# Automatic Image Quality Assessment in Python

## (Penilaian Kualitas Gambar Otomatis di Python)


1. Ardi Azhari
2. Carl Ray W
3. M. Khalif F.
4. Vika Putri A.

# LATAR BELAKANG

- Kualitas gambar adalah hal yang bersifat subjektif.
- Untuk itu diperlukan penilaian kualitas gambar yang secara kuantitatif mewakili persepsi kualitas manusia.
- Tujuannya untuk menganalisis kinerja algoritma di berbagai bidang visi komputer seperti kompresi gambar, transmisi gambar,dan pemrosesan gambar.
- Untuk mengatasai masalah tersebut ditemukanlah penilaian kualitas gambar otomatis dengan python

# Image Quality Assessment (IQA) / Penilaian Kulitas Gambar dibagi menjadi dua bidang penelitian

Reference based evaluation (Evaluasi dengan referensi)

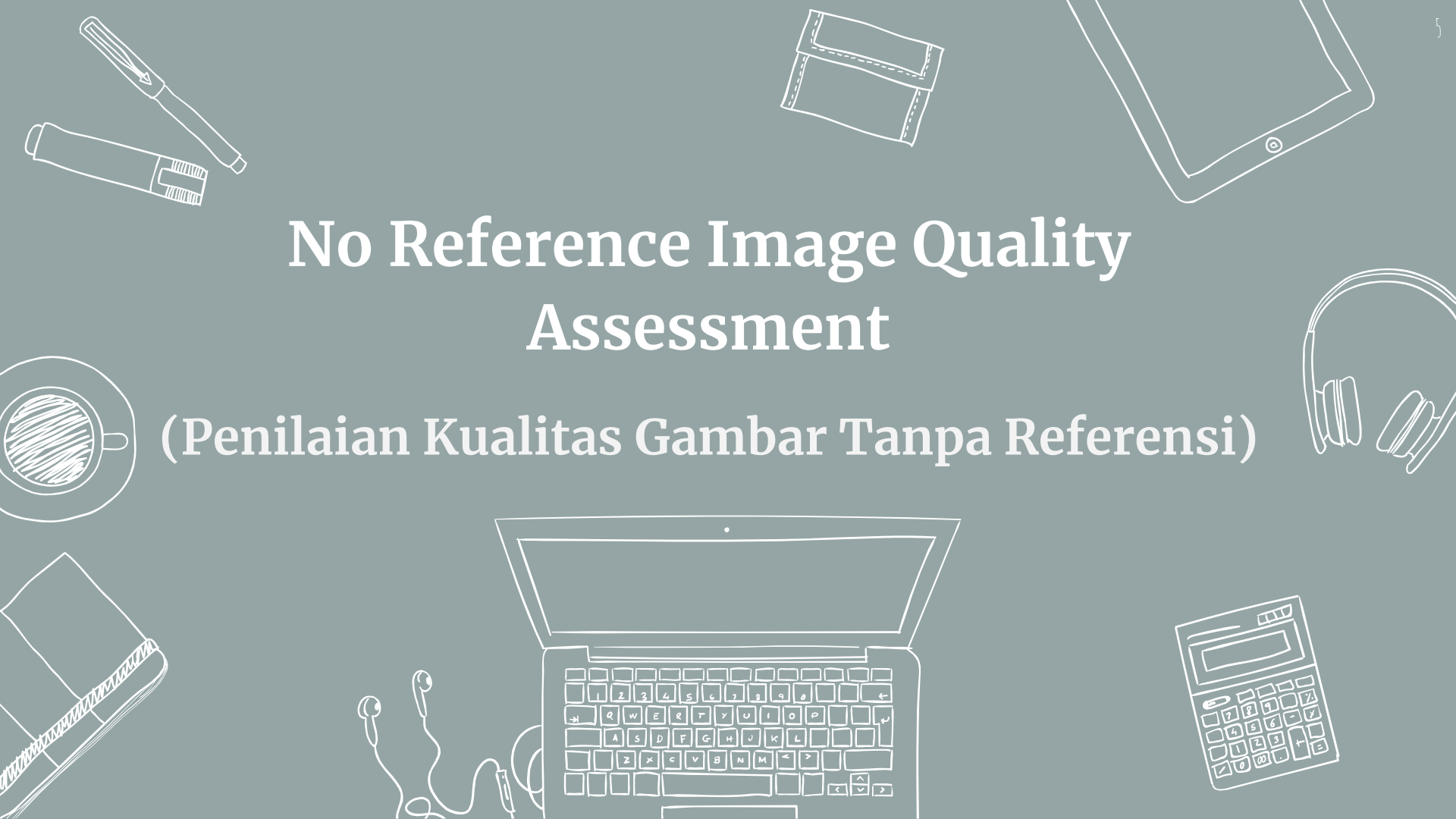
No reference evaluation (Evaluasi Tanpa referensi)

# Perbedaannya

Perbedaannya adalah metode berbasis referensi bergantung pada gambar berkualitas tinggi sebagai sumber untuk mengevaluasi perbedaan antara gambar. Contoh evaluasi berbasis referensi adalah Structural Similarity Index (SSIM). Sedangkan Penilaian kualitas gambar tanpa referensi tidak memerlukan gambar dasar untuk mengevaluasi kualitas gambar.

# No Reference Image Quality Assessment

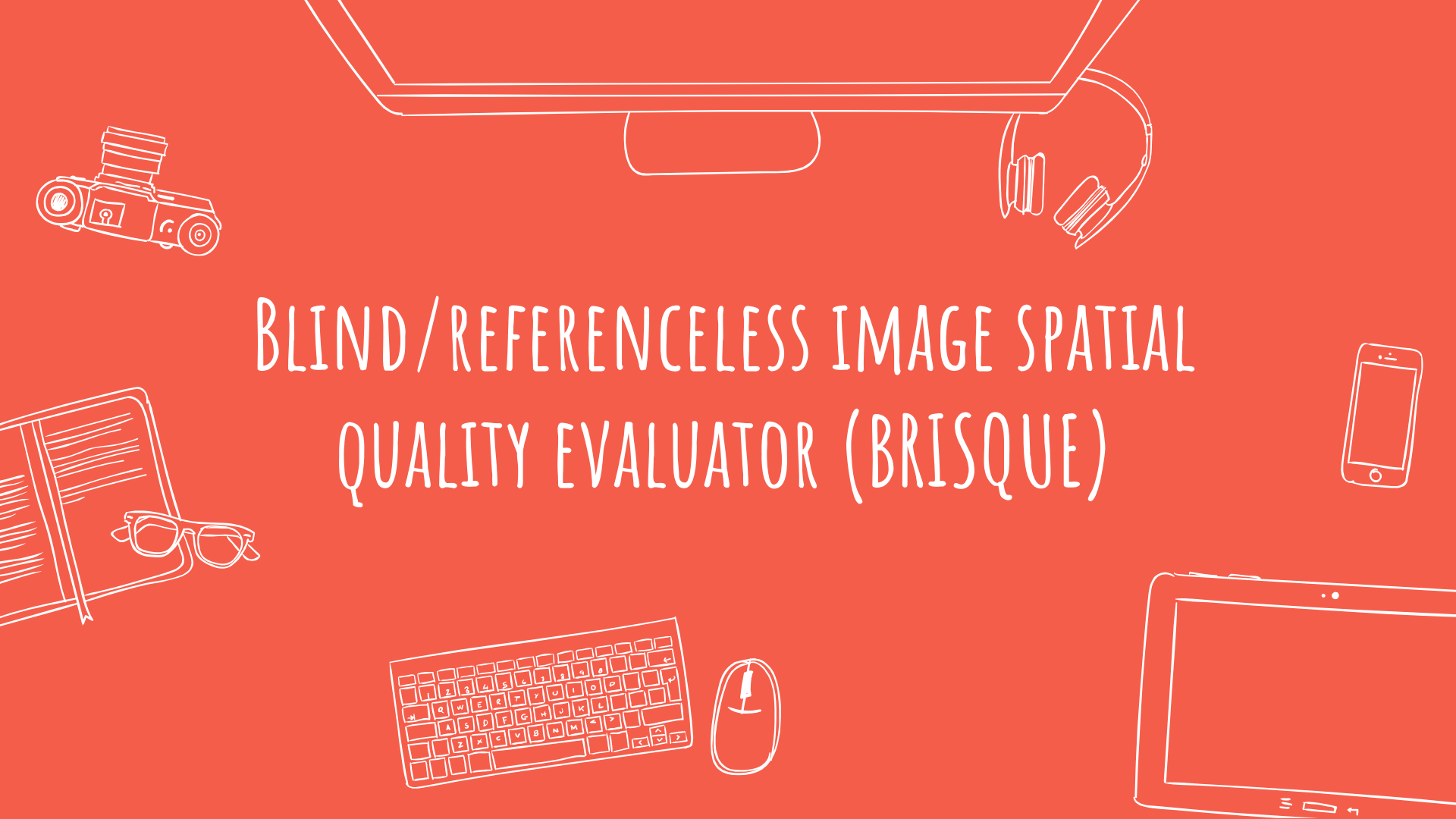
## (Penilaian Kualitas Gambar Tanpa Referensi)



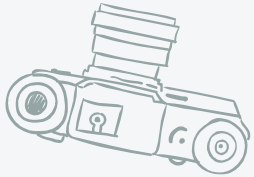
# No Reference IQA

- Tidak memerlukan gambar dasar untuk mengevaluasi kualitas gambar.
- Informasi yang diterima algoritma hanya gambar terdistorsi yang kualitasnya sedang dinilai.
- Metode Blind umumnya dibagi menjadi 2 yaitu
  1. Langkah pertama menghitung variabel yang menggambarkan struktur gambarnya
  2. Langkah kedua menemukan pola pendapat manusia dari variabel-variabel tersebut
- Untuk membandingkan kinerja algoritma IQA biasanya dengan TID2008 yaitu database yang dibuat mengikuti metodologi untuk menjelaskan cara mengukur skor pendapat manusia dari gambar yang dirujuk
- Untuk implementasi metode Deep Learning menggunakan TensorFlow 2.0

# Blind/referenceless image spatial quality evaluator (BRISQUE)



- BRISQUE adalah model yang hanya menggunakan piksel gambar untuk menghitung variabel (metode lain yang didasarkan pada transformasi gambar ke ruang lain seperti wavelet atau DCT).
- Metode ini terbukti sangat efisien karena itu tidak perlu transformasi untuk menghitung variabel.
- BRISQUE bergantung pada spasial Natural Scene Statistics (NSS) yaitu model dari koefisien luminance yang dinormalisasi secara lokal dalam domain spasial, serta model untuk produk berpasangan dari koefisien ini.



# Natural Scene Statistics (NSS) dalam Spasial Domain

```
[ ]  def normalize_kernel(kernel):
       return kernel / np.sum(kernel)

     def gaussian_kernel2d(n, sigma):
       Y, X = np.indices((n, n)) - int(n/2)
       gaussian_kernel = 1 / (2 * np.pi * sigma ** 2) * np.exp(-(X ** 2 + Y ** 2) / (2 * sigm
       return normalize_kernel(gaussian_kernel)

     def local_mean(image, kernel):
       return signal.convolve2d(image, kernel, 'same')
```

Misal diberikan sebuah gambar I(i,j)

1. Hitung luminansi yang dinormalisasi secara lokal Î(i,j) dengan pengurangan rata-rata lokal μ(i,j)
2. Membaginya dengan deviasi lokal σ(i,j) yang ditambah dengan konstanta C untuk menghindari pembagian nol.

$$\hat{I}(i,j) = \frac{I(i,j) - \mu(i,j)}{\sigma(i,j) + C}$$

```
def normalize_kernel(kernel):
    return kernel / np.sum(kernel)
```

Untuk menghitung luminansi yang dinormalisasi secara lokal atau koefisien subtracted contrast normalized (MSCN) diperlukan menghitung rata-rata lokal

$$\mu(i, j) = \sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l} I_{k,l}(i, j)$$

Dengan w adalah kernel ukuran Gaussian (K, L).

```
def gaussian_kernel2d(n, sigma):
  Y, X = np.indices((n, n)) - int(n/2)
  gaussian_kernel = 1 / (2 * np.pi * sigma ** 2) * np.exp(-(X ** 2 + Y ** 2) / (2 * sigma ** 2))
  return normalize_kernel(gaussian_kernel)
```

# Kemudian hitung deviasi lokal

$$\sigma(i, j) = \sqrt{\sum_{k=-K}^{K} \sum_{l=-L}^{L} w_{k,l} (I_{k,l}(i, j) - \mu(i, j))^2}$$

```
def local_deviation(image, local_mean, kernel):
    "Vectorized approximation of local deviation"
    sigma = image ** 2
    sigma = signal.convolve2d(sigma, kernel, 'same')
    return np.sqrt(np.abs(local_mean ** 2 - sigma))
```

# Terakhir hitung koefisien MSCN

$$\hat{I}(i,\ j) = \frac{I(i,\ j) - \mu(i,\ j)}{\sigma(i,\ j) + C}$$

```
def calculate_mscn_coefficients(image, kernel_size=6, sigma=7/6):
  C = 1/255
  kernel = gaussian_kernel2d(kernel_size, sigma=sigma)
  local_mean = signal.convolve2d(image, kernel, 'same')
  local_var = local_deviation(image, local_mean, kernel)

  return (image - local_mean) / (local_var + C)
```

Untuk spektrum gambar terdistorsi yang lebih luas menggunakan koefisien MSCN didistribusikan sebagai Generalized Gaussian Distribution (GGD)

$$f(x;\alpha,\sigma^2)=\frac{\alpha}{2\beta\Gamma(1/\alpha)}e^{-\left(\frac{|x|}{\beta}\right)^{\alpha}}$$

dimana

$$\beta=\sigma\sqrt{\frac{\Gamma\left(\frac{1}{\alpha}\right)}{\Gamma\left(\frac{3}{\alpha}\right)}}$$

dan $\Gamma$ adalah fungsi Gamma

$\alpha$=mengontol bentuk

$\sigma^2$ = variasi

```
def generalized_gaussian_dist(x, alpha, sigma):
  beta = sigma * np.sqrt(special.gamma(1 / alpha) / special.gamma(3 / alpha))

  coefficient = alpha / (2 * beta() * special.gamma(1 / alpha))
  return coefficient * np.exp(-(np.abs(x) / beta) ** alpha)
```

# Pairwise products of neighboring MSCN coefficients

Model pairwise products of neighboring MSCN coefficients di sepanjang empat arah (1) horisontal *H*, (2) vertikal *V*, (3) diagonal utama *D1* dan (4) diagonal kedua *D2*.

$$D2(i,\ j) = \hat{I}(i,\ j)\hat{I}(i+1,\ j-1)$$

```
def calculate_pair_product_coefficients(mscn_coefficients):
    return collections.OrderedDict({
        'mscn': mscn_coefficients,
        'horizontal': mscn_coefficients[:, :-1] * mscn_coefficients[:, 1:],
        'vertical': mscn_coefficients[:-1, :] * mscn_coefficients[1:, :],
        'main_diagonal': mscn_coefficients[:-1, :-1] * mscn_coefficients[1:, 1:],
        'secondary_diagonal': mscn_coefficients[1:, :-1] * mscn_coefficients[:-1, 1:]
        })
```

# Model Asymmetric Generalized Gaussian Distribution (AGGD)

$$f(x;\nu,\sigma_l^2,\sigma_r^2)=\begin{cases}\dfrac{\nu}{(\beta_l+\beta_r)\Gamma\left(\frac{1}{\nu}\right)}e^{\left(-\left(\frac{-x}{\beta_l}\right)^{\nu}\right)} & x<0\\ \dfrac{\nu}{(\beta_l+\beta_r)\Gamma\left(\frac{1}{\nu}\right)}e^{\left(-\left(\frac{x}{\beta_r}\right)^{\nu}\right)} & x>=0\end{cases}$$

dimana

$$\beta_{side}=\sigma_{side}\sqrt{\frac{\Gamma\left(\frac{1}{\nu}\right)}{\Gamma\left(\frac{3}{\nu}\right)}}$$

side dapat dituliskan r atau ι

Parameter lain yang tidak ada dalam rumus sebelumnya adalah mean

$$\eta = (\beta_r - beta_l)\frac{\Gamma\left(\frac{2}{\nu}\right)}{\Gamma\left(\frac{1}{\nu}\right)}$$

```
def asymmetric_generalized_gaussian(x, nu, sigma_l, sigma_r):
  def beta(sigma):
    return sigma * np.sqrt(special.gamma(1 / nu) / special.gamma(3 / nu))

    coefficient = nu / ((beta(sigma_l) + beta(sigma_r)) * special.gamma(1 / nu))
    f = lambda x, sigma: coefficient * np.exp(-(x / beta(sigma)) ** nu)

    return np.where(x < 0, f(-x, sigma_l), f(x, sigma_r))
```

# Fitting Asymmetric Generalized Gaussian Distribution

```
def asymmetric_generalized_gaussian_fit(x):
```

Estimate $\alpha$ using the approximation of the inverse generalized Gaussian ratio.

$$\rho(\alpha) = \frac{\Gamma(2/\alpha)^2}{\Gamma(1/\alpha)\Gamma(3/\alpha)}$$

```
def estimate_phi(alpha):
  numerator = special.gamma(2 / alpha) ** 2
  denominator = special.gamma(1 / alpha) * special.gamma(3 / alpha)
  return numerator / denominator
```

# Calculate r̂

$$\hat{r} = \frac{\left(\frac{\sum |x_k|}{N_l + N_r}\right)^2}{\frac{\sum x_k^2}{N_l + N_r}}$$

```
def estimate_r_hat(x):
    size = np.prod(x.shape)
    return (np.sum(np.abs(x)) / size) ** 2 / (np.sum(x ** 2) / size)
```

Calculate $\hat{R}$ using $\hat{\gamma}$ and $\hat{r}$ estimations.

$$\hat{R} = \hat{r}\frac{(\hat{\gamma}^3 + 1)(\hat{\gamma} + 1)}{(\hat{\gamma}^2 + 1)^2}$$

```
def estimate_R_hat(r_hat, gamma):
  numerator = (gamma ** 3 + 1) * (gamma + 1)
  denominator = (gamma ** 2 + 1) ** 2
  return r_hat * numerator / denominator
```

```
def mean_squares_sum(x, filter = lambda z: z == z):
  filtered_values = x[filter(x)]
  squares_sum = np.sum(filtered_values ** 2)
  return squares_sum / ((filtered_values.shape))
```

Calculate $\hat{\gamma}$ where $N_l$ is the number of negative samples and $N_r$ is the number of positive samples.

$$\hat{\gamma} = \frac{\sqrt{\frac{1}{N_l - 1} \sum_{k=1,\ x_k<0}^{N_l} x_k^2}}{\sqrt{\frac{1}{N_r - 1} \sum_{k=1,\ x_k>=0}^{N_r} x_k^2}}$$

```
def estimate_gamma(x):
  left_squares = mean_squares_sum(x, lambda z: z < 0)
  right_squares = mean_squares_sum(x, lambda z: z >= 0)

  return np.sqrt(left_squares) / np.sqrt(right_squares)
```

```
def estimate_alpha(x):
  r_hat = estimate_r_hat(x)
  gamma = estimate_gamma(x)
  R_hat = estimate_R_hat(r_hat, gamma)

  solution = optimize.root(lambda z: estimate_phi(z) - R_hat, [0.2]).x

  return solution[0]
```

Estimate left and right scale parameters.

$$\sigma_l = \sqrt{\frac{1}{N_l - 1} \sum_{k=1,\ x_k<0}^{N_l} x_k^2}$$

$$\sigma_r = \sqrt{\frac{1}{N_r - 1} \sum_{k=1,\ x_k>=0}^{N_r} x_k^2}$$

```
def estimate_sigma(x, alpha, filter = lambda z: z < 0):
    return np.sqrt(mean squares sum(x, filter))
```

```
def estimate_mean(alpha, sigma_l, sigma_r):
  return (sigma_r - sigma_l) * constant * (special.gamma(2 / alpha) / special.gamma(1 / alpha))

alpha = estimate_alpha(x)
sigma_l = estimate_sigma(x, alpha, lambda z: z < 0)
sigma_r = estimate_sigma(x, alpha, lambda z: z >= 0)

constant = np.sqrt(special.gamma(1 / alpha) / special.gamma(3 / alpha))
mean = estimate_mean(alpha, sigma_l, sigma_r)

return alpha, mean, sigma_l, sigma_r
```

# Menghitung BRISQUE features

```
def calculate_brisque_features(image, kernel_size=7, sigma=7/6):
  def calculate_features(coefficients_name, coefficients, accum=np.array([])):
    alpha, mean, sigma_l, sigma_r = asymmetric_generalized_gaussian_fit(coefficients)

    if coefficients_name == 'mscn':
      var = (sigma_l ** 2 + sigma_r ** 2) / 2
      return [alpha, var]

    return [alpha, mean, sigma_l ** 2, sigma_r ** 2]

  mscn_coefficients = calculate_mscn_coefficients(image, kernel_size, sigma)
  coefficients = calculate_pair_product_coefficients(mscn_coefficients)

  features = [calculate_features(name, coeff) for name, coeff in coefficients.items()]
  flatten_features = list(chain.from_iterable(features))
  return np.array(flatten_features)
```

# Hands-on







**Setelah menghitung brisque feature, selanjutnya memperkirakan kualitas gambar. Gambar yang digunakan adalah gambar yang berasal dari datase Kodak.**

**Auxiliary Functions :**

```
def load_image(url):
  image_stream = request.urlopen(url)
  return skimage.io.imread(image_stream, plugin='pil')

def plot_histogram(x, label):
  n, bins = np.histogram(x.ravel(), bins=50)
  n = n / np.max(n)
  plt.plot(bins[:-1], n, label=label, marker='o')
```

# 1. Load image

```
%matplotlib inline
plt.rcParams["figure.figsize"] = 12, 9

url = 'http://www.cs.albany.edu/~xypan/research/img/Kodak/kodim05.png'
image = load_image(url)
gray_image = skimage.color.rgb2gray(image)

_ = skimage.io.imshow(image)
```


0
100
200
300
400
500
0
100
200
300
400
500
600
700
SUZUKI
CR
250
21
SMITH

# 2. Menghitung Koefisien

```
[ ]  %%time

     mscn_coefficients = calculate_mscn_coefficients(gray_image, 7, 7/6)
     coefficients = calculate_pair_product_coefficients(mscn_coefficients)
```

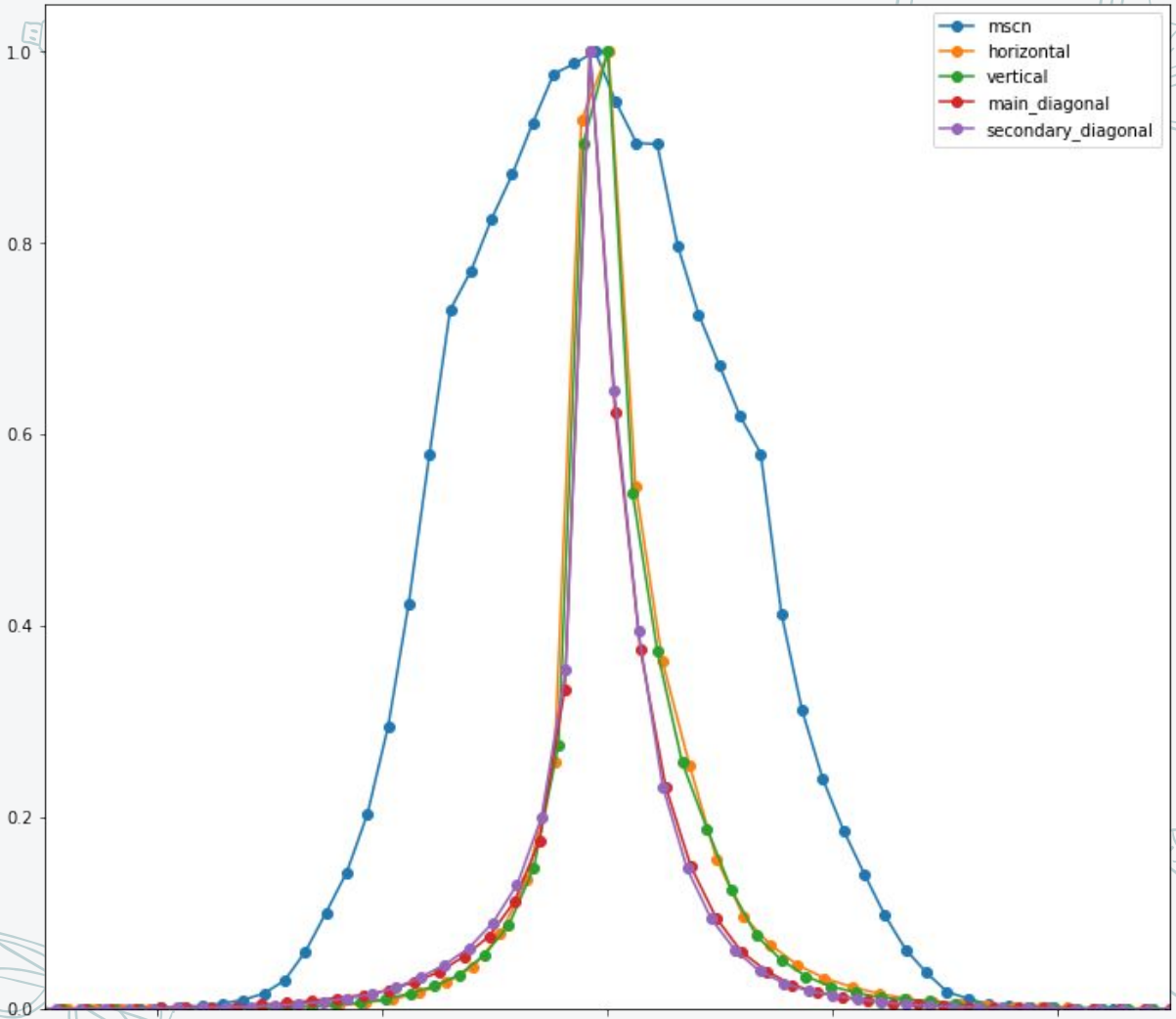
```
CPU times: user 232 ms, sys: 9.59 ms, total: 241 ms
Wall time: 261 ms
```

**Setelah menghitung koefisien MSCN dan produk berpasangan, kami dapat memverifikasi bahwa distribusi sebenarnya berbeda.**

```
%matplotlib inline
plt.rcParams["figure.figsize"] = 12, 11

for name, coeff in coefficients.items():
  plot_histogram(coeff.ravel(), name)

plt.axis([-2.5, 2.5, 0, 1.05])
plt.legend()
plt.show()
```


mscn
horizontal
vertical
main_diagonal
secondary_diagonal
1.0
0.8
0.6
0.4
0.2
0.0
−2
−1
0
1
2

# 3. Menyesuaikan Koefisien dengan Distribusi Gaussian Umum

```
[ ] %%time

    brisque_features = calculate_brisque_features(gray_image, kernel_size=7, sigma=7/6)
```

```
CPU times: user 231 ms, sys: 15.4 ms, total: 246 ms
Wall time: 250 ms
```

# 4. Mengubah Ukuran Gambar dan Menghitung BRISQUE Feature

```
[ ] %%time

    downscaled_image = cv2.resize(gray_image, None, fx=1/2, fy=1/2, interpolation = cv2.INTER_CUBIC)
    downscale_brisque_features = calculate_brisque_features(downscaled_image, kernel_size=7, sigma=7/6)

    brisque_features = np.concatenate((brisque_features, downscale_brisque_features))
```

```
CPU times: user 63 ms, sys: 6 ms, total: 69 ms
Wall time: 68.2 ms
```

## 4. Scale Features and Feed the SVR

Menggunakan model SVR pretrained untuk menghitung penilaian kualitas. Namun, untuk mendapatkan hasil yang baik, kita perlu mengatur skala fitur ke [-1, 1]

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')
     def scale_features(features):
       with open('/content/drive/My Drive/Colab Notebooks/normalize.pickle', 'rb') as handle:
         scale_params = pickle.load(handle)

       min_ = np.array(scale_params['min_'])
       max_ = np.array(scale_params['max_'])

       return -1 + (2.0 / (max_ - min_) * (features - min_))

     def calculate_image_quality_score(brisque_features):
       model = svmutil.svm_load_model('/content/drive/My Drive/Colab Notebooks/brisque_svm.txt')
       scaled_brisque_features = scale_features(brisque_features)

       x, idx = svmutil.gen_svm_nodearray(
           scaled_brisque_features,
           isKernel=(model.param.kernel_type == svmutil.PRECOMPUTED))

       nr_classifier = 1
       prob_estimates = (svmutil.c_double * nr_classifier)()

       return svmutil.libsvm.svm_predict_probability(model, x, prob_estimates)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Skala yang digunakan untuk mewakili kualitas gambar dari 0 hingga 100. Kualitas gambar 100 berarti kualitas gambarnya sangat buruk. Dalam hal gambar yang dianalisis, didapatkan bahwa itu adalah gambar yang berkualitas baik, hal ini dikarenakan gambar yang digunakan adalah gambar referensi.

```
[ ]  %%time

     calculate_image_quality_score(brisque_features)
```

```
CPU times: user 12.2 ms, sys: 874 µs, total: 13.1 ms
Wall time: 19.6 ms
4.954157281562374
```

## Kesimpulan

Metode ini diuji dengan database TID2008 dan berkinerja baik; bahkan dibandingkan dengan metode IQA.

Berikut Link Notebooknya :

https://colab.research.google.com/drive/1JBtvM9Z07y8ROlFN9D8j91SImIL0BOvx

# THANKS YOU