**UPLOAD HOST IMAGE:**

Takes an image to resize it according to the frame size while maintaining the aspect ratio of the image.

**UPLOAD SECRET IMAGE:**

Takes an image to resize it according to the frame size while maintaining the aspect ratio of the image.

**ENCODE:**

* Takes the uploaded host and secret images.

**->Watermark embedding Process:**

* Divides the host image horizontally into 6 parts and vertically into 2 parts. So we have a total of 12 tiles.
* The images were divided into tiles by dividing the width by 6 and height by 2. As the image width and height are not every time completely divisible by 6 and 2 so we add the remainder in the last tiles so as we do not lose any pixels.
* Each tile is assigned to box\_x\_img (where x ranges from 0-11)
* Then these tiles are passed to a multiprocessing pool (size=12) created using the python multiprocessing library.
* Each process calls the processing function (parameters(starting\_left\_pixel\_of\_tile,starting\_top\_pixel\_of\_tile,width\_of\_tile,Height\_of\_tile))
* The processing function first load both the secret and host images in variables then iterate through all the pixels I for width j for height for both the secret and host images and convert the pixel values to binary.
* If the img1 pixels are in the range of img2 pixels then the merge\_rgb method is called in which the pixels of 2 images are merged in a way that the first 4 bits MSB are of the host images and the LSB 4 bits are of the black pixels “0000” otherwise the pixel value of img1 is assigned to the encoded image pixel(merged image)

if i < img2.size[0] and j < img2.size[1]:

rgb2 = Image\_Engine.\_\_int\_to\_bin(pixel\_map2[i, j])

rgb, ll = Image\_Engine.\_\_merge\_rgb(rgb1, rgb2)

else:

rgb = rgb1

* In the merge\_rgb function, only those pixels of img2 are merged with the tiles of img1 where img2 have black pixels and the white pixels are completely ignored. This helps in achieving lower MSE and performing well on other metrics as well.

r1, g1, b1 = rgb1

r2, g2, b2 = rgb2

if (r2 > '00111111' or g2 == '00111111' or b2 == '00111111'):

rgb = (r1,

g1,

b1)

ll = ('11111111', '11111111', '11111111')

else:

rgb = (r1[:4] + '0000',

g1[:4] + '0000',

b1[:4] + '0000')

ll = ('00000000', '00000000', '00000000')

1. Rgb1 is the host image tile
2. Rgb2 is a secret image

* Once the 2 images are merged it is called an encoded image then it is returned and displayed on the GUI.

**Watermark extraction :**

* Takes the encoded image saved as encoded\_i\_.png from the directory.
* Divides the encoded image horizontally into 6 parts and vertically into 2 parts. So we have a total of 12 tiles.
* The images were divided into tiles by dividing the width by 6 and height by 2. As the image width and height are not every time completely divisible by 6 and 2 so we add the remainder in the last tiles so as we do not lose any pixels.
* Each tile is assigned to box\_x\_img (where x ranges from 0-11)
* Then these tiles are passed to a multiprocessing pool (size=12) created using the python multiprocessing library.
* Each process calls the unmerg\_processing function (parameters(starting\_left\_pixel\_of\_tile,starting\_top\_pixel\_of\_tile,width\_of\_tile,Height\_of\_tile))
* The unmerg\_processing function iterate through all the pixels I for width j for height on the encoded image.
* Convert the pixels to binary format and separate those pixels which have “0000” in the last 4 bits.
* Takes these 4 bits and concatenates them with the other 4 bits of zeros “0000” so we have an 8-bit pixel which is black and then convert the binary 8-bit pixel back to integer format. By that method, we will have our secret image.
* While the “0000” secret\_image\_pixel removed from the encoded image are replaced with “0101” to keep the MSE low for the host image.

r, g, b = Image\_Engine.\_\_int\_to\_bin(pixel\_map[i, j])

if (r[4:] == '0000' and g[4:] == '0000' and b[4:] == '0000'):

rgb = (r[4:] + '0000',

g[4:] + '0000',

b[4:] + '0000')

hostimg = (r[:4] + '0101',

g[:4] + '0101',

b[:4] + '0101')

else:

rgb = ('11111111', '11111111', '11111111')

hostimg = (r, g, b)

Decipher - Text Part :

