```
%ls
```

```
image/  im_h/  im_pose/  m_composite/  p_rendered/  README.md  shape/  try-on/
```

```
import torch
import torch.nn.functional as F
import numpy as np
import math
from PIL import Image
import cv2
from google.colab.patches import cv2_imshow
```

```
def gaussian(window_size, sigma):
    """
    Generates a list of Tensor values drawn from a gaussian distribution with standard
    diviation = sigma and sum of all elements = 1.

    Length of list = window_size
    """
    gauss =  torch.Tensor([math.exp(-(x - window_size//2)**2/float(2*sigma**2)) for x in
    return gauss/gauss.sum()
```

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

```
Distribution:  tensor([0.0010, 0.0076, 0.0360, 0.1094, 0.2130, 0.2660, 0.2130, 0.1094,
        0.0076, 0.0010])
Sum of Gauss Distribution: tensor(1.)
```

```
def create_window(window_size, channel=1):

    # Generate an 1D tensor containing values sampled from a gaussian distribution
    _1d_window = gaussian(window_size=window_size, sigma=1.5).unsqueeze(1)

    # Converting to 2D
    _2d_window = _1d_window.mm(_1d_window.t()).float().unsqueeze(0).unsqueeze(0)

    window = torch.Tensor(_2d_window.expand(channel, 1, window_size, window_size).contig

    return window
```

```
window = create_window(11, 3)
print("Shape of gaussian window:", window.shape)
```

```
Shape of gaussian window: torch.Size([3, 1, 11, 11])
```

```
def ssim(img1, img2, val_range, window_size=11, window=None, size_average=True, full=Fal

    L = val_range # L is the dynamic range of the pixel values (255 for 8-bit grayscale

    pad = window_size // 2

    try:
        _, channels, height, width = img1.size()
    except:
        channels, height, width = img1.size()

    # if window is not provided, init one
    if window is None:
        real_size = min(window_size, height, width) # window should be atleast 11x11
        window = create_window(real_size, channel=channels).to(img1.device)

    # calculating the mu parameter (locally) for both images using a gaussian filter
    # calculates the luminosity params
    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

    # now we calculate the sigma square parameter
    # Sigma deals with the contrast component
    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_sq
    sigma12 =  F.conv2d(img1 * img2, window, padding=pad, groups=channels) - mu12

    # Some constants for stability
    C1 = (0.01 ) ** 2  # NOTE: Removed L from here (ref PT implementation)
    C2 = (0.03 ) ** 2

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

    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)

    if size_average:
        ret = ssim_score.mean()
    else:
        ret = ssim_score.mean(1).mean(1).mean(1)
```

```
    if full:
        return ret, contrast_metric

    return ret
```

```
# helper function to load images
load_images = lambda x: np.asarray(Image.open(x).resize((480, 640)))

# Helper functions to convert to Tensors
tensorify = lambda x: torch.Tensor(x.transpose((2, 0, 1))).unsqueeze(0).float().div(255.

# display imgs
def display_imgs(x, transpose=True, resize=True):
  if resize:
    x=cv2.resize(x, (400, 400))
  if transpose:
    cv2_imshow(cv2.cvtColor(x, cv2.COLOR_BGR2RGB))
  else:
    cv2_imshow(x)
```

```
%ls
```

```
image/  im_h/  im_pose/  m_composite/  p_rendered/  README.md  shape/  try-on/
```

```
# The true reference Image
img1 = load_images("/content/Temp/image/000001_0.jpg")

# The False image
img2 = load_images("/content/Temp/try-on/000001_0.jpg")

# The noised true image
noise = np.random.randint(0, 255, (640, 480, 3)).astype(np.float32)
noisy_img = img1 + noise

print("True Image\n")
display_imgs(img1)

print("\nFalse Image\n")
display_imgs(img2)

print("\nNoised True Image\n")
display_imgs(noisy_img)
```

False Image

Noised True Image

```
# Check SSIM score of True image vs final output
_img1 = tensorify(img1)
_img2 = tensorify(img2)
true_vs_false = ssim(_img1, _img2, val_range=255)
print("True vs False Image SSIM Score:", true_vs_false)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: UserWarning: The given
  """
True vs False Image SSIM Score: tensor(0.7826)
```

```
# Check SSIM score of True image vs Noised_true Image
_img1 = tensorify(img1)
_img2 = tensorify(noisy_img)
true_vs_false = ssim(_img1, _img2, val_range=255)
print("True vs Noisy True Image SSIM Score:", true_vs_false)
```

```
True vs Noisy True Image SSIM Score: tensor(0.0407)
```

```
# Check SSIM score of True image vs True Image
_img1 = tensorify(img1)
true_vs_false = ssim(_img1, _img1, val_range=255)
print("True vs True Image SSIM Score:", true_vs_false)
```

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

```
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)
```

```
True vs Noised True Image SSIM Score: tensor(0.0407)
```

```
_img1 = tensorify(img1)
true_vs_false = ssim(_img1, _img1, val_range=255)
print("True vs True Image SSIM Score:", true_vs_false)
```

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

```
# Check SSIM score of True image vs False Image
_img1 = tensorify(img1)
_img2 = tensorify(img2)
true_vs_false = ssim(_img1, _img2, val_range=255)
print("True vs False Image SSIM Score:", true_vs_false)
```

```
True vs False Image SSIM Score: tensor(0.7826)
```

```
# The true reference Image
img3 = load_images("/content/Temp/image/000001_0.jpg")

# The False image
img4 = load_images("/content/Temp/try-on/000001_0.jpg")

print(img3)
print(img4)
```

```
[[[232 232 230]
  [232 232 230]
  [232 232 230]
  ...
  [232 234 231]
  [232 234 231]
  [232 234 231]]

 [[232 232 230]
  [232 232 230]
  [232 232 230]
  ...
  [232 234 231]
  [232 234 231]
  [232 234 231]]

 [[232 232 230]
  [232 232 230]
  [232 232 230]
  ...
  [232 234 231]
  [232 234 231]
  [232 234 231]]

 ...

 [[238 239 234]
  [238 239 234]
  [238 239 234]
  ...
```

```
       [227 227 225]
       [227 227 225]
       [227 227 225]]

      [[238 239 234]
       [238 239 234]
       [238 239 234]
       ...
       [227 227 225]
       [227 227 225]
       [227 227 225]]

      [[238 239 234]
       [238 239 234]
       [238 239 234]
       ...
       [227 227 225]
       [227 227 225]
       [227 227 225]]]
     [[[224 224 224]
       [224 224 224]
       [223 223 223]
       ...
       [214 214 216]
       [213 213 215]
       [213 213 215]]

      [[224 224 224]
```

```
from pathlib import Path

# s = set()
files = []
path = Path('/content/Temp/image')
set = ()
for file in path.iterdir():
    file = str(file)
    file2 = str(file)
    file2 = file2.replace('image','try-on')
    files.append([file,file2])

print(files)

```

```
[['/content/Temp/image/012385_0.jpg', '/content/Temp/try-on/012385_0.jpg'], ['/content/
```

```

MaxRes = 0.0
img1,img2 = '',''
for x in files:
  # print(x)
```

```
  _img1 = tensorify(load_images(x[0]))
  _img2 = tensorify(load_images(x[1]))
  true_vs_false = ssim(_img1, _img2, val_range=255)
  # print("True vs Try-on Image SSIM Score:", true_vs_false)
  if MaxRes < true_vs_false:
    MaxRes = true_vs_false
    img1 = x[0]
    img2 = x[1]

print(MaxRes)
print("Original Image\n")
display_imgs(load_images(img1))

print("\nTry-On Image\n")
display_imgs(load_images(img2))
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-63-8f004e5153e2> in <module>()
      6   _img1 = tensorify(load_images(x[0]))
      7   _img2 = tensorify(load_images(x[1]))
----> 8   true_vs_false = ssim(_img1, _img2, val_range=255)
      9   # print("True vs Try-on Image SSIM Score:", true_vs_false)
     10   if MaxRes < true_vs_false:

<ipython-input-29-1d2170e7bff3> in ssim(img1, img2, val_range, window_size, window, size_average, full)
     28     sigma1_sq = F.conv2d(img1 * img1, window, padding=pad, groups=channels) - mu1_sq
     29     sigma2_sq = F.conv2d(img2 * img2, window, padding=pad, groups=channels) - mu2_sq
---> 30     sigma12 =  F.conv2d(img1 * img2, window, padding=pad, groups=channels) - mu12
     31
     32     # Some constants for stability
```

```

  MaxRes1 = 0.0
  img3,img4 = '',''
  i = 1
  for x in files:
    # print(x)
    _img3 = tensorify(load_images(x[0]))
    _img4 = tensorify(load_images(x[1]))
    true_vs_false = ssim(_img3, _img4, val_range=255)
    print(f'True vs Try-on Image SSIM Score {i} :', true_vs_false)
    if MaxRes < true_vs_false:
      MaxRes = true_vs_false
      img3 = x[0]
      img4 = x[1]
    i+=1

  print(MaxRes1)
```

```
print("Original Image\n")
display_imgs(load_images(img3))

print("\nTry-On Image\n")
display_imgs(load_images(img4))
```

```
True vs Try-on Image SSIM Score 2028 : tensor(0.7906)
True vs Try-on Image SSIM Score 2029 : tensor(0.7793)
True vs Try-on Image SSIM Score 2030 : tensor(0.8262)
True vs Try-on Image SSIM Score 2031 : tensor(0.7659)
True vs Try-on Image SSIM Score 2032 : tensor(0.6334)
0.0
Original Image
```

Try-On Image