing embeddings and semantic similarity with OpenAI...

In this article, we will explore an example of visualizing semantic similarities in language...

4 min read · Aug 7, 2023

 46

Q 1



[See more recommendations](#)