Practica5-1_BlazquezRamirezJavier

December 7, 2023

1 Blázquez Ramírez : 53903011G

2 Solución

Lo primero que se debe obtener son las fotos *cameraman* y *peppers*, para ello nos basaremos en el paquete ***TestImage***, el cual nos otorga ambas imágenes.

[ ]:
```
using Images, ImageView, TestImages, Colors, Plots, Interpolations,␣
↪ImageTransformations
```

[ ]:
```
img1 = testimage("cameraman")
```

[ ]:
```
img2 = testimage("peppers")
```

Ahora tenemos ya ambas fotos disponibles para trabajar con ellas. La imagen 1 es la llamada *cameraman* y la segunda *peppers*.

Ahora calculemos su resolución espacial de ambas imágenes.

Tenemos que la reslución espacial de las imágenes son:

[ ]:
```
resE_img1 = size(img1)
println("La resolución de la imagen 1 es", resE_img1 )
```

```
La resolución de la imagen 1 es(512, 512)
```

[ ]:
```
resE_img2 = size(img2)
println("La resolución de la imagen 2 es", resE_img2 )
```

```
La resolución de la imagen 2 es(512, 512)
```

Ahora calcularemos el tamaño de las imágenes. Primero tenemos que saber como interpreta Julia las imágenes.

En Julia, cada imagen es una matriz bidimensional. Para el caso de una imagen en escala de grises, se tendrá una sola matriz, con cada valor de la intensidad. Sin embargo, para una imagen en color, se tendrán tres matrices, una para cada color RGB.

De esta manera, una imagen en escala de grises será del tamaño de una matriz, y la imagen en color, tendrá el tamaño de 3 matrices.

Las imágenes en Julia son mostradas teniendo en cuenta que el 0 significa *negro* y el 1 significa *blanco*. Esto se aplica también a los píxeles, si los codificamos en rango de 0 a 255. Julia utiliza un tipo especial, llamado ***N0f8***, la cual representa un entero de 8-bit.

Sabiendo esto, podemos calcular ahora el tamaño.

[ ]:
```
tamB_img1 = sizeof(img1);
tamB_img2 = sizeof(img2);
tamB_img2 / tamB_img1
```

```
3.0
```

Vemos que la relación entre ambos tamaños es de 3. Esto ocurre porque la imagen de color tiene tres veces más canales que la imagen en escala de grises.

Este tamaño calculado no son bits ni bytes. Son píxeles totales en la imagen. Ahora, debemos hacer la conversión a bits y a bytes.

Cada pixel contiene un tipo que equivale a un entero de 8 bytes, por tanto, el tamaño de la imagen será el número de píxeles total de todos los canales multiplicado por 8.

[ ]:
```
tamB_img1 = sizeof(img1) * 8 # pixels * byte/pixels = bit
```

```
2097152
```

[ ]:
```
tamB_img2 = sizeof(img2) * 8 # pixels * byte/pixels = bit
```

```
6291456
```

Ahora convertimos a kiloBytes debido a que con esta unidad estamos más acostumbrados.

[ ]:
```
println("El tamaño de la imagen 1 es ", tamB_img1 / 1000, " kiloBytes" )
```

```
El tamaño de la imagen 1 es 2097.152 kiloBytes
```

[ ]:
```
println("El tamaño de la imagen 2 es ", tamB_img2 / 1000, " kiloBytes" )
```

```
El tamaño de la imagen 2 es 6291.456 kiloBytes
```

Ahora pasemos la imagen dos, a escala de grises:

[ ]:
```
img2_g = Gray.(img2)
```

Para obtener los canales por separado los canales RGB de la imagen 2 los obtenemos de la siguiente manera.

[ ]:
```
R_img2 = red.(img2)
```

```
512×512 Array{N0f8,2} with eltype N0f8:
 0.396  0.549  0.592  0.627  0.631  …  0.655  0.71   0.753  0.804  0.471
 0.482  0.749  0.745  0.729  0.714     0.596  0.553  0.604  0.533  0.451
 0.494  0.725  0.725  0.722  0.702     0.62   0.584  0.565  0.58   0.443
 0.482  0.733  0.729  0.718  0.729     0.659  0.584  0.592  0.604  0.463
 0.498  0.733  0.714  0.757  0.722     0.608  0.639  0.58   0.592  0.463
 0.486  0.745  0.729  0.706  0.722  …  0.647  0.6    0.592  0.612  0.463
 0.467  0.722  0.718  0.753  0.729     0.631  0.62   0.6    0.569  0.459
 0.471  0.725  0.745  0.722  0.741     0.627  0.639  0.584  0.596  0.451
```

```
 0.455  0.765  0.733  0.765  0.725     0.647  0.612  0.624  0.608  0.451
 0.424  0.722  0.753  0.698  0.741     0.608  0.624  0.58   0.659  0.439

 0.478  0.557  0.592  0.494  0.545     0.718  0.733  0.722  0.745  0.722
 0.471  0.592  0.545  0.541  0.471     0.729  0.729  0.753  0.737  0.749
 0.463  0.549  0.529  0.494  0.498  …  0.71   0.749  0.741  0.741  0.741
 0.4    0.478  0.502  0.518  0.435     0.733  0.686  0.718  0.725  0.714
 0.396  0.471  0.514  0.514  0.416     0.694  0.737  0.682  0.694  0.718
 0.361  0.549  0.506  0.529  0.494     0.725  0.675  0.729  0.671  0.749
 0.361  0.498  0.533  0.506  0.455     0.722  0.733  0.71   0.694  0.741
 0.361  0.557  0.494  0.494  0.502  …  0.725  0.675  0.686  0.69   0.769
 0.365  0.467  0.545  0.533  0.553     0.667  0.671  0.651  0.671  0.776
```

[ ]:
```
G_img2 = green.(img2)
```

```
512×512 Array{N0f8,2} with eltype N0f8:
 0.0  0.0    0.0    0.0    0.0    0.0    …  0.0    0.0    0.0    0.0    0.0
 0.0  0.376  0.341  0.361  0.361  0.361     0.776  0.773  0.792  0.737  0.753
 0.0  0.4    0.365  0.322  0.345  0.329     0.792  0.776  0.784  0.784  0.765
 0.0  0.314  0.392  0.333  0.349  0.345     0.8    0.769  0.78   0.792  0.769
 0.0  0.357  0.424  0.361  0.333  0.376     0.769  0.796  0.792  0.78   0.784
 0.0  0.345  0.38   0.318  0.341  0.329  …  0.796  0.776  0.784  0.769  0.773
 0.0  0.325  0.314  0.341  0.275  0.345     0.796  0.773  0.761  0.757  0.773
 0.0  0.333  0.353  0.29   0.329  0.349     0.8    0.776  0.784  0.773  0.78
 0.0  0.231  0.322  0.353  0.306  0.345     0.796  0.788  0.776  0.788  0.78
 0.0  0.325  0.361  0.314  0.333  0.345     0.78   0.792  0.773  0.784  0.769

 0.0  0.62   0.58   0.604  0.565  0.537     0.82   0.816  0.839  0.863  0.816
 0.0  0.635  0.604  0.553  0.541  0.549     0.812  0.827  0.851  0.847  0.804
 0.0  0.569  0.561  0.486  0.443  0.416  …  0.808  0.827  0.831  0.835  0.835
 0.0  0.557  0.529  0.525  0.412  0.404     0.835  0.808  0.808  0.827  0.835
 0.0  0.537  0.569  0.533  0.349  0.443     0.835  0.839  0.796  0.8    0.812
 0.0  0.525  0.584  0.553  0.451  0.455     0.839  0.78   0.855  0.792  0.835
 0.0  0.537  0.569  0.51   0.471  0.439     0.851  0.859  0.835  0.824  0.773
 0.0  0.486  0.451  0.525  0.447  0.427  …  0.867  0.8    0.827  0.808  0.647
 0.0  0.31   0.529  0.561  0.643  0.443     0.8    0.8    0.78   0.804  0.784
```

[ ]:
```
B_img2 = blue.(img2)
```

```
512×512 Array{N0f8,2} with eltype N0f8:
 0.0  0.557  0.573  0.545  0.58   0.522  …  0.573  0.573  0.62   0.694  0.71
 0.0  0.18   0.2    0.169  0.212  0.22      0.333  0.349  0.333  0.337  0.333
 0.0  0.196  0.188  0.188  0.224  0.169     0.318  0.345  0.357  0.302  0.341
 0.0  0.184  0.216  0.157  0.2    0.208     0.337  0.365  0.318  0.337  0.349
 0.0  0.212  0.22   0.208  0.184  0.176     0.325  0.341  0.341  0.322  0.329
 0.0  0.184  0.184  0.176  0.231  0.188  …  0.337  0.349  0.333  0.325  0.329
 0.0  0.208  0.176  0.149  0.2    0.161     0.369  0.369  0.341  0.325  0.325
 0.0  0.18   0.169  0.149  0.161  0.2       0.329  0.369  0.318  0.325  0.322
```

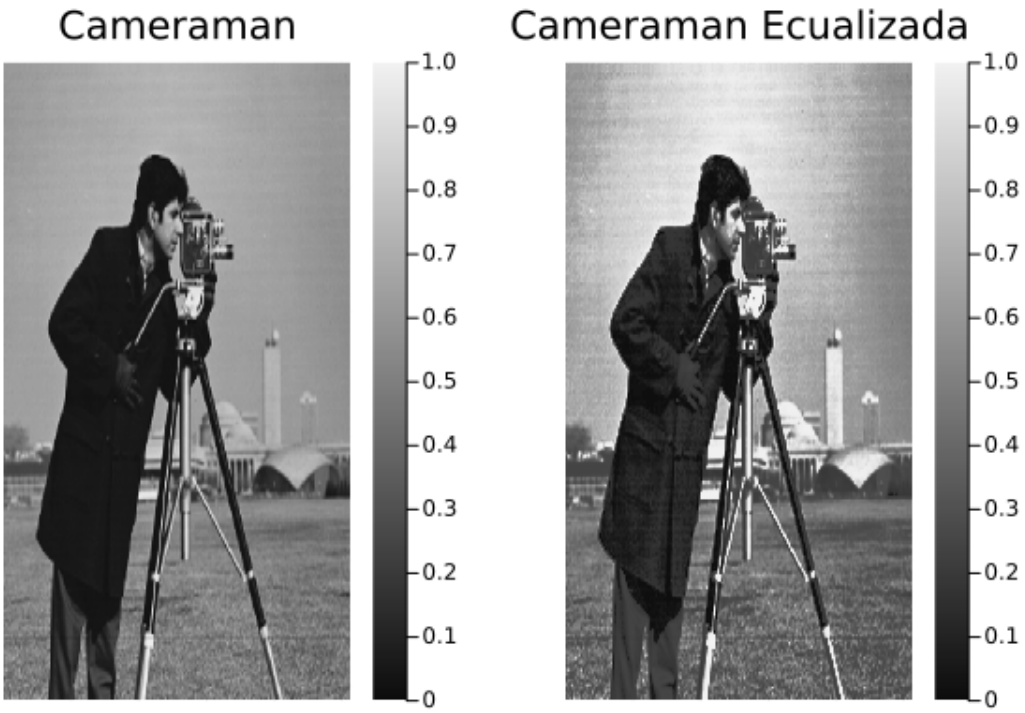
```
0.0  0.173  0.161  0.169  0.22   0.161     0.333  0.345  0.329  0.314  0.349
0.0  0.18   0.165  0.153  0.141  0.188     0.337  0.337  0.314  0.302  0.333

0.0  0.196  0.227  0.208  0.192  0.176     0.612  0.639  0.682  0.69   0.655
0.0  0.231  0.216  0.224  0.196  0.192     0.631  0.686  0.682  0.659  0.659
0.0  0.239  0.184  0.208  0.165  0.145  …  0.643  0.667  0.643  0.639  0.682
0.0  0.184  0.259  0.176  0.169  0.145     0.71   0.733  0.604  0.639  0.604
0.0  0.173  0.2    0.212  0.129  0.188     0.667  0.671  0.702  0.627  0.643
0.0  0.196  0.235  0.22   0.169  0.188     0.663  0.624  0.682  0.651  0.706
0.0  0.184  0.212  0.188  0.169  0.204     0.71   0.694  0.682  0.667  0.506
0.0  0.204  0.188  0.176  0.184  0.18   …  0.651  0.639  0.639  0.667  0.459
0.0  0.141  0.227  0.224  0.282  0.231     0.635  0.6    0.588  0.71   0.671
```

Una vez tenemos los canales de las imágenes, podemos trabajar con ellas. Una cosa que podemos hacer sería ecualizar los canales de la imagen.

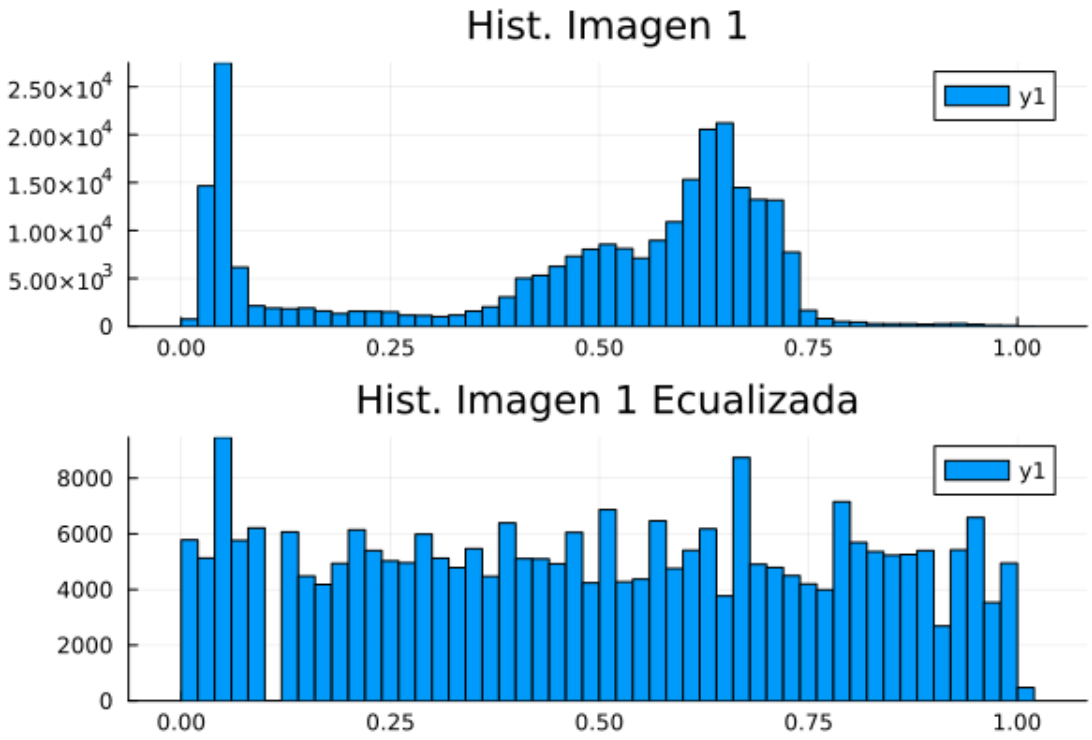
Para la ecualización, nos basaremos en la función para ecualizar el histograma proporcionada por el paquete *Images*, la función es *adjust_histogram*.

[ ]:
```
img1_n = adjust_histogram(img1, Equalization(nbins = 256, minval = 0, maxval =␣
 ↪1))
println("Son ambas imágenes iguales: ",img1 == img1_n)
plot(
    heatmap(channelview(img1), color=:grays, axis=false, title="Cameraman"),
    heatmap(channelview(img1_n), color=:grays, axis=false, title="Cameraman␣
 ↪Ecualizada"),
    yflip=true,
    layout=(1, 2)
)
```

```
Son ambas imágenes iguales: false
```



[ ]:
```
plot(
    histogram(vec(Float64.(img1)),bins=50,title="Hist. Imagen 1"),
    histogram(vec(Float64.(img1_n)),bins=50,title="Hist. Imagen 1 Ecualizada"),
    layout=(2,1)
)
```

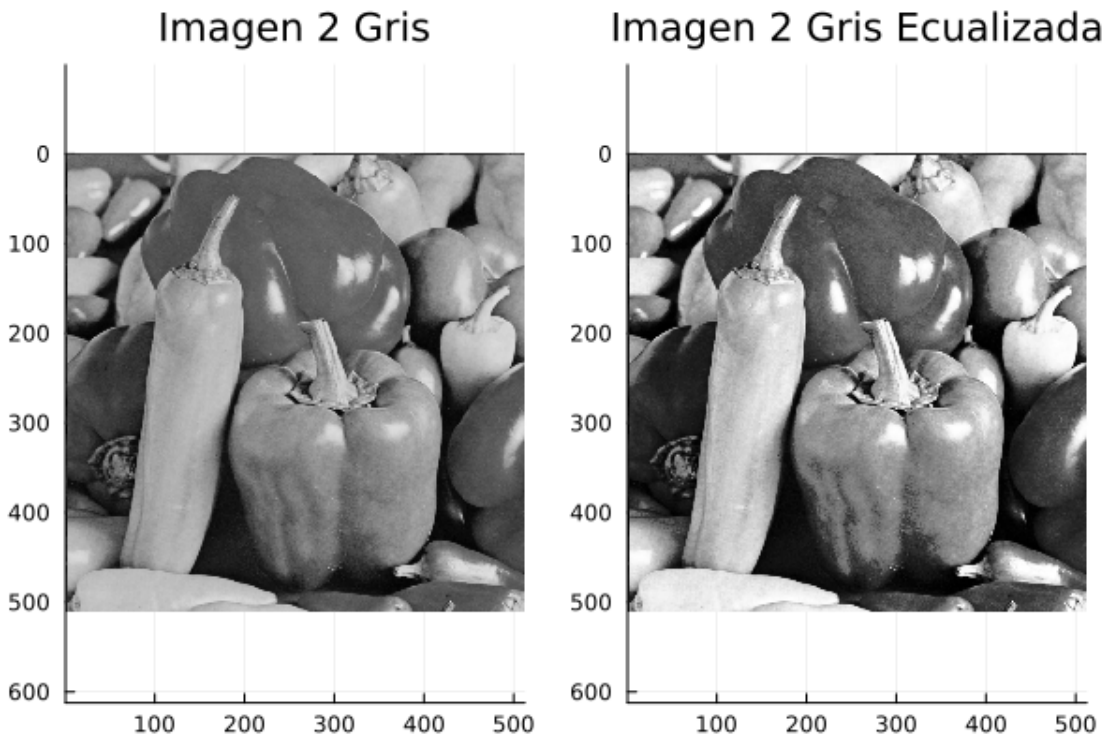


Para este caso, la ecualización ha sido exitosa.

Si trabajamos con la imagen gris de la foto *peppers* si que notaremos diferencia. Veremos que la imagen ecualizada es diferente a la imagen original

[ ]:
```
img2_g_n = adjust_histogram(img2_g, Equalization(nbins = 256, minval = 0,␣
 ↪maxval = 1))
println("Son ambas imágenes iguales: ",img2_g == img2_g_n)
plot(
    heatmap(img2_g, title="Imagen 2 Gris"),
    heatmap(img2_g_n, title="Imagen 2 Gris Ecualizada"),
    layout=(1, 2),
    yflip = true
)
```

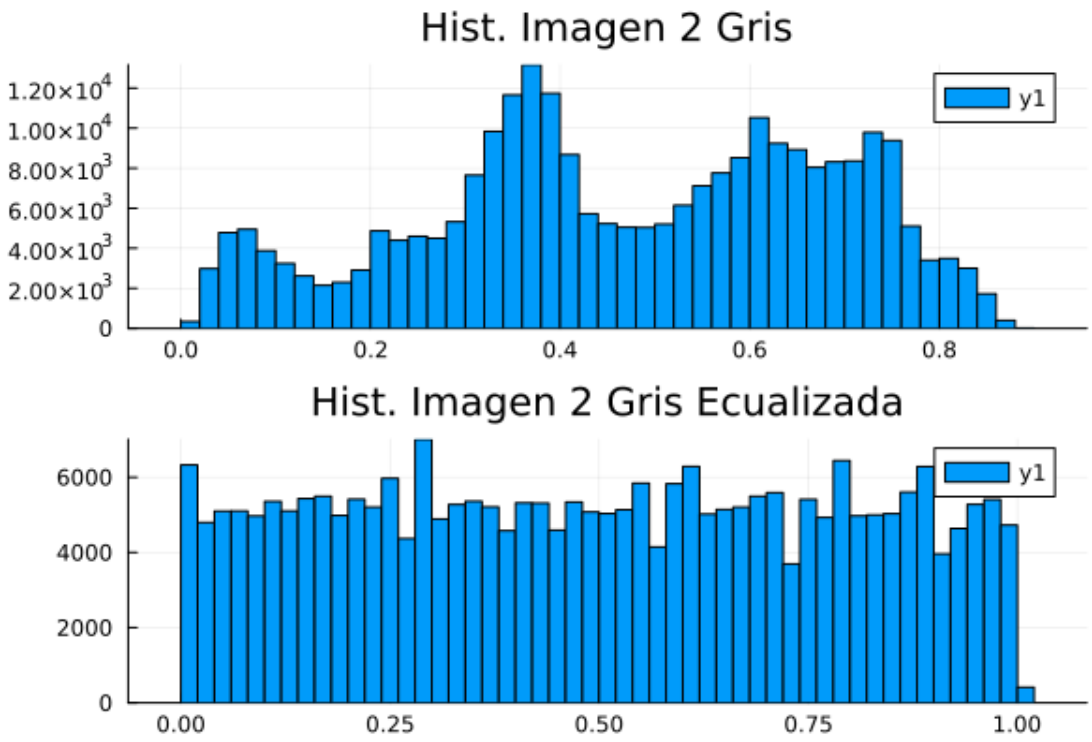
```
Son ambas imágenes iguales: false
```



Vemos como las imágenes no son parecidas, vemos como hay ciertos colores que son más oscuros, sobre todo los tonos rojos de la imagen original.

Veamos además los histogramas, para ver como sí son diferentes.

[ ]:
```
plot(
    histogram(vec(Float64.(img2_g)),bins=50,title="Hist. Imagen 2 Gris"),
    histogram(vec(Float64.(img2_g_n)),bins=50,title="Hist. Imagen 2 Gris␣
 ↪Ecualizada"),
    layout=(2,1)
)
```

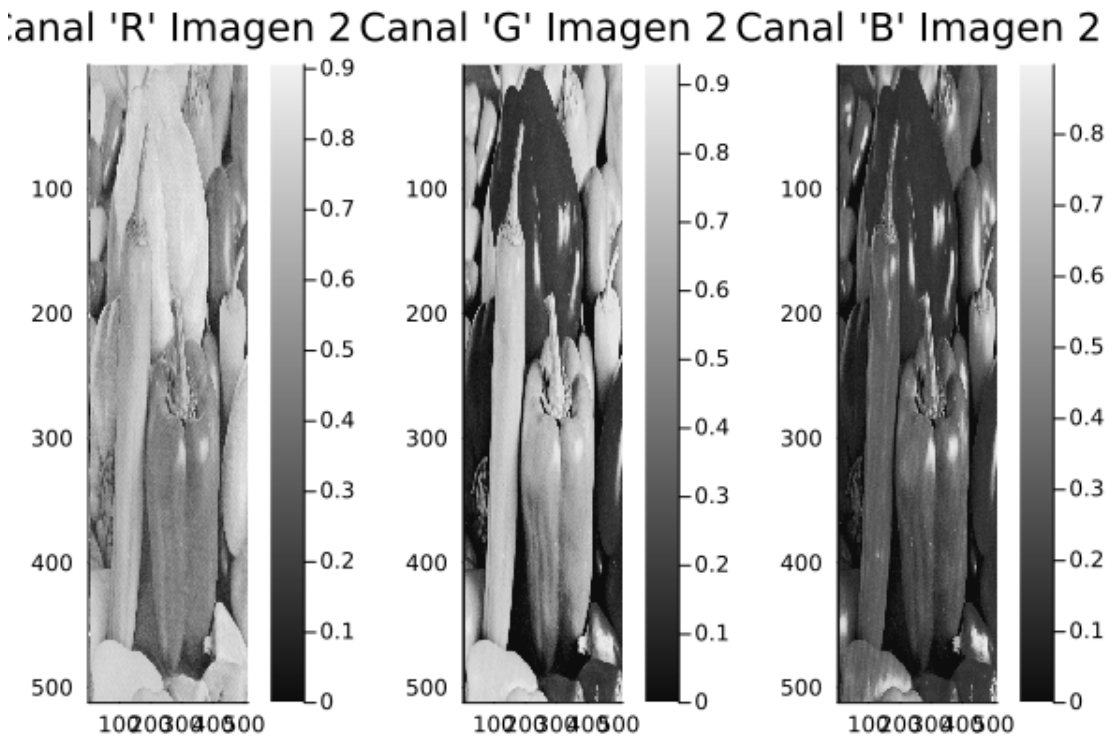


Vemos como este histograma si que ha sido ecualizado. La forma de ambos histogramas no se parece. POdemos decir que la ecualización ha sido satisfactoria.

Ahora, veremos la segunda imagen a color, y veremos también la imagen respectiva de cada canal de la imagen.

[ ]:
```
img2 # Imagen 2: 'Peppers'
```

[ ]:
```
plot(
    heatmap(R_img2, title="Canal 'R' Imagen 2",c=:grays),
    heatmap(G_img2,title="Canal 'G' Imagen 2",c=:grays),
    heatmap(B_img2,title="Canal 'B' Imagen 2",c=:grays),
    layout=(1,3),
    yflip = true
)
```



Ahora ya tenemos representada la imagen y los tres canales de la imagen RGB.

Ahora reduciremos la escala de la imagen 1, primero tenemos que saber, que si reducimos por un factor de 10, la nueva imagen será 10 veces más pequeña, por tanto, a la función que reescala le pasaremos un factor de 1/10.

[ ]:
```
img1_red1 = imresize(img1, ratio = 1/10, method=BSpline(Constant()))
```

Ya tenemos la primera reducción hecha, ahora vayamos a las siguientes:

[ ]:
```
img1_red2 = imresize(img1, ratio = 1/10, method=BSpline(Linear()))
```

[ ]:
```
img1_red3 = imresize(img1,ratio=0.1,algorithm=:bicubic)
```

Las desplegamos conjuntamente para verlas juntas.

[ ]:
```
mosaicview(img1_red1, img1_red2, img1_red3, nrow=1)
```

Ahora las volvemos a aumentar, por el factor de 10. Las mostraremos todas juntas.

[ ]:
```
img1_aug1 = imresize(img1_red1, ratio = 10, method=BSpline(Constant()))
img1_aug2 = imresize(img1_red2, ratio = 10, method=BSpline(Linear()))
img1_aug3 = imresize(img1_red3, ratio = 10 ,algorithm=:bicubic)
mosaicview(img1_aug1, img1_aug2, img1_aug3, nrow=1)
```

Como vemos, al cambiar el tamaño de las imágenes, hay datos que se pierden, y al interpolar para agrandar la imagen, los datos no son iguales que la imagen original.

Se debe tener cuidado al reducir y ampliar imágenes, porque la calidad de estas se ve afectada.