Homework 2 Video Shot Change Detection

E94071013 俞杉麒

1.Enviornment

Ubuntu 20.04.1

Python 3.8.10

Command line: **python hw2-1.py**(or **hw2-2.py**) **news**(or **ngc** or **ftfm**)

Revise the frame name to 0.jpg(jpeg), 1.jpg(jpeg), 2.jpg(jpeg), and so forth.

2.Used visual features and Shot change detection algorithm

The shot change detection algorithm is realized by examining the similarity between two continuous frames (i.e., test_img=*135.jpg* and data_img=*136.jpg*), which will be further compared to the threshold, in order to indicates the frame (i.e., data_img=*136.jpg*) whether it is a shot change or not.

hw2-1.py uses the *image-similarity-measures 0.3.5* package, which is implemented by several different algorithms. Here, we refer to the ensemble learning's voting process, where we choose 4 different algorithms from this package and assume there is a shot change if at least 3 algorithms match the threshold.

(a)*Root mean square error (RMSE)*:

Root mean square error (RMSE) measures the amount of change per pixel due to the processing. RMSE values are non-negative and a value of 0 means the image or videos being compared are identical. The RMSE between a reference or original image, image1—K and the enhanced or predicted image, image2—I(i, j) is given by:

$$RMSE = \sqrt{\frac{1}{M * N} \sum_{i=0,j=0}^{M-1,N-1} [I(i,j) - K(i,j)]^2}$$

(b)*Peak signal-to-noise ratio (PSNR)*:

Peak signal-to-noise ratio (PSNR) measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is usually expressed in terms of the logarithmic decibel scale. Typical values for the PSNR in lossy image and video compression in 8-bits are between 30 and 50 dB, where higher values are more desirable. For 16-bit data, typical values for the PSNR are between 60 and 80 dB.

To compute the PSNR, the package first computes the mean-squared error (MSE) using the following equation:

$$MSE = \frac{\sum_{M,N}[I_1(m,n) - I_2(m,n)]^2}{M * N}$$

In the previous equation, M and N are the number of rows and columns in the input images. The PSNR is subsequently calculated using the following equation:

$$PSNR = 10log_{10}(\frac{R^2}{MSE})$$

In the PSNR calculation, RR is the maximum possible pixel value of the image. Note that log transform is applied to have the value in decibels.

(c)*Structural Similarity Index (SSIM)*:

Structural Similarity Index (SSIM) quantifies image quality degradation caused by processing, such as data compression, or by losses in data transmission. SSIM is based on visible structures in the image. In order words, SSIM actually measures the perceptual difference between two similar images. The algorithm does not judge which of the two is better. However, that can be inferred from knowing which is the original image and which has been subjected to additional processing, such as data compression. The SSIM value is between -1 and 1 with 1 indicating perfect structural similarity.

The measure between two windows x and y:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

In the above equation, μ_x is the average of x ; μ_y is the average of y; σ_x is the variance of x; σ_y is the variance of y, σ_{xy} is the covariance of x and y; $c_1=(k_1L)^2$ and $c_2=(k_2L)^2$ are two variables that stabilize the division with a weak denominator, L is the dynamic range of the pixel values (typically this is $2^{no.ofbitsperpixel}-1$), and $k_1=0.01$, $k_2 = 0.03$ by default.

(d)*Signal to reconstruction error ratio (SRE)*:

Signal to reconstruction error ratio (SRE) was originally implemented in this paper (Charis Lanaras, José Bioucas-Dias, Silvano Galliani, Emmanuel Baltsavias, and Konrad Schindler "Super-resolution of Sentinel-2 images: Learning a globally applicable deep neural network") and it measures the error relative to the power of the signal. It's been stated in the paper that using SRE is better suited to make errors comparable between images of varying brightness. Whereas the popular PSNR would not achieve the same effect since the peak intensity is constant. SRE is computed as:

$$SRE = 10log_{10}\frac{\mu_x^2}{|\hat{x}-x|^2/n}$$

In the SRE equation σ_x is the average value of x. The values of SRE are given in decibels (dBs).

hw2-2.py uses 5 common algorithms for examining similarities between two images. Here, we find the average value with equal weights of the 5 similarities from 5 different algorithms (which will lie between 0 for least similar and 1 for most similar), and further compare it with the threshold.

(a)*Average Hash (aHash)*:
This approach crushes the image into a grayscale 8x8 image and sets the 64 bits in the hash based on whether the pixel's value is greater than the average color for the image.

(b)*Perceptive Hash pHash*:
This algorithm is similar to aHash but use a discrete cosine transform (DCT) and compares based on frequencies rather than color values.

(c)dHash
dHash Like aHash and pHash, dHash is pretty simple to implement and is far more accurate than it has any right to be. As an implementation, dHash is nearly identical to aHash but it performs much better. While aHash focuses on average values and pHash evaluates frequency patterns, dHash tracks gradients.

(1)Reduce size:
The fastest way to remove high frequencies and detail is to shrink the image. In this case, shrink it to 9x8 so that there are 72 total pixels. By ignoring the size and aspect ratio, this hash will match any similar picture regardless of how it is stretched.
(2)Reduce color:
Convert the image to a grayscale picture. This changes the hash from 72 pixels to

a total of 72 colors. (For optimal performance, either reduce color before scaling or perform the scaling and color reduction at the same time.)

(3)Compute the difference:

The dHash algorithm works on the difference between adjacent pixels. This identifies the relative gradient direction. In this case, the 9 pixels per row yields 8 differences between adjacent pixels. Eight rows of eight differences becomes 64 bits.

(4)Assign bits:

Each bit is simply set based on whether the left pixel is brighter than the right pixel. The order does not matter, just as long as you are consistent.

(d)*grayscale histogram comparison*

The histogram of an image with gray level in the range [0, L-1] is a discrete function

$$h(r_k) = n_k$$

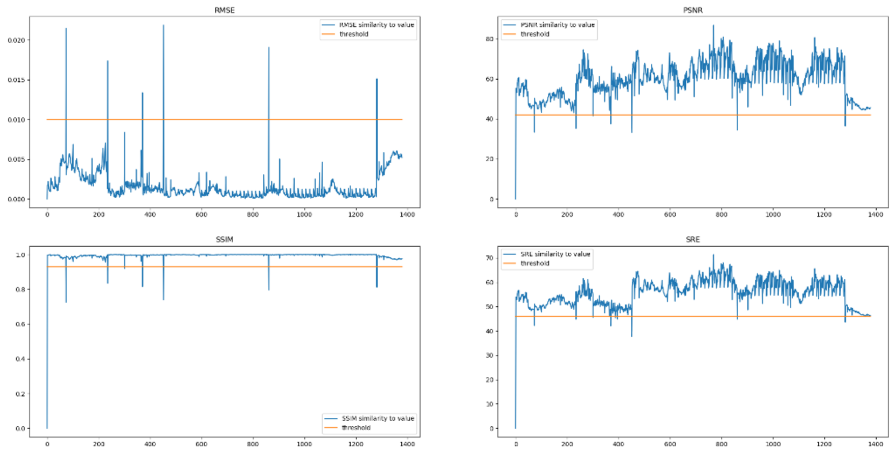
r_k is the kth gray level and n_k is the number of pixels in the image having gray level r_k

(e)*color histogram comparison*

Color histogram is a representation of the distribution of colors in an image. Discretize colors into a number of bins, and counting the number of pixels with colors in each bin.

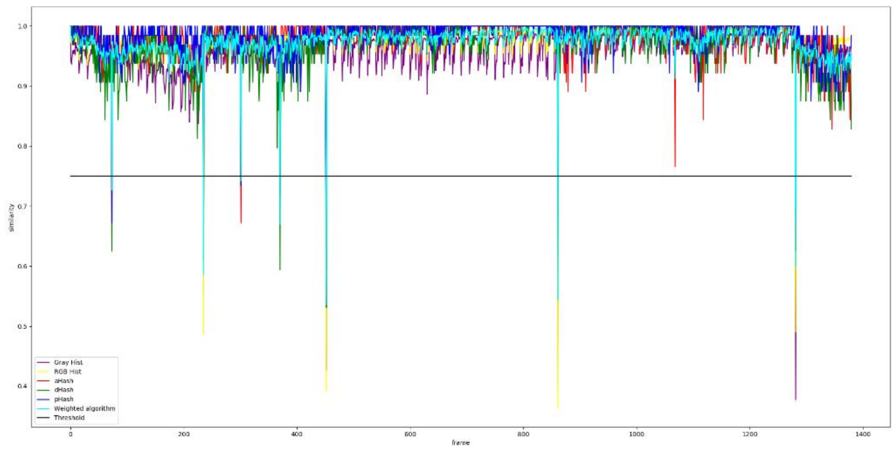
3. Effectiveness examples for detecting shot change

news 1380 frames	G.truth Shot changes	Detected Shot changes	False -ve	False +ve	Recall	Prec.
hw2-1.py	7	8	0	1	100%	87.50%



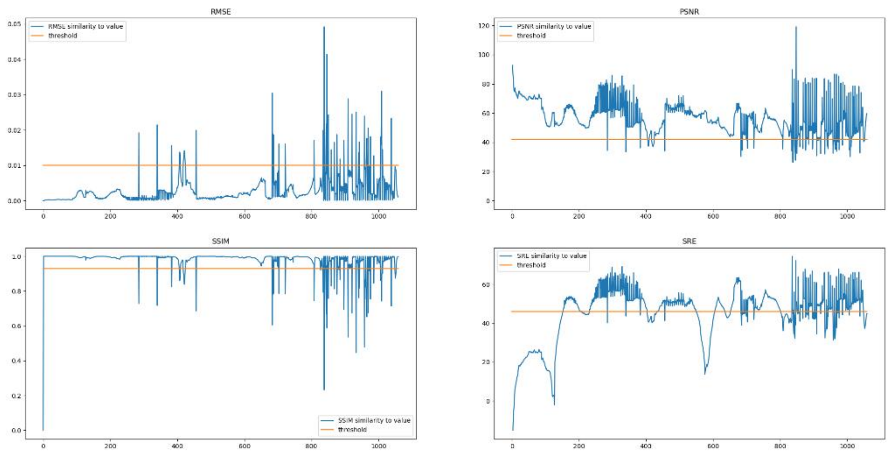
Threshold used for different algorithms:
RMSE = 0.01, PSNR = 42, SSIM = 0.93, SRE = 46

news 1380 frames	G.truth Shot changes	Detected Shot changes	False -ve	False +ve	Recall	Prec.
hw2-2.py	7	7	0	0	100%	100%



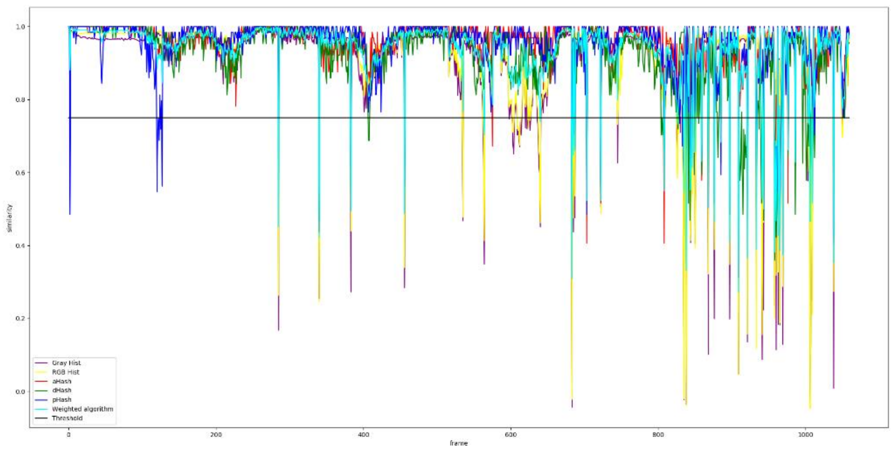
Threshold used : 0.75

ngc 1060 frames	G.truth Shot changes	Detected Shot changes	False -ve	False +ve	Recall	Prec.
hw2-1.py	36	41	7	12	80.56%	70.73%



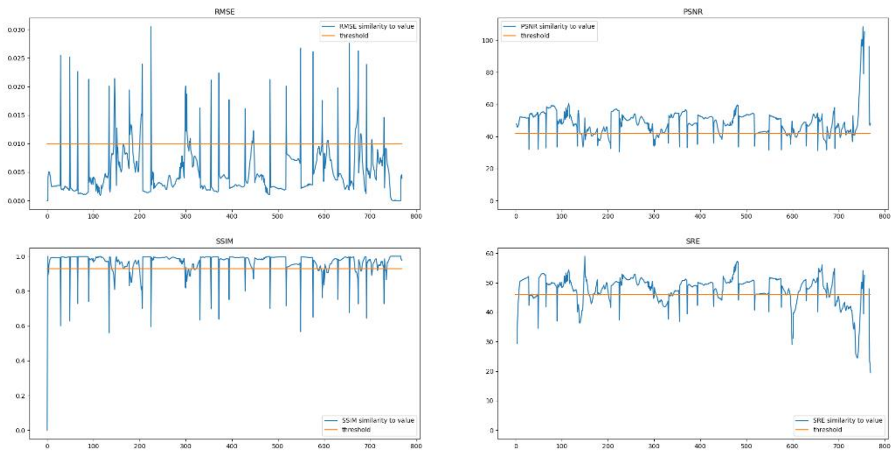
Threshold used for different algorithms:
RMSE = 0.01, PSNR = 42, SSIM = 0.93, SRE = 46

ngc 1060 frames	G.truth Shot changes	Detected Shot changes	False -ve	False +ve	Recall	Prec.
hw2-2.py	36	37	7	8	80.56%	78.38%



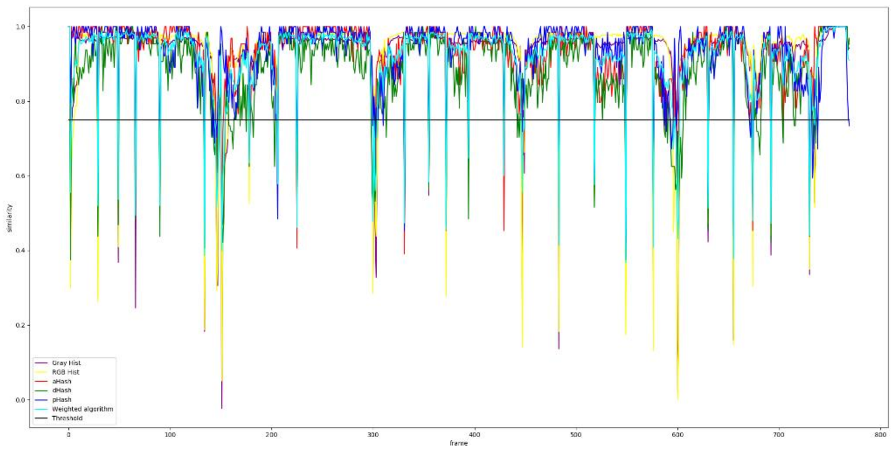
Threshold used : 0.75

ftfm 770 frames	G.truth Shot changes	Detected Shot changes	False -ve	False +ve	Recall	Prec.
hw2-1.py	27	77	2	52	92.59%	32.47%



Threshold used for different algorithms:
RMSE = 0.01, PSNR = 42, SSIM = 0.93, SRE = 46

ftfm 770 frames	G.truth Shot changes	Detected Shot changes	False -ve	False +ve	Recall	Prec.
hw2-2.py	27	31	0	4	100%	87.10%

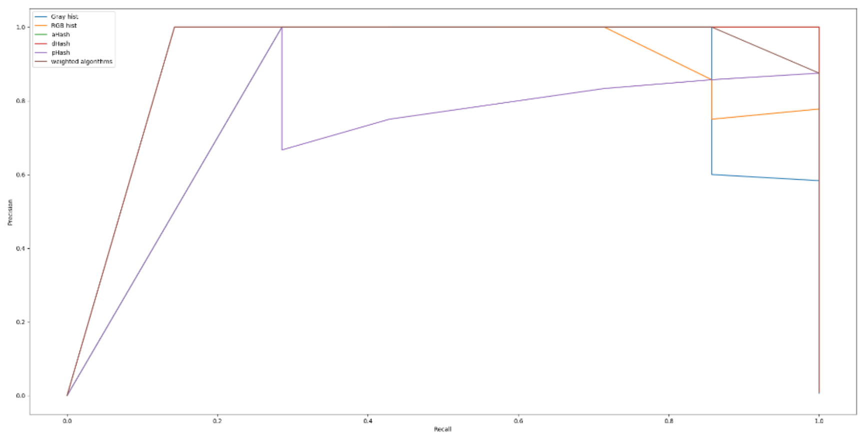


Threshold used : 0.75

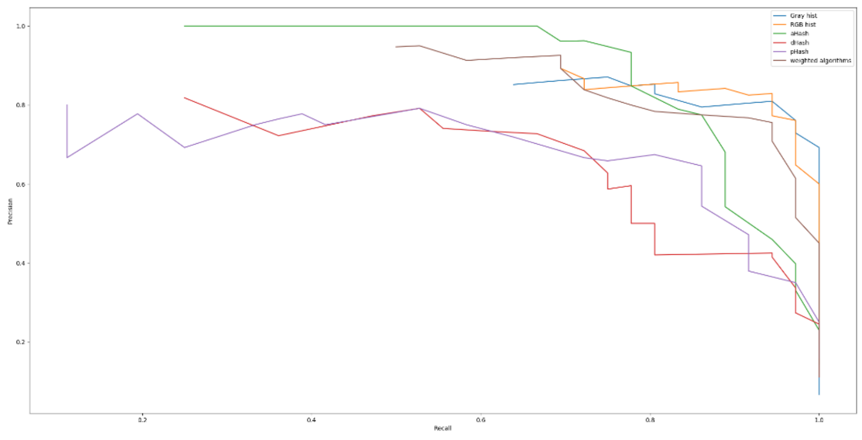
4.PR Curve of different thresholds in hw2-2.py

25 different thresholds with value = 0.5 ~ 0.98 (interval = 0.02)

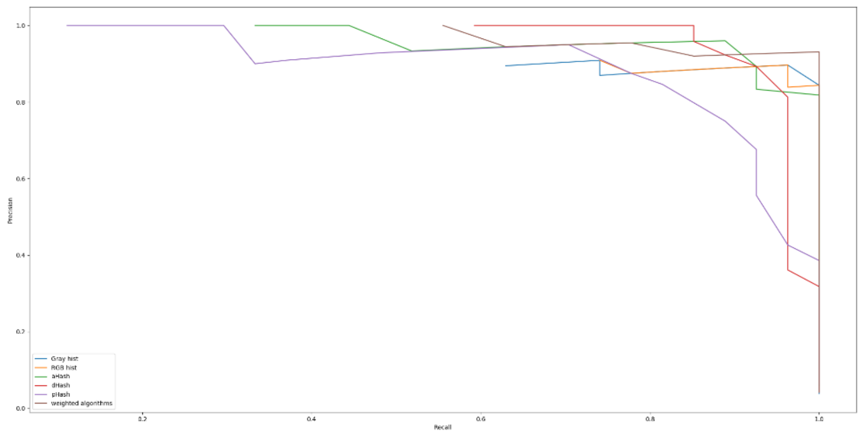
news:



ngc:



ftfm:



5.Ways to improve performance

(a)Optimize the threshold.

(b)Optimize the weights. Assign higher weight on the good algorithm, lower weight on algorithm that doesn’t perform well.

(c)Increase the frames used for detection, which leads to a larger number of frames in testing sets. As above, we had test_img=*135.jpg* and data_img=*136.jpg*, which we can improve the performance by increasing the testing sets to test the data_img between preframes (testpre_img) and postframes(testpost_img), which will ultimately lead to

testpre_img=*131.jpg, 132.jpg, 133.jpg, 134.jpg, 135.jpg*

testpost_img=*137.jpg, 138.jpg, 139.jpg, 140.jpg, 141.jpg*

and data_img=*136.jpg*.

We will consider it a shot change intuitively if the data_img matches better with postframes and matches worse with preframes.