Denoising Images

Applications of the Walsh-Hadamard Transform

Derek Wayne

13 April 2019

Abstract

The Hadamard transform is an example of a generalized Fourier transform. It has numerous applications in signal processing, data compression, and image/video coding to name a few. The Fast Walsh-Hadamard transform (FWHT) is an efficient algorithm for denoising images by using various methods of pixel thresholding. In this note, we explore how this might be implented on a noisy image.

Hadamard Matrices

For *Hadamard* matrices H_m and H_n we have the following useful recursive properties (note that since m and n are powers of 2, the matrices are symmetric),

$$H_m H_m^T = 2^k I \tag{1}$$

$$H_m^{-1} = \frac{1}{m} H_m \tag{2}$$

Proposition 1.

If we let $\hat{Z} = H_m Z H_n$, then $Z = H_m \hat{Z} H_n/(mn)$

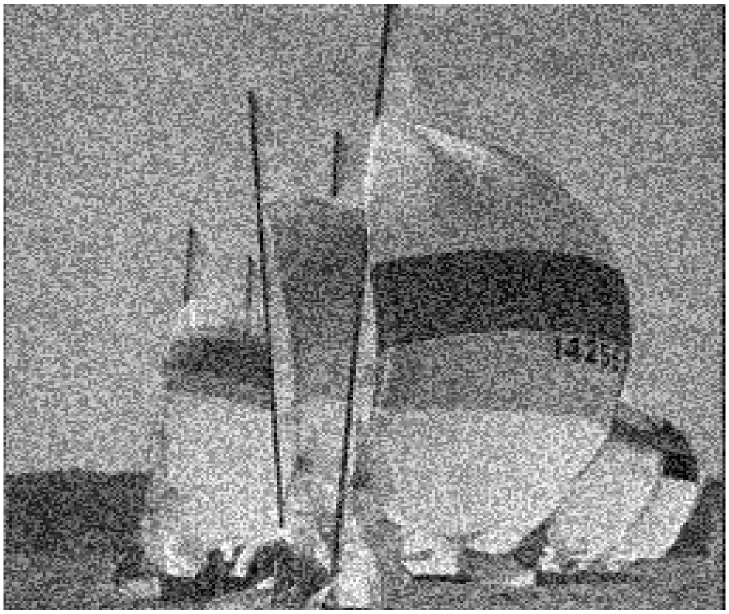
Proof

$$\begin{aligned} \hat{Z} &= H_m Z H_n \\ \iff (1/m)H_m\hat{Z}(1/n)H_n &= H_m^{-1}H_m Z H_n H_n^{-1} \\ \iff H_m\hat{Z}H_n/(mn) &= Z \end{aligned}$$

Loading in the image as a matrix:

```
boats <- matrix(scan("boats.txt"), ncol=256, byrow=T)
image(boats, axes=F, col=grey(seq(0,1,length=256)), main = "Original")
```

Original



Below is an implementation of the FWHT.

```
fwht2d <- function(x) {
  h <- 1
  length <- ncol(x)
  while (h < length) {
    for (i in seq(1, length, by=h*2)) {
      for (j in seq(i,i+h-1)) {
        a <- x[,j]
        b <- x[,j+h]
        x[,j] <- a + b
        x[,j+h] <- a - b
      }
    }
    h <- 2*h
  }
  h <- 1
  length <- nrow(x)
  while (h < length) {
    for (i in seq(1, length, by=h*2)) {
      for (j in seq(i,i+h-1)) {
        a <- x[j, ]
        b <- x[j+h, ]
        x[j, ] <- a + b
```

```
        x[j+h, ] <- a - b
      }
    }
    h <- 2*h
  }
  x
}
```

Below are the functions to perform soft and hard thresholding.

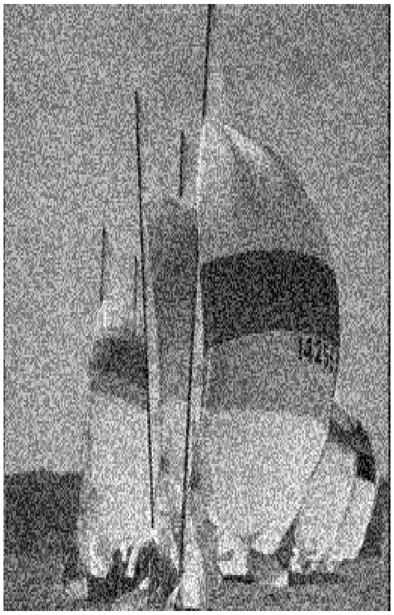
```
hard.thresh <- function(x, lambda) {ifelse(abs(x) > lambda, x, 0)}
soft.thresh <- function(x, lambda) {
  sign(x) * (abs(x) >= lambda) * (abs(x) - lambda)
}
```

The function below will divide the original image up into subimages depending on the function parameter "size"; referring to n where $n = 2^k, \quad 2 \leq k \leq 8$ so that the subimages are n by n pixel sub-images.

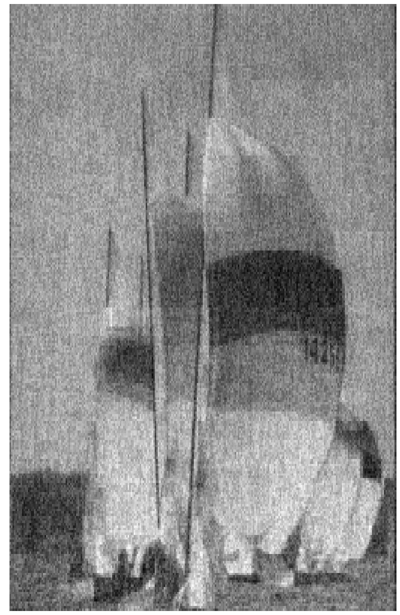
```
sub.im.transform <- function(image, size, Th.type="soft", thresh) {
  # image size must be square
  N <- ncol(image)

  if ((N %% size) != 0) {
    stop("image size is not divisible by specified sub-image size")
  }
  # create a list of submatrices to apply the transform to
  flat <- as.vector(image)
  subimages <- list()
  A <- N^2/size^2 # index span for each subimage
  for (j in 0:(A-1)) {
    subimages[[j+1]] <- matrix(flat[(j*(size^2)+1):((j+1)*(size^2))],ncol=size)
  }
  # apply transform to each sub-image and reconstruct the original matrix/image
  subimages.hat <- list()
  i <- 1
  for (im in subimages) {
    Zhat <- fwht2d(im)
    if (Th.type == "soft") {
      Zhat.star <- soft.thresh(Zhat, thresh)
    } else {
      Zhat.star <- hard.thresh(Zhat, thresh)
    }
    Zstar <- fwht2d(Zhat.star) / size
    subimages.hat[[i]] <- Zstar
    i <- i + 1
  }

  # reconstruct original image
  original <- c()
  for (i in 1:A) {
    original <- c(original, as.vector(subimages.hat[[i]]))
  }
  original <- matrix(original, ncol = 256)
  original
}
```

Results

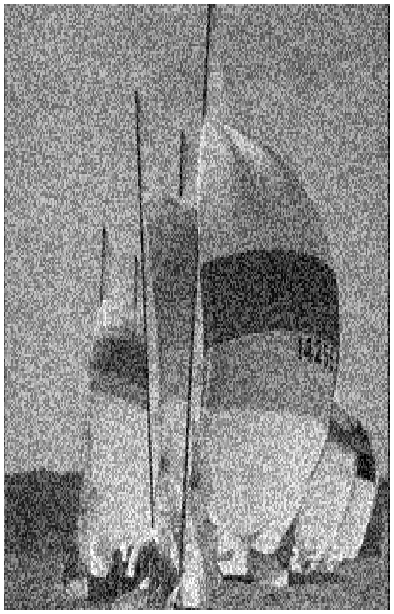
Original



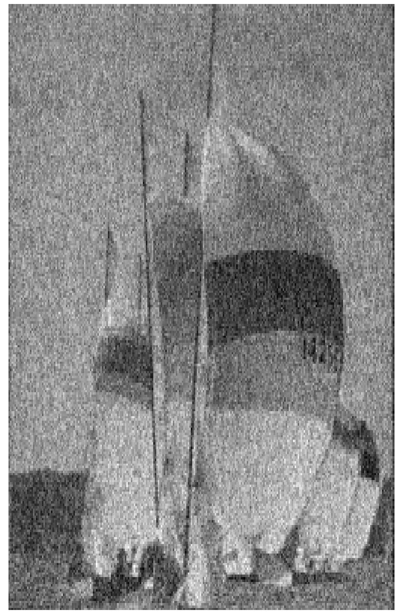
Soft Threshold: lam=7



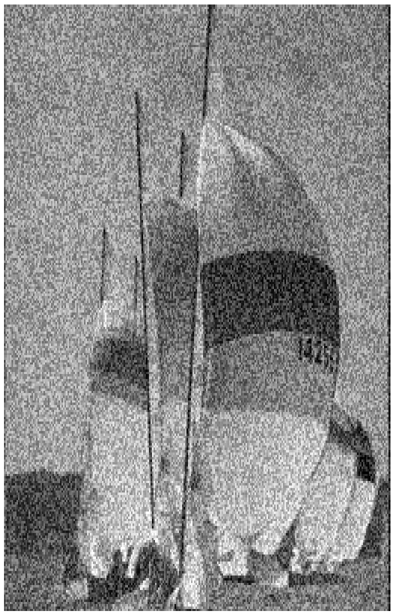
Original



Hard Threshold: lam=6



Original



Soft Threshold: lam=0.8