\*Name: Pratap Shingane

PRN: 2020BTEIT00050

Course: Computer Algorithm

Procedure

Contractions

Let (E,d)(E,d) be a [complete metric space](https://en.wikipedia.org/wiki/Complete_metric_space) and f:E→Ef:E→E a mapping from EE to EE.

We will say that ff is a [contraction](https://en.wikipedia.org/wiki/Contraction_mapping) if there exists 0<s<10<s<1 such that:

∀x,y∈E,d(f(x),f(y))≤sd(x,y)∀x,y∈E,d(f(x),f(y))≤sd(x,y)

From here, ff will denote a contraction with contractivity factor ss.

There are two important theorems on contractions: the [Contraction Mapping Theorem](https://en.wikipedia.org/wiki/Banach_fixed-point_theorem) and the [Collage Theorem](https://en.wikipedia.org/wiki/Collage_theorem).

*Theorem (Contraction Mapping Theorem)*: ff has an unique fixed point x0x0.

From here, x0x0 will denote the fixed point of ff.

*Theorem (Collage Theorem)*: If d(x,f(x))<ϵd(x,f(x))<ϵ then d(x,x0)<ϵ1−sd(x,x0)<ϵ1−s.

The second theorem tells us that if we find a contraction ff such that f(x)f(x) is near to xx then we are sure that the fixed point of ff is also near to xx.

This result will be fundamental in the following. Indeed instead of saving an image, we will only save a contraction whose fixed point is near to the image.

Contractions for Images

In this part, we will show how to build a contraction such that its fixed point is near to a given image.

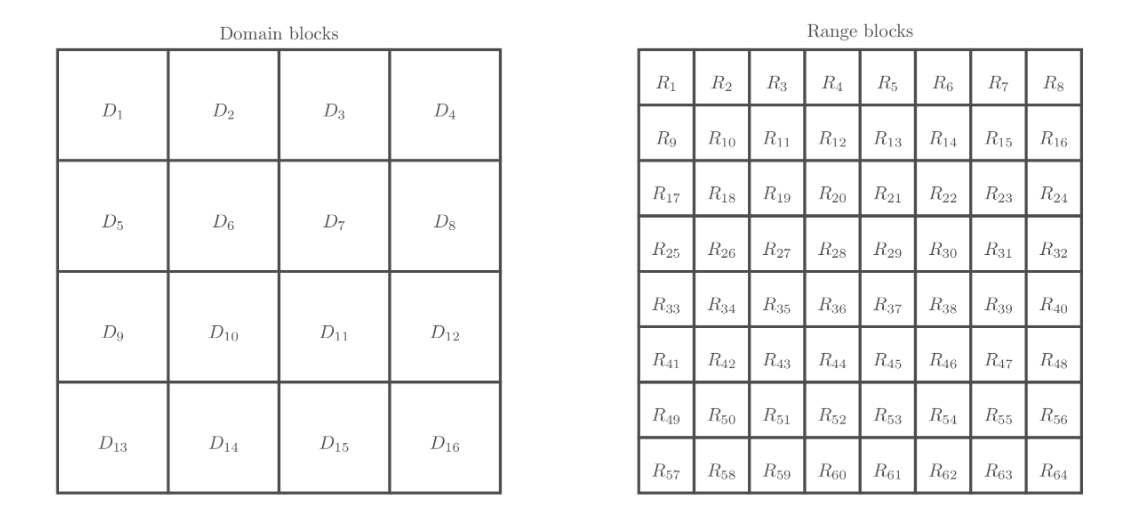
Firstly, let us define the image set and a distance. We choose E=[0,1]h×wE=[0,1]h×w. EE is the set of matrices with hh rows, ww columns and with coefficients in [0,1][0,1]. Then we take d(x,y)=(∑hi=1∑wj=1(xij−yij)2)0.5d(x,y)=(∑i=1h∑j=1w(xij−yij)2)0.5. dd is the distance obtained from the [Frobenius norm](https://en.wikipedia.org/wiki/Matrix_norm#Frobenius_norm).

Now, let x∈Ex∈E the image we want to compress.

We will segment twice the image in blocks:

* Firstly, we partition the image in *destination* or *range* blocks R1,...,RLR1,...,RL. These blocks are disjoint and they cover the whole image.
* Then, we segment the image in *source* or *domain* blocks D1,...,DKD1,...,DK. These blocks are not necessarily disjoint and neither they necessarily cover the image.

For instance, we can segment the image like this:



Then, for each range block RlRl, we will choose a domain block DklDkl and a mapping fl:[0,1]Dkl→[0,1]Rlfl:[0,1]Dkl→[0,1]Rl.

Finally, we can define our function ff as:

f(x)ij=fl(xDkl)ij if (i,j)∈Rlf(x)ij=fl(xDkl)ij if (i,j)∈Rl

Claim: If all flfl are contractions then ff is a contraction.

It remains one question to answer to: How to choose DklDkl and flfl?

The Collage Theorem suggests us a way to choose them: if xRlxRl is near to f(xDkl)f(xDkl) for all ll then xx is near to f(x)f(x) and according to the Collage Theorem xx and x0x0 will be near too.

Thus, we will, independently for each ll, construct lots of contractions from each DkDk to RlRl and select the best one. We will show all the nitty-gritty details in the next section.

# **Implementation**

## Segmentations

I keep things really simple. The source blocks and the destination blocks segment the image as a grid, as on the image above.

The size of the blocks are powers of two as it makes things easier. The source blocks are 8 by 8 while the destination blocks are 4 by 4.

There exist more advanced schemes for segmentation. For instance, we can use a quadtree to segment more the areas with lots of details.

## Transformations

In this section, we will show how to construct the contractions from DkDk to RlRl.

Remember that we want to generate a mapping flfl such that f(xDk)f(xDk) is near to xRlxRl. So, the more mappings we generate, the more likely we are to find a good one.

However, the quality of the compression depends on the number of bits necessary to save flfl. So if we have a set of functions that is too large, the compression will be bad. There is a tradeoff to find.

I chose that flfl will have the following form: fl(xDk)=s×rotateθ(flipd(reduce(xDk)))+bfl(xDk)=s×rotateθ(flipd(reduce(xDk)))+b

where reducereduce is a function to go from 8 by 8 blocks to 4 by 4 blocks, flipflip and rotaterotate are affine transformations, ss changes the contrast and bb the brightness.

The function reduce reduces the size of an image by averaging neighborhoods

The function rotate simply rotates by the given angle the image

In order to preserve the shape of the image, the angle θθ will be in {0∘,90∘,180∘,270∘}{0∘,90∘,180∘,270∘}.

The function flip flips the image if direction is equal to -1 and does not if it is equal to 1

The whole transformation is done by the function apply\_transformation

We need 1 bit to remember if we flip or not and 2 bits for the angle of rotation. Moreover, if we save ss and bb using 8 bits each then we need only 19 bits in total to save the transformation.

Moreover, we should check that these functions are contractions. The proof is a bit tedious and we do not care much. Maybe I will add it in appendix later.

## Compression

The algorithm for compression is simple. First we generate all possible affine transformations of all source blocks using the function generate\_all\_transformed\_blocks:

Then for each destination block, we try all the previously generated transformed source blocks. For each we optimize the contrast and the brightness using the method find\_contrast\_and\_brightness2 and if the tested transformation is the best we have seen so far, we saved it:

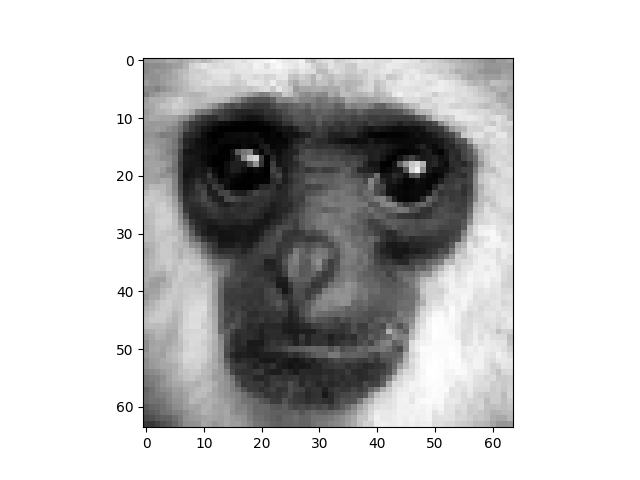
To find the best contrast and brightness, the method find\_contrast\_and\_brightness2 simply solves a least square problem

## Decompression

The decompression algorithm is even simpler. We just start from a completely random image and then we apply the contraction ff several times

It works because the contraction has a unique fixed point and whatever initial image we choose, we will tend to it.

I think, it is time for a little example. We will try to compress and decompress this image of a monkey:



The function test\_greyscale loads the image compresses it, decompresses and shows each iteration of the decompression

