STAT 598 Project

Dan Tran

4/21/2020

Setup

Instal packages

```
#install.packages("png")
#install.packages("rlist")
#install.packages("pracma")
#install.packages("base")
#install.packages("doSNOW")
require(png)
require(rlist)
require(pracma)
require(base)
require(doSNOW)
```

Import image

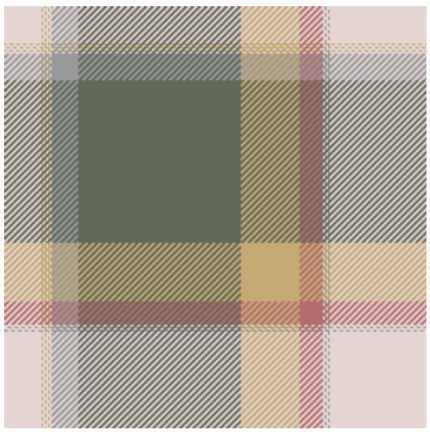
```
# Read file
my.img <- readPNG("IMG_0045.PNG")

# Check dimension
dim(my.img)
```

```
## [1] 252 252   4
```

Display image

```
# Create function to display imported PNG
# Modified codes from PNG packages documentation

display.img <- function(img){
  if (exists("rasterImage")) {
    y.val = dim(img)[1]
    x.val = dim(img)[2]

    plot(1:2, type='n', asp=(y.val/x.val),
         bty="n", axes=F, ylab="",xlab="") # remove borders

    if (names(dev.cur()) == "windows") {
      transparent <- img[,,4] == 0
      img <- as.raster(img[,,1:3])
      img[transparent] <- NA
      rasterImage(img, 1, 1, 2, 2, interpolate=FALSE)
    } else {
      rasterImage(img, 1, 1, 2, 2)
    }
  }
}


# Display
display.img(my.img)
title('Imported Image')
```

Imported Image



Measure patterns lengths

I modified the function a little bit but the main loop is the same. The result is the same from the old one.

Modification made: + Auto input my.pixel from image dimension + Add default values for my.length and want.length + Change i, i1 variables name

```
get.pattern <- function(img, my.length=100,want.length=50){
  my.pixels <- dim(img)[1]

  result <- matrix(c(NA,NA,NA))
  row.names(result) <- c("pixels","your length","client length")

  pixel.start=1
  pixel.end=1

  repeat{
    if(pixel.end==dim(img)[1]){
      result <- cbind(result,
                      c(pixel.end-pixel.start,
                        round((pixel.end-pixel.start)*my.length/my.pixels,1),
                        round((pixel.end-pixel.start)*want.length/my.pixels,1)))
      break
    }
    if(all(as.matrix(img[pixel.start:pixel.end,pixel.start:pixel.end,1])==
           as.matrix(img[pixel.start:pixel.end,pixel.start:pixel.end,1])[1,1])==FALSE){
      result <- cbind(result,
                      c(pixel.end-pixel.start,
                        round((pixel.end-pixel.start)*my.length/my.pixels,1),
                        round((pixel.end-pixel.start)*want.length/my.pixels,1)))
      pixel.start <- pixel.end
    }else{
      pixel.end = pixel.end+1
    }

  }
  return(result[,-1])
}


(pattern.length <- get.pattern(my.img))
```

```
##               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## pixels        22.0  1.0  1.0  1.0  1.0  1.0  1.0 16.0 97.0  35.0  14.0   1.0
## your length    8.7  0.4  0.4  0.4  0.4  0.4  0.4  6.3 38.5  13.9   5.6   0.4
## client length  4.4  0.2  0.2  0.2  0.2  0.2  0.2  3.2 19.2   6.9   2.8   0.2
##               [,13] [,14] [,15] [,16]
## pixels          1.0   1.0   1.0  57.0
## your length     0.4   0.4   0.4  22.6
## client length   0.2   0.2   0.2  11.3
```

Modify results

We can see that some of the columns have value of pixel equals 1, if we look at the image we can see there is some patterns there. So I created a function to combine those 1 pixel columns into one.

What it does is goes through all columns in the result above, if the pixel row value is 1 then look at next column, if it also 1 then combine 2 columns. Repeat until next column is not 1. If the pixel is not 1 then keep it.

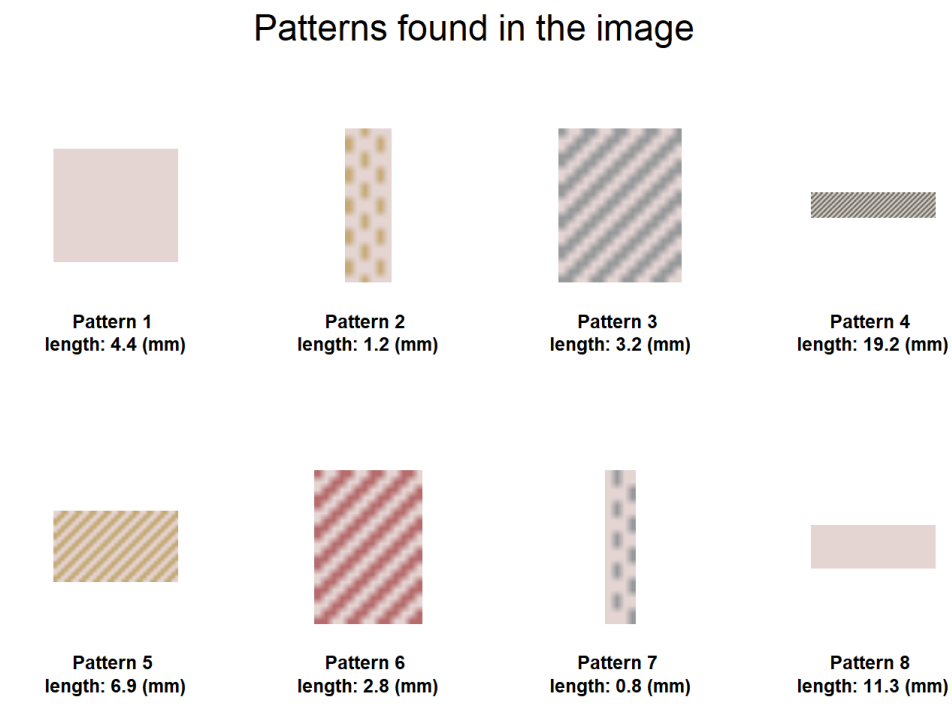
```
# Function to join 1 pixel columns
clean.result <- function(pattern){
  output = matrix(c(NA,NA,NA))
  c = 1
  while (c <= ncol(pattern)){
    if(pattern[1,c]==1){
      new.col <- pattern[,c]
      c = c +1
      while(pattern[1,c]==1){
        new.col <- new.col + pattern[,c]
        c = c+1
      }
      output <- cbind(output,unname(new.col))
    } else {
      output <- cbind(output,pattern[,c])
      c = c +1
    }
  }

  return(output[,-1])
}

# Run function
clean.result(pattern.length)
```

```
##               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## pixels        22.0  6.0 16.0 97.0 35.0 14.0  4.0 57.0
## your length    8.7  2.4  6.3 38.5 13.9  5.6  1.6 22.6
## client length  4.4  1.2  3.2 19.2  6.9  2.8  0.8 11.3
```

Display results

I think that it will be nice if we can show the patterns.

```
# Function to display recognized patterns
display.pattern <- function(patterns){
  patterns <- clean.result(patterns)
  pt.pixel <- patterns[1,]
  num.pat <- length(pt.pixel)
  num.row <- 2
  par(mfrow=c(num.row,(num.pat/num.row)),oma = c(0, 0, 2, 0))
  for (i in 1:length(pt.pixel)){
    v = pt.pixel[i]
    if (v > 1){
      #print(v)
      if (i == 1){
        pt.start <- 1
      } else{
        pt.start <- sum(pt.pixel[1:(i-1)],1)
      }
      pt.end <- pt.start + v -1
      #cat("start: ", pt.start," end: ",pt.end, "\n")
      display.img(my.img[1:20, pt.start:pt.end,])
      title(paste('Pattern', i, '\nlength:', patterns[3,i], '(mm)' ),line = -12)
    }
  }
  mtext("Patterns found in the image", outer = TRUE, cex = 1.5)

}

# Show patterns
display.pattern(pattern.length)
```



Parallel Computing

Main code

```
# Setup
cl <- makeCluster(2, type="SOCK") # 4 â□? number of cores
registerDoSNOW(cl) # Register Backend Cores for Parallel Computing
```

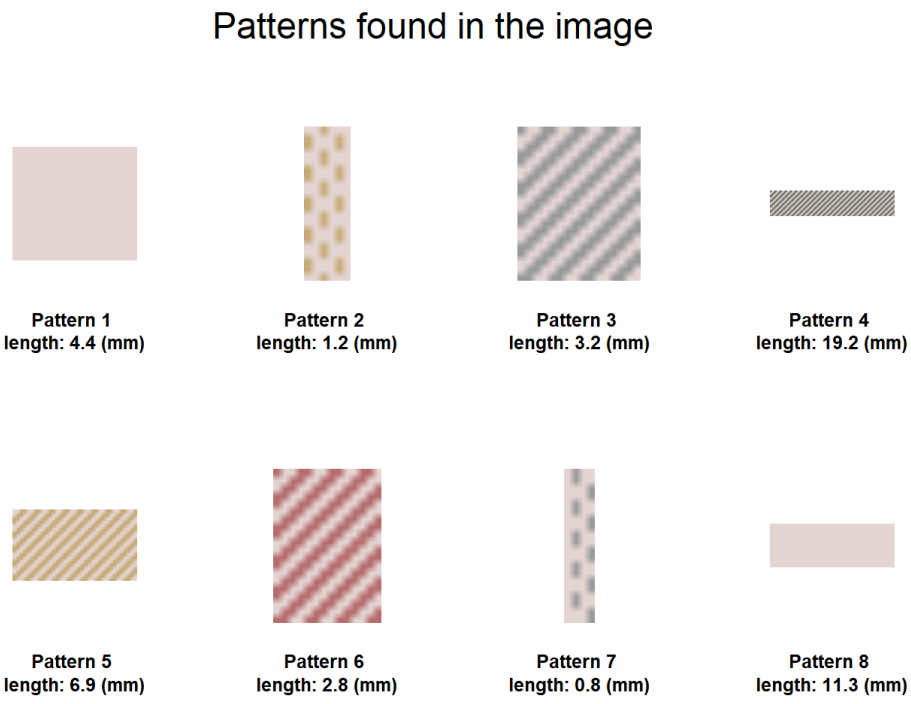
Since I don't have other images, I just make copy of the one above.

```
# Import new image
my.img2 <- my.img
my.img3 <- my.img
my.img4 <- my.img

# Put images into a list
all.imgs <- list(my.img, my.img2,my.img3,my.img4)

# Run in parallel
output_parallel <- foreach (img = all.imgs) %dopar% {
  pt <- get.pattern(img, my.length = 100, want.length = 50)
  clean.result(pt)
}

output_parallel
```

```
## [[1]]
##               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## pixels        22.0  6.0 16.0 97.0 35.0 14.0  4.0 57.0
## your length    8.7  2.4  6.3 38.5 13.9  5.6  1.6 22.6
## client length  4.4  1.2  3.2 19.2  6.9  2.8  0.8 11.3
##
## [[2]]
##               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## pixels        22.0  6.0 16.0 97.0 35.0 14.0  4.0 57.0
## your length    8.7  2.4  6.3 38.5 13.9  5.6  1.6 22.6
## client length  4.4  1.2  3.2 19.2  6.9  2.8  0.8 11.3
##
## [[3]]
##               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## pixels        22.0  6.0 16.0 97.0 35.0 14.0  4.0 57.0
## your length    8.7  2.4  6.3 38.5 13.9  5.6  1.6 22.6
## client length  4.4  1.2  3.2 19.2  6.9  2.8  0.8 11.3
##
## [[4]]
##               [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## pixels        22.0  6.0 16.0 97.0 35.0 14.0  4.0 57.0
## your length    8.7  2.4  6.3 38.5 13.9  5.6  1.6 22.6
## client length  4.4  1.2  3.2 19.2  6.9  2.8  0.8 11.3
```

Then we can display patterns from the result.

```
display.pattern(output_parallel[[2]])
```



Benchmark

Compare time cost using normal loop and parallel computing.

```
iter = 100 # number of runs

# Using loop

loop.time <- system.time(result <- for(run in 1:iter){
    for(img in all.imgs){
    pt <- get.pattern(img, my.length = 100, want.length = 50)
    clean.result(pt)
    }
  }
)

# Using parallel
par.time <- system.time( for(run in 1:iter){
    output_parallel <- foreach (img = all.imgs) %dopar% {
    pt <- get.pattern(img, my.length = 100, want.length = 50)
    clean.result(pt)
    }
  }
)
```

```
#Time using doSNOW:
print(par.time)
```

```
##    user  system elapsed
##    3.70    1.22    8.17
```

```
#Time using loop:
print(loop.time)
```

```
##    user  system elapsed
##    5.00    0.02    5.52
```

“User CPU time” gives the CPU time spent by the current process (i.e., the current R session)

“System CPU time” gives the CPU time spent by the kernel (the operating system) on behalf of the current process.

“Elapsed time” gives real time passed since the process was started.

Numbers are in seconds.

Problem: doSNOW has higher elaspsed time which should be lower.

```
# Stop using doSNOW
stopCluster(cl)
```