plot.cimg of 4d RGB array only plotting in grayscale. How do I get it to plot in color?

Asked 4 months ago Modified today Viewed 37 times Part of R Language Collective

1

I am attempting to plot an image that I created by combining two images using imager in R. I have the matrix as a cimg object with 3 color channels, and a depth channel, but it will only plot in greyscale for some reason even though it is supposed to default to rgb for these dimensions. How do I get it in color?

Here is my code. I am combining the SVD decompositions of two images. I don't think that is where my problem is. the problem may be with the format of "im" in the function svd4.

```
library(imager)
library(abind)
library(magick)
library(OpenImageR)
```

#any image can go here. feel free to replace it if you want to try. Size is rescaled so that doesn't matter.

```
image <- load.image('guernica2.png')
#image=imrotate(image,90)
h=dim(image)[1]
l=dim(image)[2]
red_image <- image[,,1]
green_image <- image[,,2]
blue_image <- image[,,3]
```

```
image2 = load.image("stars.png")
image2=resize(image2,h,l)
red_image2 <- image2[,,1]
green_image2 <- image2[,,2]
blue_image2 <- image2[,,3]
```

```
#get first p components from SVD
#image = face (testing input)
#RGB = 1 for red decomposition, 2 for green, 3 for blue
svd2 = function(image,p,RGB){
  image.svd = svd(image[,,RGB])
  s=image.svd$d[1:p]
  S=matrix(rep(0,length(s)^2),nrow=length(s))
  diag(S)=s
  V=image.svd$v[,1:p]
  U=image.svd$u[,1:p]
  im=U%*%S%*%t(V)
  return(im)
}
```

```
#this gets the last q singular values
#image here is the training (artist) image
```

```
svd3 = function(image,q,RGB){
  image.svd = svd(image[,,RGB])
  s=image.svd$d[q:length(image.svd$d)]
  S=matrix(rep(0,length(s)^2),nrow=length(s))
  diag(S)=s
  V=image.svd$v[,q:ncol(image.svd$v)]
  U=image.svd$u[,q:ncol(image.svd$u)]
  im=U%*%S%*%t(V)
  return(im)
}



#apply svd2 to each color channel and combine them
svd4 = function(image1, image2, p,q){
  R1 = svd2(image1,p,1)
  dim(R1) = c(dim(R1), 1)
  G1 = svd2(image1,p,2)
  dim(G1) = c(dim(G1), 1)
  B1 = svd2(image1,p,3)
  dim(B1) = c(dim(B1), 1)
  I1 = svd2(image1,p,4)
  dim(I1) = c(dim(I1), 1)
  I1 = matrix(rep(1,ncol(image1)*nrow(image1)),nrow=nrow(image1))
  dim(I1) = c(dim(I1), 1)
  train = abind(R1,G1,B1,I1,rev.along=0)
  R2 = svd3(image2,q,1)
  dim(R2) = c(dim(R2), 1)
  G2 = svd3(image2,q,2)
  dim(G2) = c(dim(G2), 1)
  B2 = svd3(image2,q,3)
  dim(B2) = c(dim(B2), 1)
  I2 = svd2(image2,p,4)
  I2=I1
  test = abind(R2,G2,B2,I2,rev.along=0)
  im = train + test
  im = as.cimg(im)
  plot(image1)
  #the line below is where I am having the problem. I have tried scaling in
  many ways, tried several functions for the colorscale argument, and more.
  Not sure what to do.
  plot(im)
  return(im)
}



im = svd4(image,image2,25,35)
```

I expected a colorized output of my synthesized image but it is greyscale. The images themselves plot in color no problem. Any ideas?

r image-processing svd dimensionality-reduction imager Edit tags

Share Edit Follow Close Flag

edited 1 min ago

Sandipan Dey

21.4k 2 49 63

asked Apr 13 at 20:27

Garrison Bullard

11 1

Sorted by:

1 Answer

Reset to default

Date modified (newest first)

0

Testing with the following couple of png images (with / without transparency channel),

```
image1 <- load.image('parrot.png')
dim(image1)
# [1] 453 340   1   3
```

```
image2 = load.image("lena.png")
image2=resize(image2,dim(image1)[1],dim(image1)[2])
dim(image2)
# [1] 453 340   1   4
```

Note that after the image is loaded with the function `load.image()`, it has 4 dimensions, not 3. Hence, the current implementation accesses the same color channel all the time, that's why it's outputting a grayscale image: check the following inside the function `svd4()`:

```
all(R1 == G1)
[1] TRUE
```

```
all(R1 == B1)
[1] TRUE
```

Hence, changing the functions `svd2()` and `svd3()` appropriately, so that it accesses the right color channel:

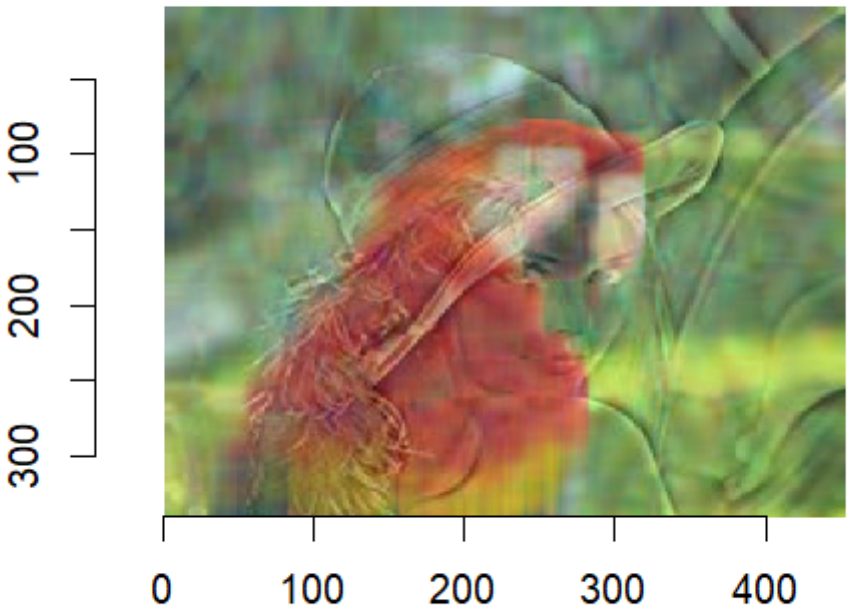
```
svd2 = function(image,p,RGB){
  image.svd = svd(image[,,,RGB]) # change the slicing here
  s=image.svd$d[1:p]
  S=matrix(rep(0,length(s)^2),nrow=length(s))
  diag(S)=s
  V=image.svd$v[,1:p]
  U=image.svd$u[,1:p]
  im=U%*%S%*%t(V)
  return(im)
}

svd3 = function(image,q,RGB){
  image.svd = svd(image[,,,RGB]) # change the slicing here
  s=image.svd$d[q:length(image.svd$d)]
  S=matrix(rep(0,length(s)^2),nrow=length(s))
  diag(S)=s
  V=image.svd$v[,q:ncol(image.svd$v)]
  U=image.svd$u[,q:ncol(image.svd$u)]
  im=U%*%S%*%t(V)
  return(im)
}

svd4 = function(image1, image2, p,q){
  R1 = svd2(image1,p,1)
  dim(R1) = c(dim(R1), 1)
  G1 = svd2(image1,p,2)
  dim(G1) = c(dim(G1), 1)
  B1 = svd2(image1,p,3)
  dim(B1) = c(dim(B1), 1)
  train = abind(R1,G1,B1,rev.along=0) # ignoring the transparency channel
  R2 = svd3(image2,q,1)
  dim(R2) = c(dim(R2), 1)
  G2 = svd3(image2,q,2)
  dim(G2) = c(dim(G2), 1)
  B2 = svd3(image2,q,3)
  dim(B2) = c(dim(B2), 1)
  test = abind(R2,G2,B2,rev.along=0) # ignoring the transparency channel
  im = train + test
  im = as.cimg(im)
  return(im)
}

im = svd4(image1,image2,10,12)
plot(im)
```

Here is the output produced, as desired (note that it contains the smooth / average components from image 1 and the noisy / edge components from image 2):



Share Edit Delete Flag

answered 4 mins ago

Sandipan Dey

21.4k 2 49 63