Let me start by setting a tone for this blog which shall help you decide if you should give it a read. This blog is written with the intent to introduce you to a list of Full-Reference evaluation metrics for assessing Image Quality. We will be understanding all the concepts by the first principle thus, making sure we don't get ahead of ourselves.

Let's Begin...
The following 4 evaluation metrics will be covered:

- Peak Signal to Noise Ratio (PSNR)
- Structural SIMilarity index (SSIM)
- Spectral Angle Mapper (SAM)
- Spatial Correlation Coefficient (SCC)

We will go through the math behind each of these so that after you finish reading this blog you have a very clear understanding of what actually goes behind the scenes. Doing this will not only help you differentiate between them but also help you decide which metric is a better fit for your use case.
PSNR: As the name suggests this technique has to deal with the peak image value and the noise density present in the image. The higher this value, the more is the noise less able to affect the peak value in the image. Thereby better image quality.

$$PSNR = 20 \log_{10} \left(\frac{MAX_f}{\sqrt{MSE}} \right)$$

Peak Signal-to-Noise Ratio

Evaluating the value on a logarithmic scale helps us to get a normalized sense of the measure of quality for the degraded image. Thus, taking care of the dynamic range of values an image can possibly have.
The two components of this evaluation metric can be explained as below:
For 8-bit pixels, the value of each pixel shall range from 0 to 255. Therefore when we refer to the peak pixel value (denoted as MAXf) this would refer to a value between this range that is present in the image.
Noise is a measure of the pixel-wise difference between the clean image and the noised image. This noise is denoted as MSE between these two images. Mathematically this is given as:

$$MSE = \frac{1}{mn} \sum_{0}^{m-1} \sum_{0}^{n-1} \|f(i,j) - g(i,j)\|^2$$

MSE as a measure of noise

where: f represents the original image, g represents the degraded image, (m,n) represents the dimensions of the image matrix

```
import cv2
import numpy as np
```

```
def getPSNR(I1, I2):
    '''
    :param I1: represents original image matrix
    :param I2: represents degraded image matrix
    :return: psnr score
    '''
    # mse calculation
    s1 = cv2.absdiff(I1, I2)
    # cannot make a square on 8 bits
    s1 = np.float32(s1)
    s1 = s1 * s1
    sse = s1.sum()
    # return zero if the difference is extremely small
    if sse <= 1e-10:
        return 0
    else:
        shape = I1.shape
        mse = 1.0 * sse / (shape[0] * shape[1] * shape[2])
        # here 255 denotes the maximum possible
        # value in a 8-bit channel
        psnr = 10.0 * np.log10((255 * 255) / mse)
        return psnr
```

SSIM: This evaluation metric looks at three features to output a final overall score. These are luminosity, contrast, and structure.

Mathematically,

Luminance corresponds to the mean, contrast corresponds to the standard deviation, and structure is an indirect determinant of the covariance of pixel values in the image.

$$l(I,J) = \frac{2\mu_I\mu_J + C_1}{\mu_I^2 + \mu_J^2 + C_1}$$

Intensity Comparison

$$c(I,J) = \frac{2\sigma_I\sigma_J + C_2}{\sigma_I^2 + \sigma_J^2 + C_2}$$

Contrast Comparison

Note: Structure comparison is performed after luminance subtraction and contrast normalization.

```
import cv2
import numpy as np

def getSSISM(I1, I2):
    '''
    :param i1: represents original image matrix
    :param i2: represents degraded image matrix
    :return: SSIM score
    '''
    # Constants for luminance and contrast
    C1 = 6.5025
    C2 = 58.5225
    # C3=C2/2
```

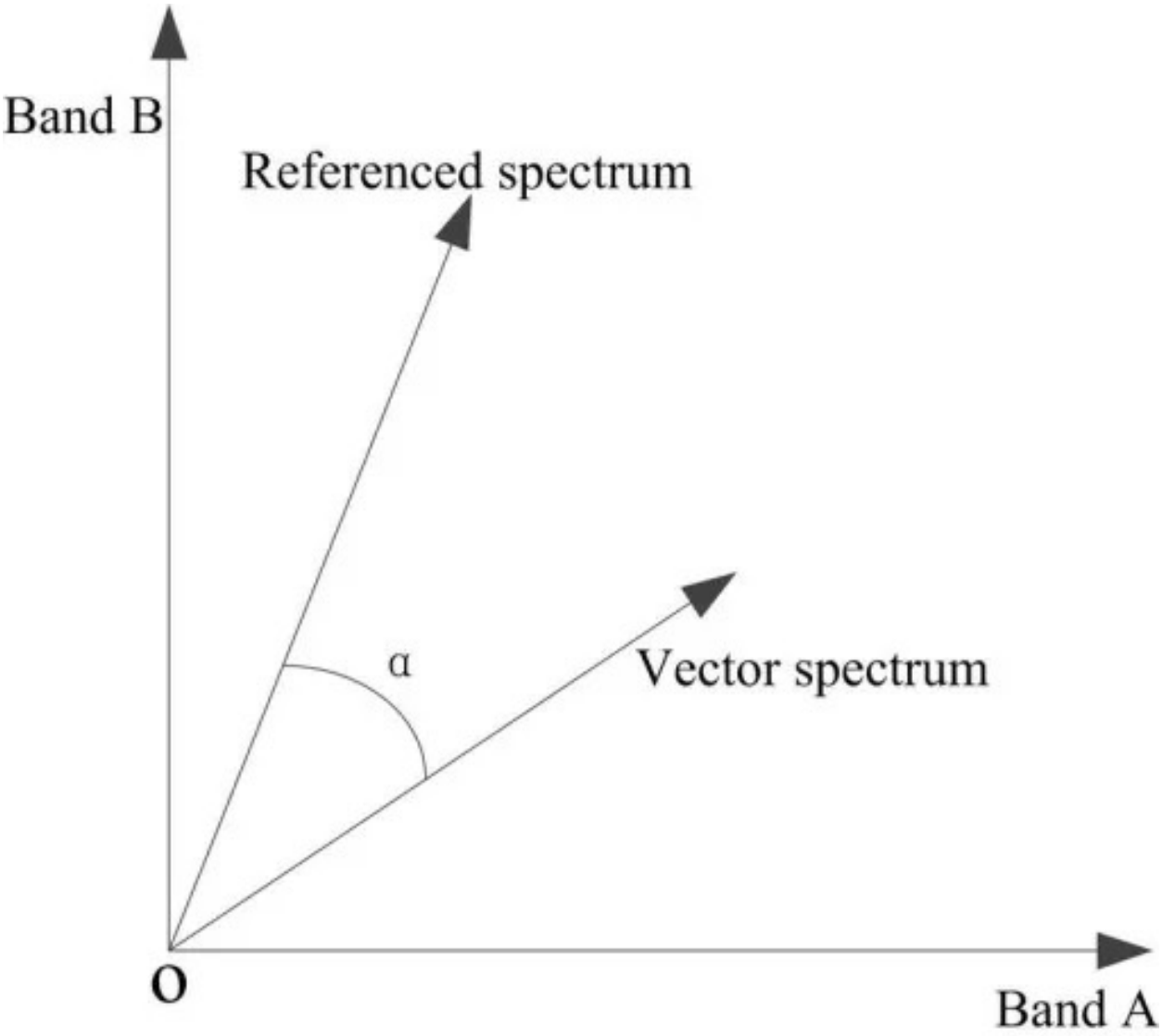
```
    # converting to float for squaring
    I1 = np.float32(I1)
    I2 = np.float32(I2)
    I2_2 = I2 * I2
    I1_2 = I1 * I1
    I1_I2 = I1 * I2

    # applying GaussianBlur with (11,11) kernel where mean=st_dev=1.5
    mu1 = cv2.GaussianBlur(I1, (11, 11), 1.5)
    mu2 = cv2.GaussianBlur(I2, (11, 11), 1.5)
    mu1_2 = mu1 * mu1
    mu2_2 = mu2 * mu2
    mu1_mu2 = mu1 * mu2

    sigma1_2 = cv2.GaussianBlur(I1_2, (11, 11), 1.5)
    sigma1_2 -= mu1_2
    sigma2_2 = cv2.GaussianBlur(I2_2, (11, 11), 1.5)
    sigma2_2 -= mu2_2
    sigma12 = cv2.GaussianBlur(I1_I2, (11, 11), 1.5)
    sigma12 -= mu1_mu2

    t1 = 2 * mu1_mu2 + C1
    t2 = 2 * sigma12 + C2
    t3 = t1 * t2  # t3 = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))
    t1 = mu1_2 + mu2_2 + C1
    t2 = sigma1_2 + sigma2_2 + C2
    t1 = t1 * t2  # t1 =((mu1_2 + mu2_2 + C1).*(sigma1_2 + sigma2_2 + C2))
    ssim_map = cv2.divide(t3, t1)
    ssim = cv2.mean(ssim_map)
    return ssim
```

SAM: Any image of dimension (n,m) can be represented as a vector in a N = (n x m) dimensional space. Therefore two images with the same dimensions which can both be represented as an N-dimensional vector can be compared by evaluating the angle between these two vectors. Two exactly similar images will have zero angle between their vector representations and pi when they are completely different.



SAM plot for vector comparison

$$a = \cos^{-1}\left(\frac{\sum_{i=1}^{nb} t_i r_i}{\sqrt{\sum_{i=1}^{nb} t_i^2} \sqrt{\sum_{i=1}^{nb} r_i^2}} \right)$$

Angle calculation

The above equation is nothing but a derivation from the dot product equation between two N-dimensional vectors under consideration as can be seen below.

$$Cos\theta = \frac{u \bullet v}{\|u\| \cdot \|v\|}$$

Equation representing dot product

```
import numpy as np

def sam (I1,I2):
  '''
  :param I1: represents original image matrix
  :param I2: represents degraded image matrix
  :returns:  float -- sam value.
  '''
  I1,I2 = _initial_check(I1,I2)

  I1 = I1.reshape((I1.shape[0]*I1.shape[1],I1.shape[2]))
  I2 = I2.reshape((I2.shape[0]*I2.shape[1],I2.shape[2]))

  N = I1.shape[1]
  sam_angles = np.zeros(N)
  for i in range(I1.shape[1]):
    val = np.clip(np.dot(I1[:,i],I2[:,i]) / (np.linalg.norm(I1[:,i])*np.linalg.norm(I2[:,i])),-1,1)
    sam_angles[i] = np.arccos(val)

  return np.mean(sam_angles)
```

SCC: This technique of image quality evaluation performs a sequence of steps on the two input images to study the correlation of variance between the clean and noised image.
Both these images are first passed through a high-pass filter with a window size of 3X3, which as a result separates out any high-frequency components from both images. These images are then sequentially passed through a uniform filter to generate variance plots. These variance plots are then studied using an L2 norm as a metric for both the images combined for each channel. The average value of the L2 norm across all channels is the final SCC score denoting the image quality.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

Correlation of variance between two images x and y

If you have followed the theory above, it will be easier for you to follow the code. I have also commented it to make it more legible.

```
import numpy as np

def _scc_single(I1,I2,win,ws):
  def _scc_filter(inp, axis, output, mode, cval):
    return correlate(inp, win , output, mode, cval, 0)

  # simple HPF operation
  I1_hp = generic_laplace(I1.astype(np.float64), _scc_filter)
  I2_hp = generic_laplace(I2.astype(np.float64), _scc_filter)

  # creating window for uniform filter
  win = fspecial(Filter.UNIFORM,ws)
  sigmaI1_sq,sigmaI2_sq,sigmaI1_I2 = _get_sigmas(I1_hp,I2_hp,win)

  sigmaI1_sq[sigmaI1_sq<0] = 0
  sigmaI2_sq[sigmaI2_sq<0] = 0

  den = np.sqrt(sigmaI1_sq) * np.sqrt(sigmaI2_sq)
  idx = (den==0)
  den = _replace_value(den,0,1)
  scc = sigmaI1_I2 / den
  scc[idx] = 0
  return scc

def scc(I1,I2,win=[[-1,-1,-1],[-1,8,-1],[-1,-1,-1]],ws=8):
  """
  :param I1: represents original image matrix
  :param P: represents degraded image matrix
  :param win: high pass filter for spatial processing (default=[[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]).
  :param ws: sliding window size (default = 8).
  :returns:  float -- scc value.
  """
  I1,I2 = _initial_check(I1,I2)
```

```
  coefs = np.zeros(I1.shape)
  for i in range(I1.shape[2]):
    coefs[:,:,i] = _scc_single(I1[:,:,i],I2[:,:,i],win,ws)
  return np.mean(coefs)
```

Conclusion:
Hurray!! (if you have come this far) we have successfully completed the blog. I hope it helped you to look at these techniques from an analytical viewpoint.
Any suggestions that could help me make it more legible for the readers to come are most welcome. You can drop your suggestions in the comments section.
PS: This is my first blog. So, if you liked the content, please don't forget to hit that clap button below.
Cheers!!
References
These functions are a part of the bigger class of python package named Sewar. The link to its Github repository can be found here.
If you're looking for your next great read, here are a few recommendations that are sure to pique your interest.
Documents Image Quality Assessment Part -1, Documents Image Quality Assessment Part -2
Python
Image Processing