

Ejercicio4_Reducción_de_dimensionalidad

January 10, 2025

1 Ejercicio 4: Reducción de dimensionalidad

Felipe Andres Casillo

```
[1]: using Pkg  
#Pkg.add("Colors")  
#Pkg.add("LaTeXStrings")  
using Images  
using LinearAlgebra  
using Colors  
using CairoMakie  
using Base.Filesystem  
using LaTeXStrings
```

1.1 Introducción

El **Análisis de Componentes Principales (PCA)** es una técnica ampliamente utilizada en machine learning. Se utiliza para reducir la dimensionalidad (el número de variables o columnas) de los conjuntos de datos manteniendo al mismo tiempo la mayor cantidad de información posible. PCA transforma las variables originales en otras nuevas, llamadas componentes principales, mediante combinaciones lineales de estas.

La idea principal detrás del Análisis de Componentes Principales es identificar las direcciones de espacios en las que la varianza del conjunto de datos es máxima. Así, proyectando los datos sobre estas direcciones, se puede reducir la dimensionalidad y conservar la mayor varianza posible de los valores originales. Matemáticamente estas direcciones son los vectores propios (también conocidos como autovectores) de la matriz de covarianza de los datos.

La **Descomposición en Valores Singulares (SVD)**, por sus siglas en inglés, Singular Value Decomposition) es una potente técnica matemática que se utiliza para descomponer una matriz en sus partes constituyentes. Es una generalización de la descomposición en autovalores de una matriz, y se puede aplicar a cualquier matriz rectangular, no solo a las matrices cuadradas.

La descomposición de una matriz A en su SVD viene dada por la ecuación $A = U\Sigma V^T$, donde U y V son matrices ortogonales y Σ es una matriz diagonal con los valores singulares de A en la diagonal.

Una de las más importantes y llamativas aplicaciones de la descomposición SVD es su utilización en la **compresión de imágenes digitales** de modo que puedan ser transmitidas de manera eficiente por medios electrónicos. El problema que se quiere considerar es el de saber cuál es la cantidad

mínima de información que se necesita trasmitir para lograr imágenes nítidas, sin que se pierdan las partes esenciales, y por otra parte se ahorre almacenamiento.

Una forma de evaluar la compresión de una imagen es con la suma de los cuadrados de los valores singulares (σ^2); en concreto, para saber cuánto se preserva al truncar en k valores singulares, se tiene la siguiente expresión

$$E(k) = \frac{\sum_{i=1}^k \sigma_i^2}{\sum_{i=1}^n \sigma_i^2}$$

Una buena compresión se logra cuando un k pequeño retiene un alto porcentaje de la información.

A continuación, se aplicará el método SVD a diferentes imágenes a color y en escala de grises.

1.2 Imágenes a escala de grises

```
[2]: #Esta función realiza la comprensión de una imagen por medio de la
      ↪Descomposición en Valores Singulares;
#dicha compresión está dada por el parámetro k el cual indica el número de
      ↪componentes principales a conservar.
#Devuelve la imagen comprimida y el tamaño en KB
function imageSVDgray(image, k::Int)
    #Cargar la imagen
    img = load(image)
    #Obtener la matriz de la imagen en escala de grises
    img_gray = Gray.(img)
    #SVD
    G = svd(img_gray)
    new_G = G.U[:, 1:k]*Diagonal(G.S[1:k])*G.Vt[1:k, :]
    #Asegura que cada uno de los valores se encuentre en el rango [0,1]
    new_G = clamp.(new_G, 0.0, 1.0)
    #Guardar la nueva imagen
    name, ext = splitext(image)
    save("$(name)_Gk$k$ext", new_G)
    size = stat("$(name)_Gk$k$ext").size
    return new_G, size/1000
end
```

```
[2]: imageSVDgray (generic function with 1 method)
```

```
[3]: #Esta función gráfica una muestra de cuatro imágenes
function plotimages(images, labels)
    fig = Figure(size = (1000,1000))
    n = 1
    for i in 1:2, j in 1:2
        ax = CairoMakie.Axis(fig[i, j], aspect = DataAspect(), title =
            ↪labels[n])
        if n != 1 hidedecorations!(ax) end
        image!(ax, rotr90(images[n]))
        n+=1
    end
```

```

    end
    return fig
end

```

[3]: plotimages (generic function with 1 method)

[4]: #Esta función gráfica la suma de los valores singulares al cuadrado.

```

function norm_sum_plot(image::Matrix{Gray{N0f8}})
    A = svd(image)
    f(k) = sum(x->x^2, A.S[1:k])/sum(x->x^2, A.S)
    Sum = [f(x) for x in 1:length(A.S)]
    i = findfirst(item -> item >= 0.95, Sum)
    fig = Figure(size=(400,400))
    ax = CairoMakie.Axis(fig[1,1], xlabel = L"k-singular\text{ }value", title =_
    "Suma normalizada de los Valores Singulares")
    lines!(ax, 1:length(A.S), Sum)
    hlines!(ax, [0.95], color = "red", alpha = 0.6)
    vlines!(ax, [i], color = "red", label = "k = $(i), Sum > 95%", alpha = 0.6)
    axislegend(position=:rb)
    return fig
end

function norm_sum_plot(image::Matrix{RGB{N0f8}})
    #Canales RGB
    channels = channelview(image)
    fig = Figure(size=(1200,400))
    for c in 1:3
        chan = convert(Matrix{Float64}, channels[c,:,:])
        C = svd(chan)
        f(k) = sum(x->x^2, C.S[1:k])/sum(x->x^2, C.S)
        Sum = [f(x) for x in 1:length(C.S)]
        i = findfirst(item -> item >= 0.95, Sum)
        #Gráfica
        ax = CairoMakie.Axis(fig[1,c], xlabel = L"k-singular\text{ }value")
        lines!(ax, 1:length(C.S), Sum)
        hlines!(ax, [0.95], color = "red", alpha = 0.6)
        vlines!(ax, [i], color = "red", label = "k = $(i), Sum > 95%", alpha =_
        0.6)
        axislegend(position=:rb)
    end
    Label(fig[1,1, Top()], "Red", fontsize=16, color="red")
    Label(fig[1,2, Top()], "Green", fontsize=16, color="green")
    Label(fig[1,3, Top()], "Blue", fontsize=16, color="blue")
    axislegend(position=:rb)
    return fig
end

```

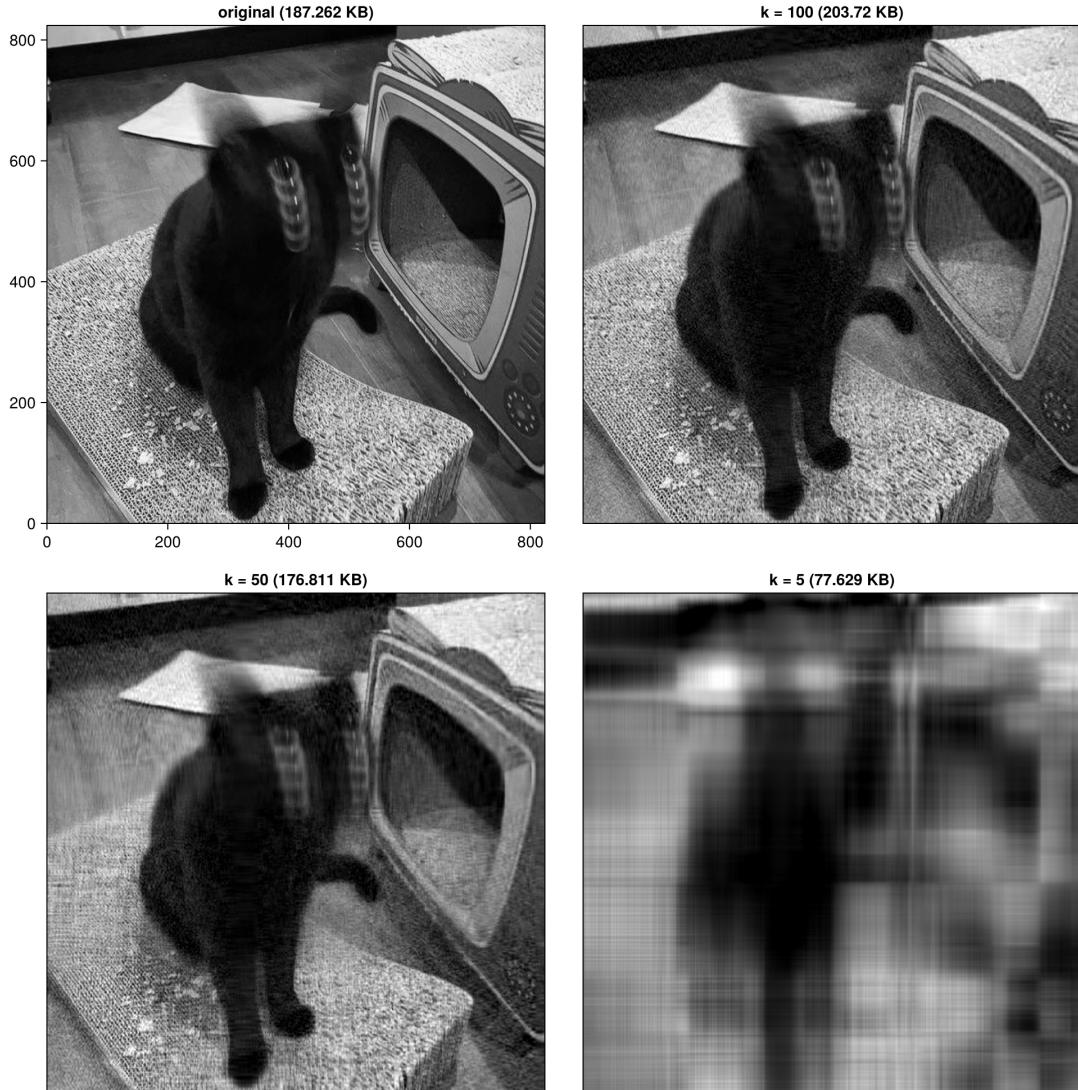
[4]: norm_sum_plot (generic function with 2 methods)

1.2.1 Ejemplo 1

```
[5]: gato_G = Gray.(load("../fig/gato.jpg"))
save("../fig/gato_G.jpg", gato_G)
s1 = stat("../fig/gato_G.jpg").size
gato_Gk100, s2 = imageSVDgray("../fig/gato.jpg", 100)
gato_Gk50, s3 = imageSVDgray("../fig/gato.jpg", 50)
gato_Gk5, s4 = imageSVDgray("../fig/gato.jpg", 5);
```

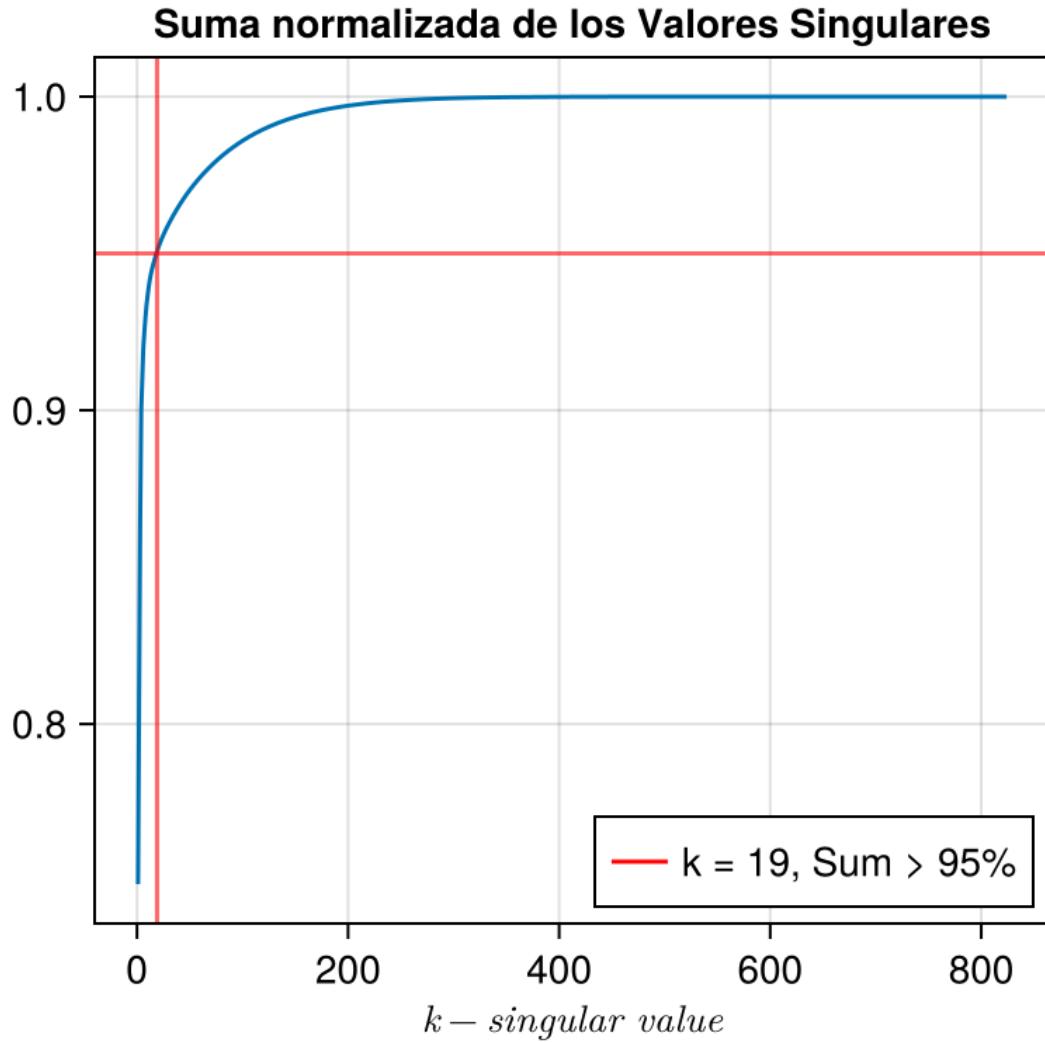
```
[6]: images = [gato_G, gato_Gk100, gato_Gk50, gato_Gk5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB", "k = 5 ($s4) KB"]
plotimages(images, labels)
```

[6]:



```
[7]: norm_sum_plot(gato_G)
```

```
[7]:
```

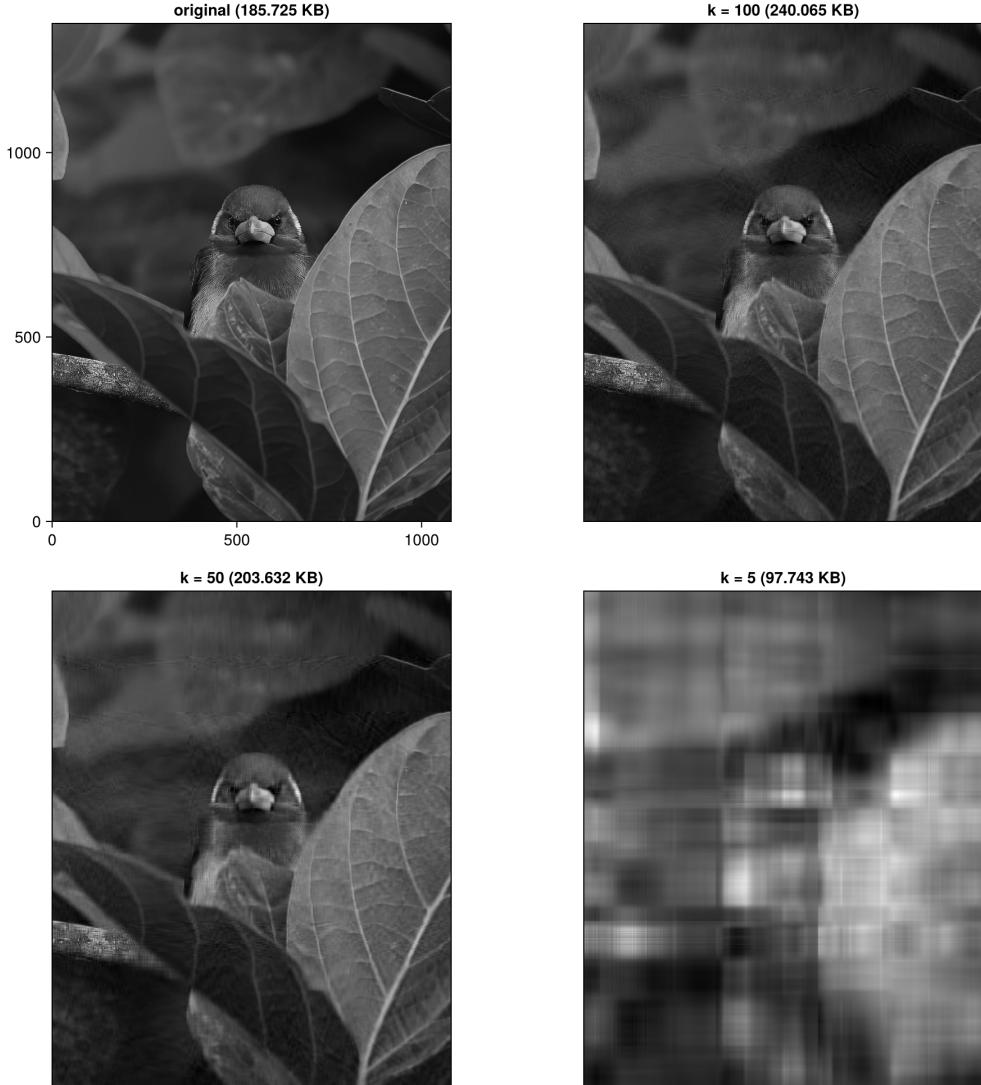


1.2.2 Ejemplo 2

```
[8]: pajaro_G = Gray.(load("../fig/pajaro.jpg"))
save("../fig/pajaro_G.jpg", pajaro_G)
s1 = stat("../fig/pajaro_G.jpg").size
pajaro_Gk100, s2 = imageSVDgray("../fig/pajaro.jpg", 100)
pajaro_Gk50, s3 = imageSVDgray("../fig/pajaro.jpg", 50)
pajaro_Gk5, s4 = imageSVDgray("../fig/pajaro.jpg", 5);
```

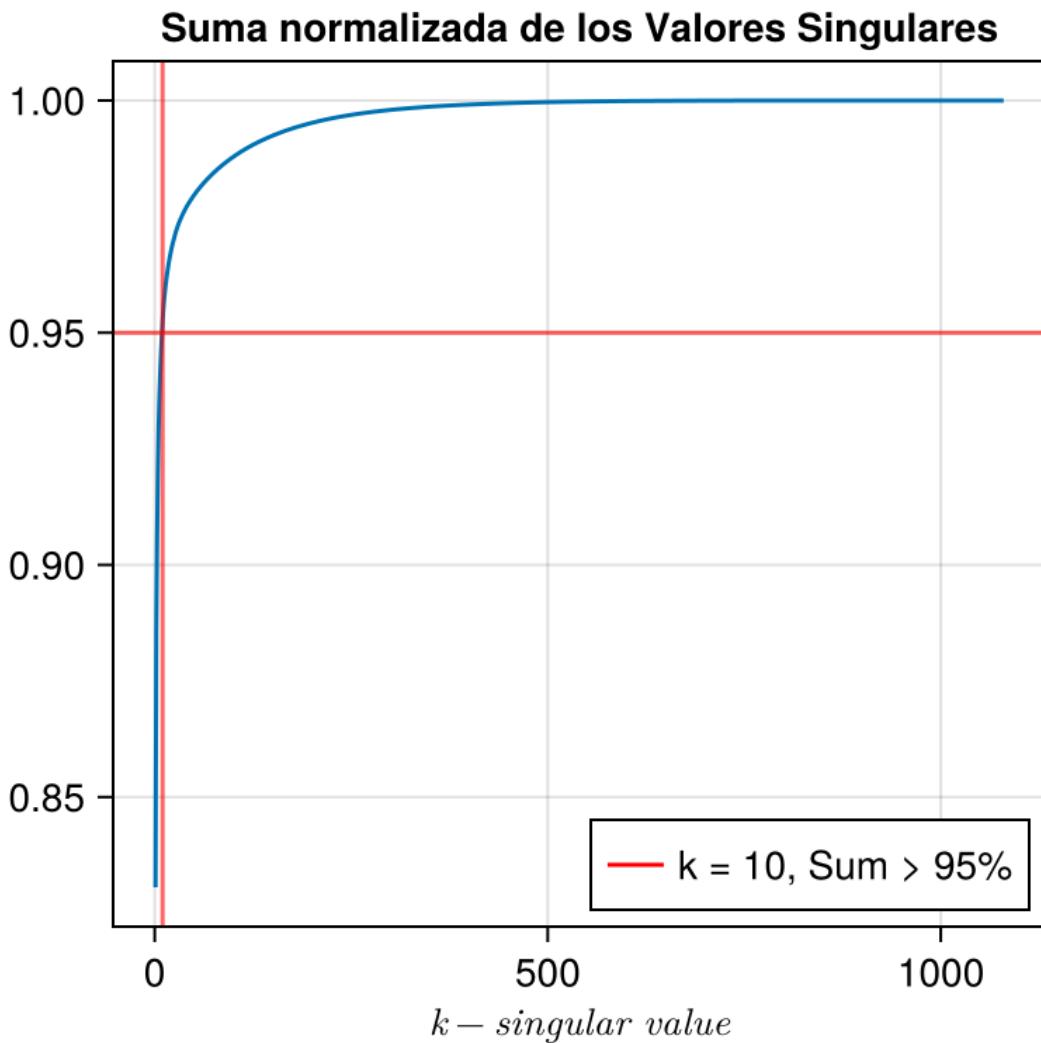
```
[9]: images = [pajaro_G, pajaro_Gk100, pajaro_Gk50, pajaro_Gk5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB", "k = 5 ($s4) KB"]
plotimages(images, labels)
```

[9]:



```
[10]: norm_sum_plot(pajaro_G)
```

[10]:



1.2.3 Ejemplo 3

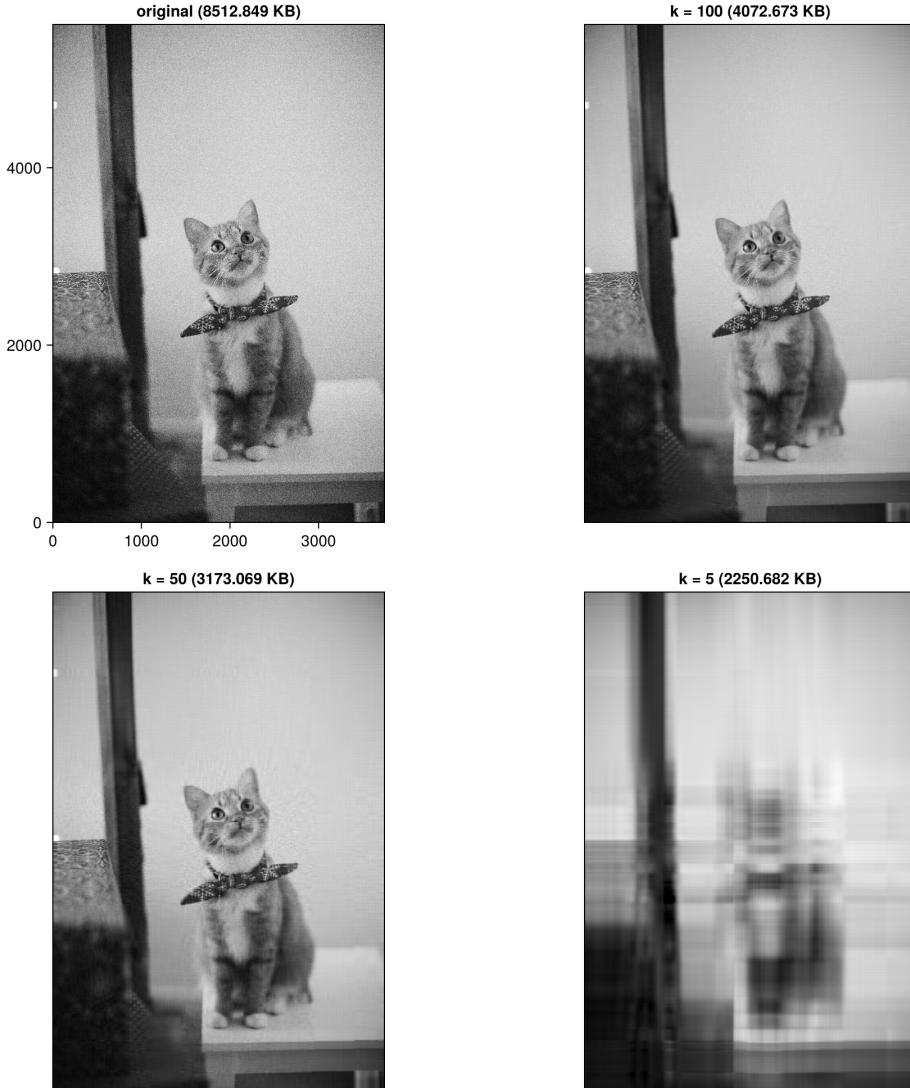
```
[11]: michi_G = Gray.(load("./fig/michi.jpg"))
save("./fig/michi_G.jpg", michi_G)
s1 = stat("./fig/michi_G.jpg").size
michi_Gk100, s2 = imageSVDgray("./fig/michi.jpg", 100)
michi_Gk50, s3 = imageSVDgray("./fig/michi.jpg", 50)
michi_Gk5, s4 = imageSVDgray("./fig/michi.jpg", 5);
```



```
[12]: images = [michi_G, michi_Gk100, michi_Gk50, michi_Gk5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB", "k = 5 ($s4) KB"]
```

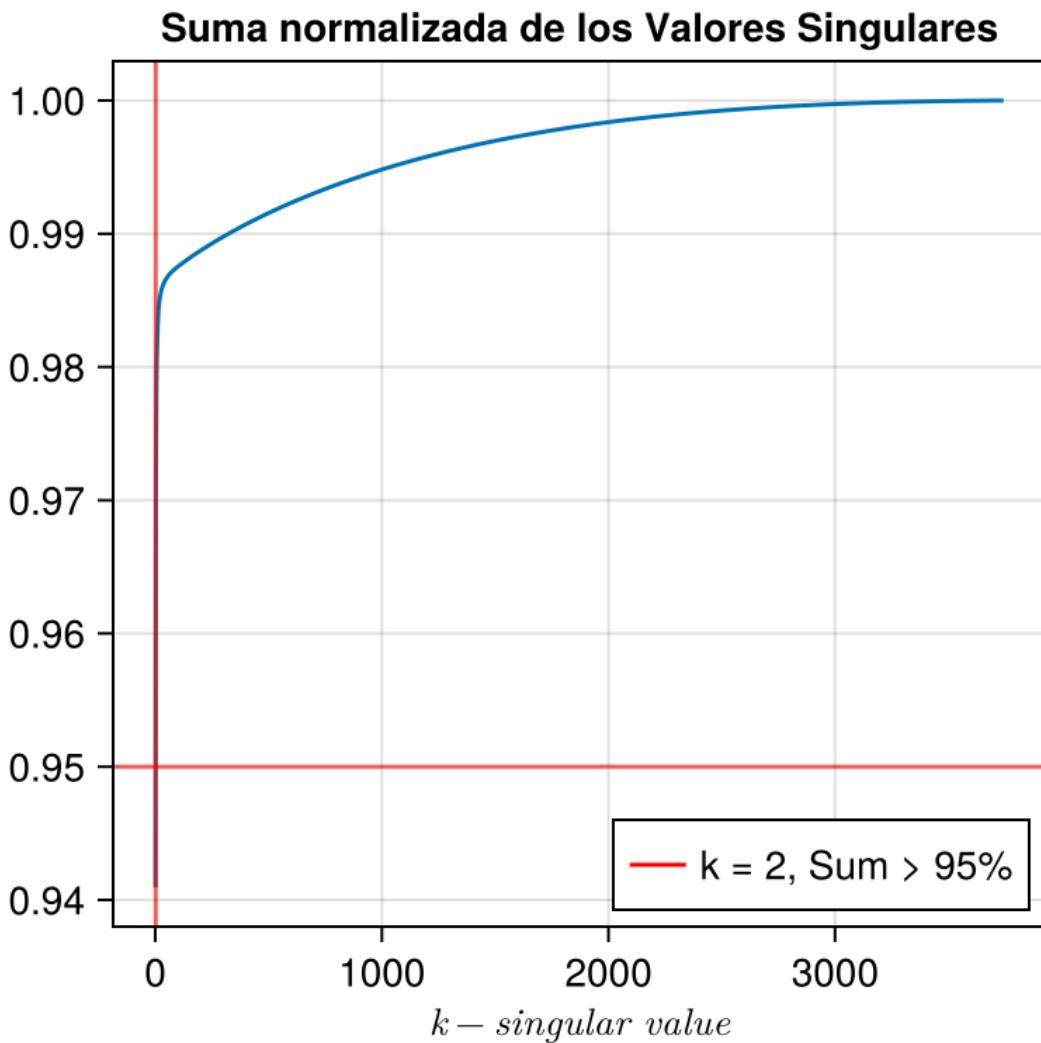
```
plotimages(images, labels)
```

[12] :



```
[13] : norm_sum_plot(michi_G)
```

[13] :



1.2.4 Ejemplo 4

```
[14]: #Estas imágenes están en formato PNG
salchi_G = Gray.(load("./../fig/salchi.png"))
save("./../fig/salchi_G.png", salchi_G)
s1 = stat("./../fig/salchi_G.png").size
salchi_Gk100, s2 = imageSVDgray("./../fig/salchi.png", 100)
salchi_Gk50, s3 = imageSVDgray("./../fig/salchi.png", 50)
salchi_Gk5, s4 = imageSVDgray("./../fig/salchi.png", 5);
```

```
[15]: images = [salchi_G, salchi_Gk100, salchi_Gk50, salchi_Gk5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB", "k = 5 ($s4) KB"]
```

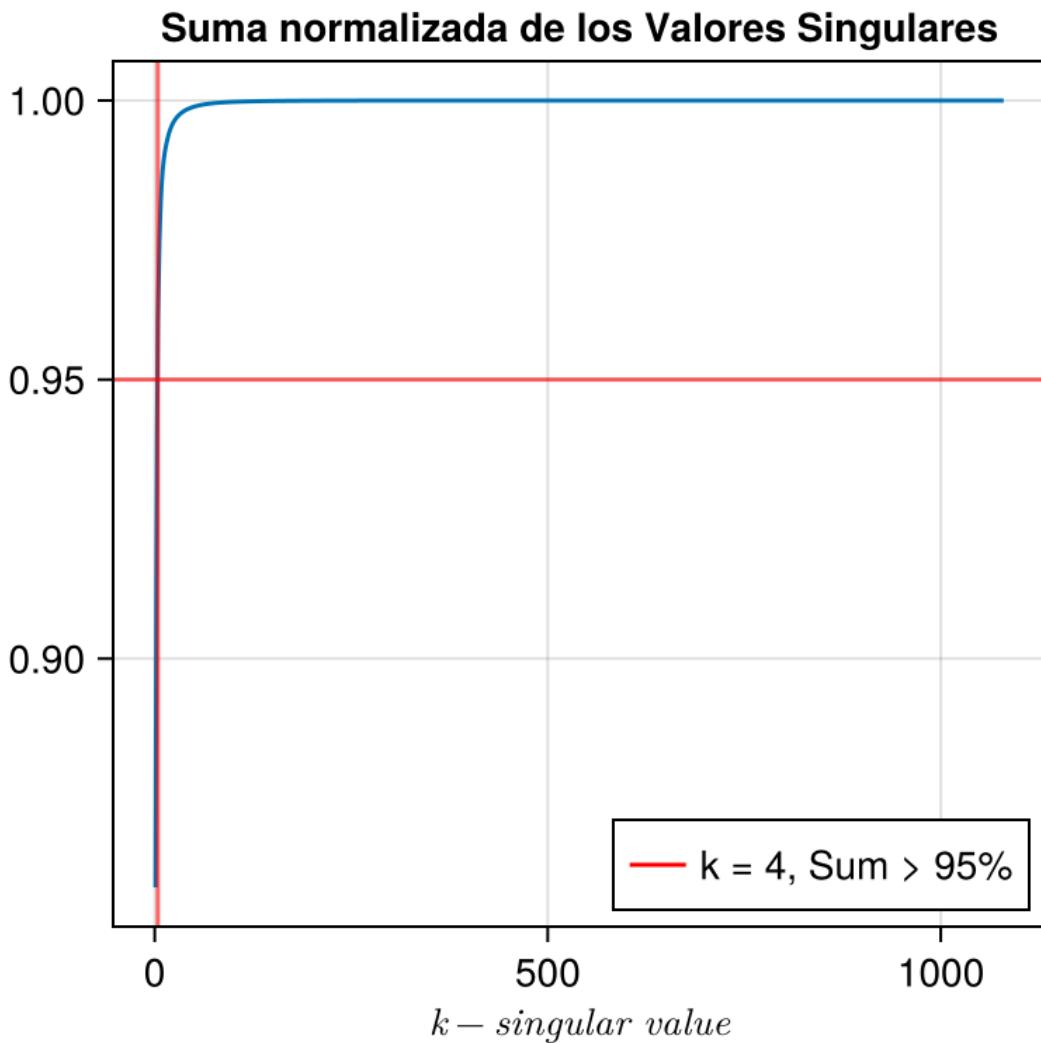
```
plotimages(images, labels)
```

[15] :



[16] : norm_sum_plot(salchi_G)

[16] :



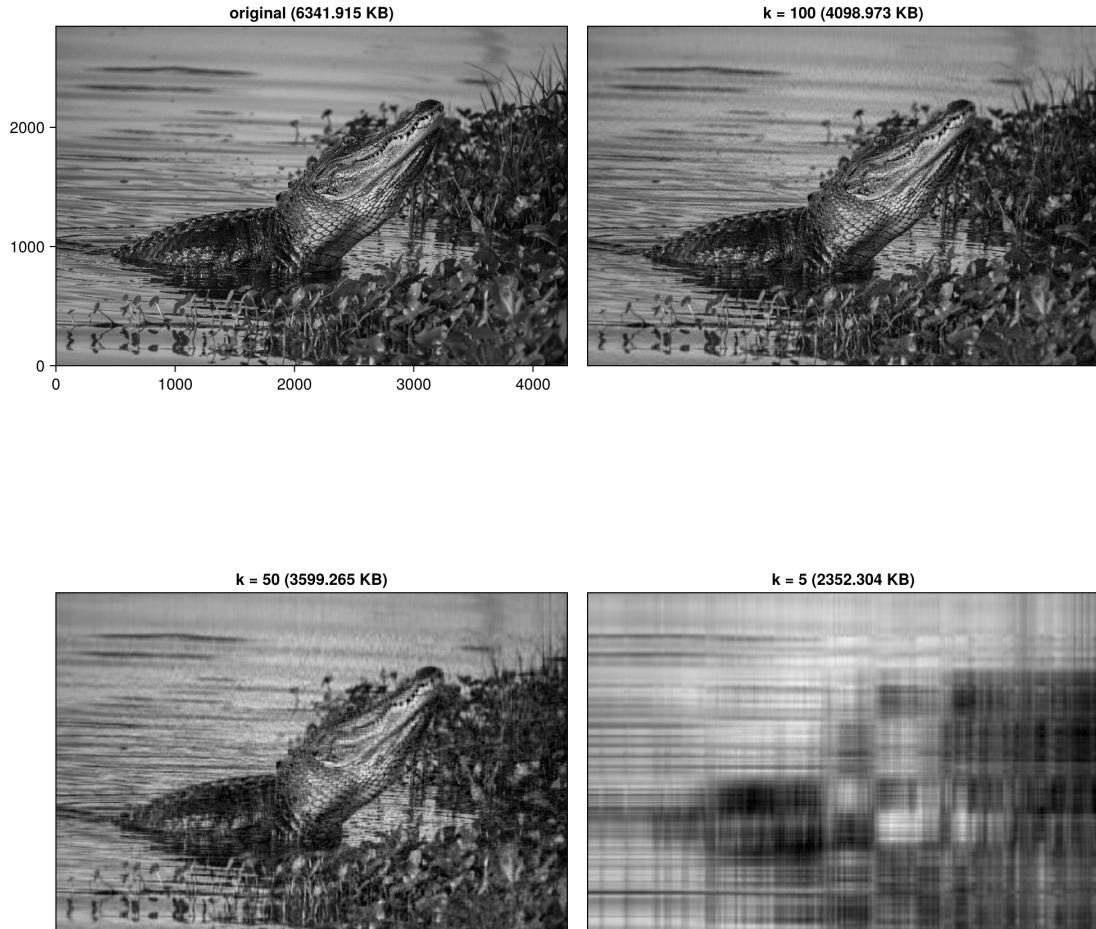
1.2.5 Ejemplo 5

```
[17]: wani_G = Gray.(load("./fig/wani.png"))
save("./fig/wani_G.png", wani_G)
s1 = stat("./fig/wani_G.png").size
wani_Gk100, s2 = imageSVDgray("./fig/wani.png", 100)
wani_Gk50, s3 = imageSVDgray("./fig/wani.png", 50)
wani_Gk5, s4 = imageSVDgray("./fig/wani.png", 5);
```

```
[18]: images = [wani_G, wani_Gk100, wani_Gk50, wani_Gk5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB", "k = 5 ($s4) KB"]
```

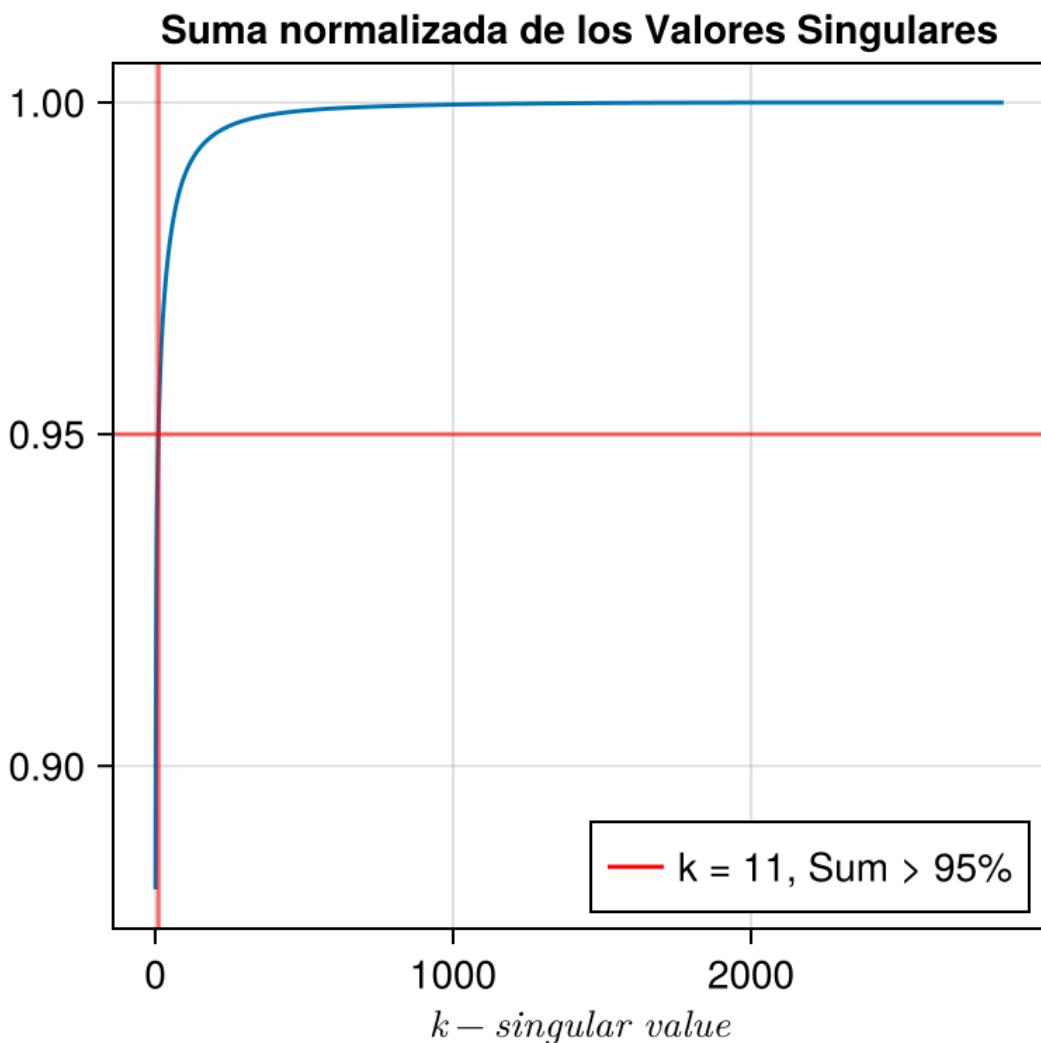
```
plotimages(images, labels)
```

[18] :



```
[19] : norm_sum_plot(wani_G)
```

[19] :



1.3 Imágenes a color

[20]: #Esta función realiza la compresión de una imagen por medio de la ↵Descomposición en Valores Singulares;
#dicha compresión está dada por el parámetro k el cual indica el número de ↵componentes principales a conservar.

```
function imageSVDcolor(image, k::Int)
    #Cargar la imagen
    img = load(image)
    #Obtener las matrices de cada canal (RGB)
    channels = channelview(img)
    new_chan = []
```

```

#SVD
for c in 1:3
    chan = convert(Matrix{Float64}, channels[c,:,:])
    C = svd(chan)
    new_C = C.U[:,1:k]*Diagonal(C.S[1:k])*C.Vt[1:k,:]
    #Asegura que cada uno de los valores se encuentre en el rango [0,1]
    new_C = clamp.(new_C, 0.0, 1.0)
    push!(new_chan, NOf8.(new_C))
end
#Crear la nueva imagen RGB
new_img = colorview(RGB, new_chan[1], new_chan[2], new_chan[3])
#Guardar la nueva imagen
name, ext = splitext(image)
save("${name}_k${ext}", new_img)
size = stat("${name}_k${ext}").size
return new_img, size/1000
end

```

[20]: imageSVDcolor (generic function with 1 method)

1.3.1 Ejemplo 1

```

[21]: gato = load("../fig/gato.jpg")
s1 = stat("../fig/gato.jpg").size
gato_k100, s2 = imageSVDcolor("../fig/gato.jpg", 100)
gato_k50, s3 = imageSVDcolor("../fig/gato.jpg", 50)
gato_k5, s4 = imageSVDcolor("../fig/gato.jpg", 5);

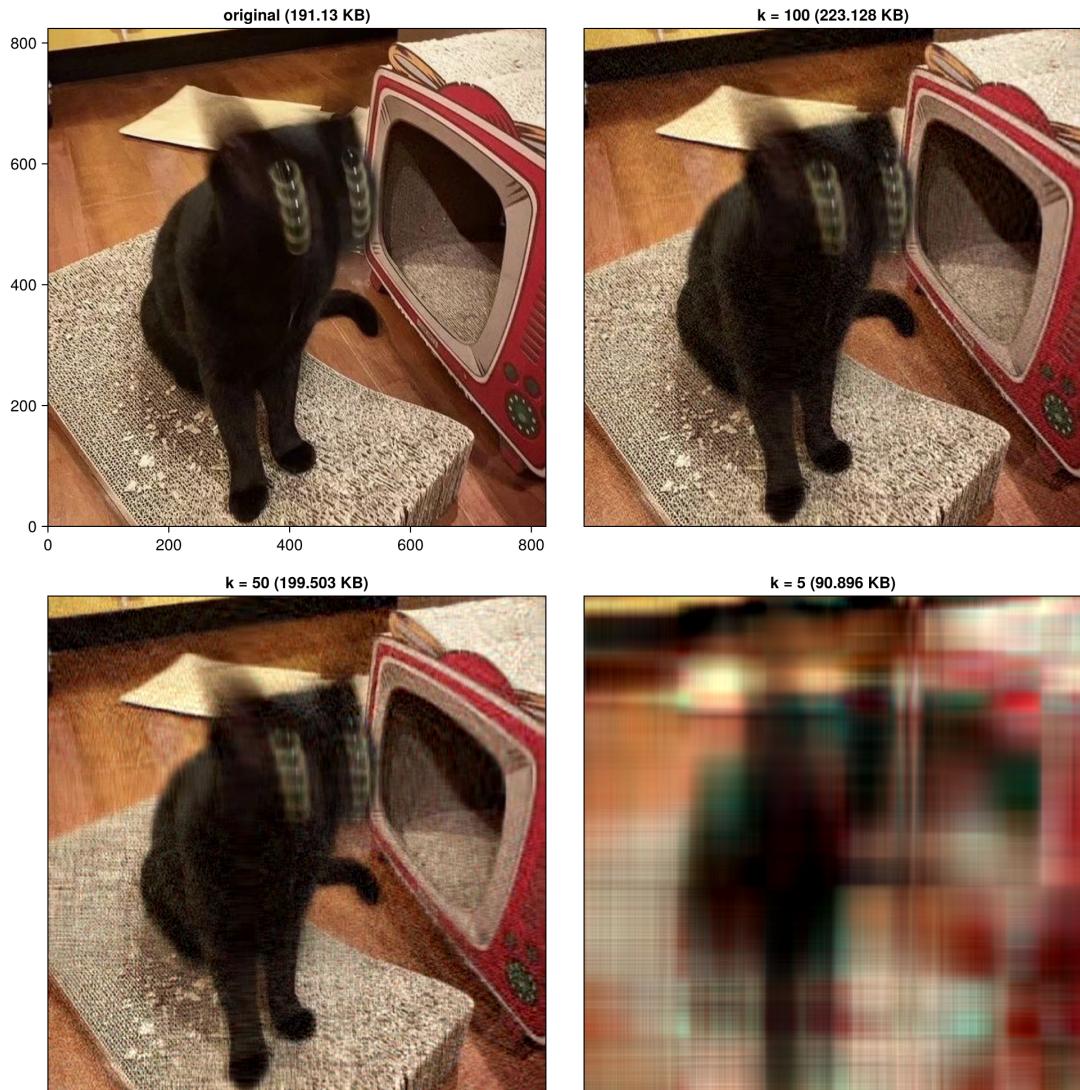
```

```

[22]: images = [gato, gato_k100, gato_k50, gato_k5]
labels = ["original ($(s1/1000) KB)", "k = 100 ($(s2) KB)", "k = 50 ($(s3) KB)", "k = 5 ($(s4) KB)"]
plotimages(images, labels)

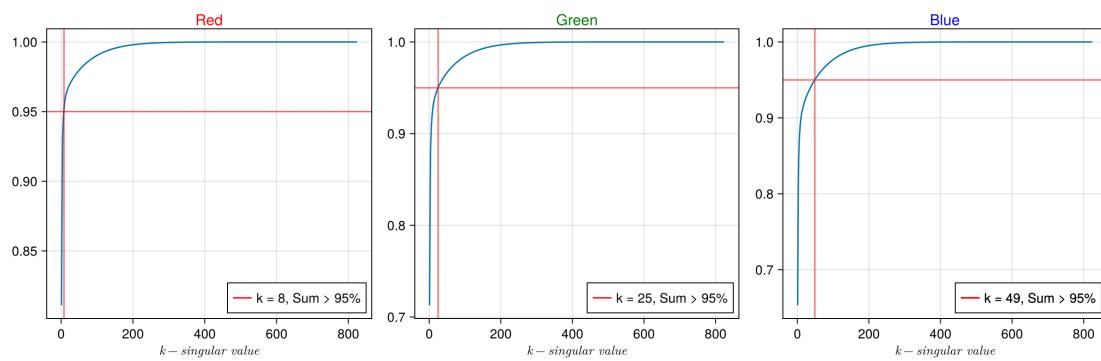
```

[22]:



[23] : norm_sum_plot(gato)

[23] :



1.3.2 Ejemplo 2

```
[24]: pajaro = load("./../fig/pajaro.jpg")
s1 = stat("./../fig/pajaro.jpg").size
pajaro_k100, s2 = imageSVDcolor("./../fig/pajaro.jpg", 100)
pajaro_k50, s3 = imageSVDcolor("./../fig/pajaro.jpg", 50)
pajaro_k5, s4 = imageSVDcolor("./../fig/pajaro.jpg", 5);

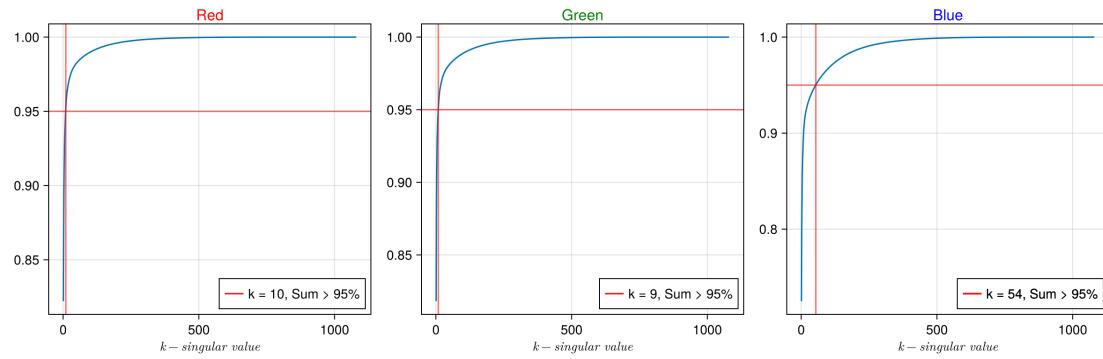
[25]: images = [pajaro, pajaro_k100, pajaro_k50, pajaro_k5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB", "k = 5 ($s4) KB"]
plotimages(images, labels)
```

[25] :



```
[26]: norm_sum_plot(pajaro)
```

```
[26]:
```



1.3.3 Ejemplo 3

```
[27]: michi = load("./../fig/michi.jpg")
s1 = stat("./../fig/michi.jpg").size
michi_k100, s2 = imageSVDcolor("./../fig/michi.jpg", 100)
michi_k50, s3 = imageSVDcolor("./../fig/michi.jpg", 50)
michi_k5, s4 = imageSVDcolor("./../fig/michi.jpg", 5);
```

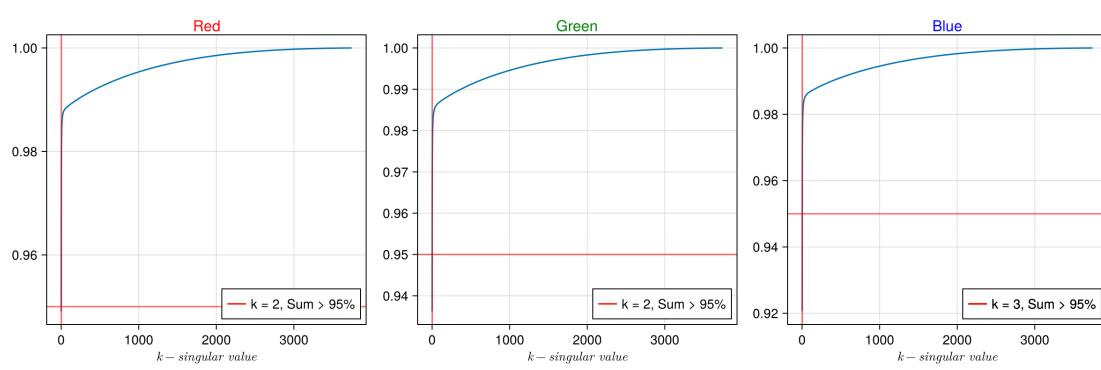
```
[28]: images = [michi, michi_k100, michi_k50, michi_k5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB",
          "k = 5 ($s4) KB"]
plotimages(images, labels)
```

```
[28]:
```



```
[29]: norm_sum_plot(michi)
```

[29] :

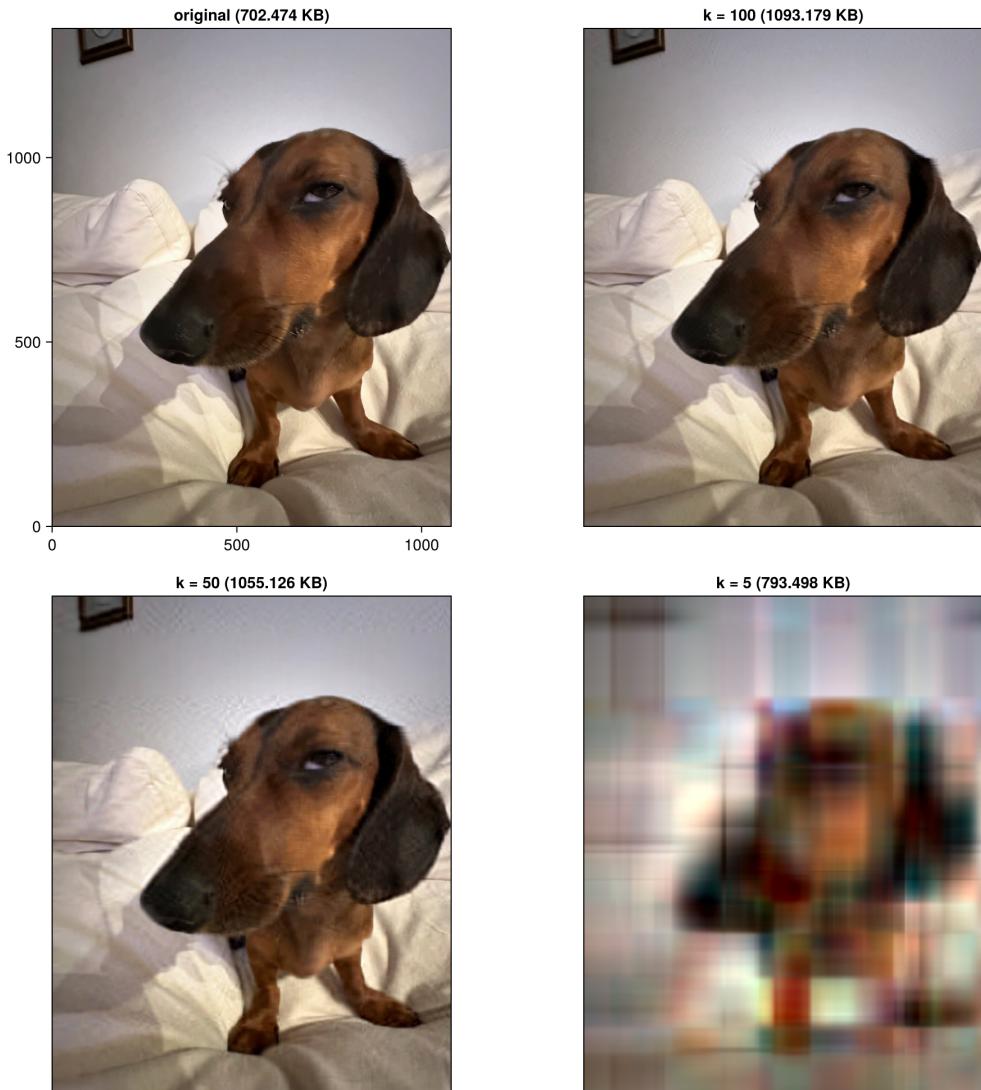


1.3.4 Ejemplo 4

```
[30]: #Estas imágenes están en formato PNG
salchi = load("./../fig/salchi.png")
s1 = stat("./../fig/salchi.png").size
salchi_k100, s2 = imageSVDcolor("./../fig/salchi.png", 100)
salchi_k50, s3 = imageSVDcolor("./../fig/salchi.png", 50)
salchi_k5, s4 = imageSVDcolor("./../fig/salchi.png", 5);

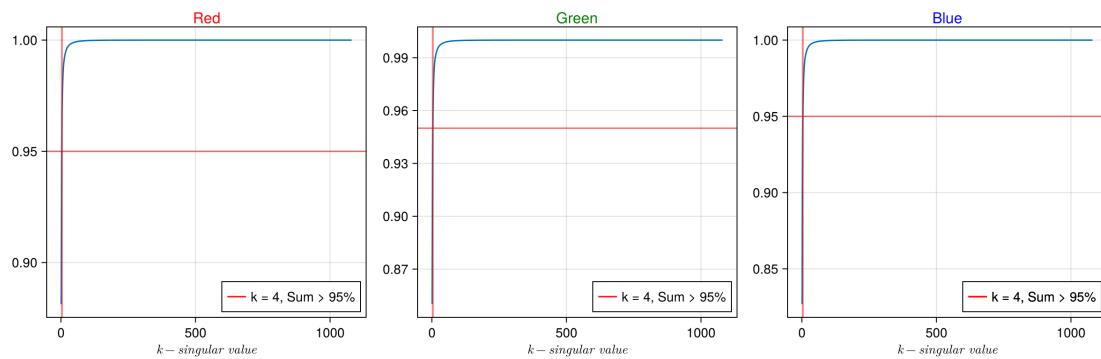
[31]: images = [salchi, salchi_k100, salchi_k50, salchi_k5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB", "k = 5 ($s4) KB"]
plotimages(images, labels)
```

[31]:



```
[32]: norm_sum_plot(salchi)
```

```
[32]:
```

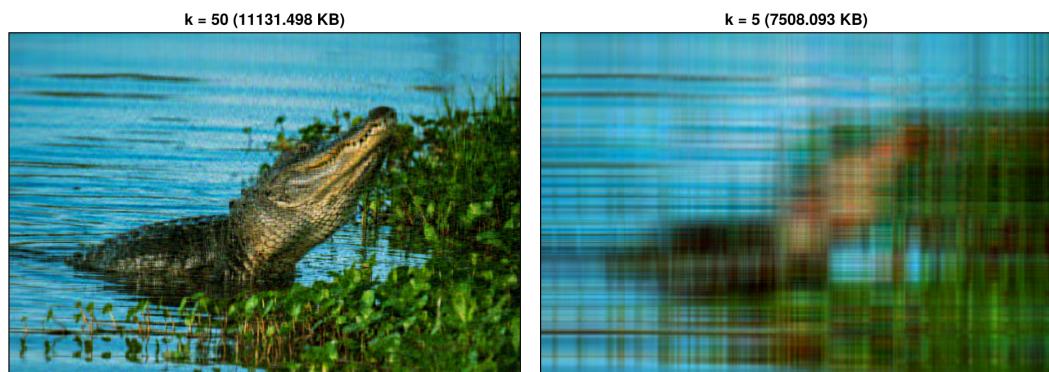
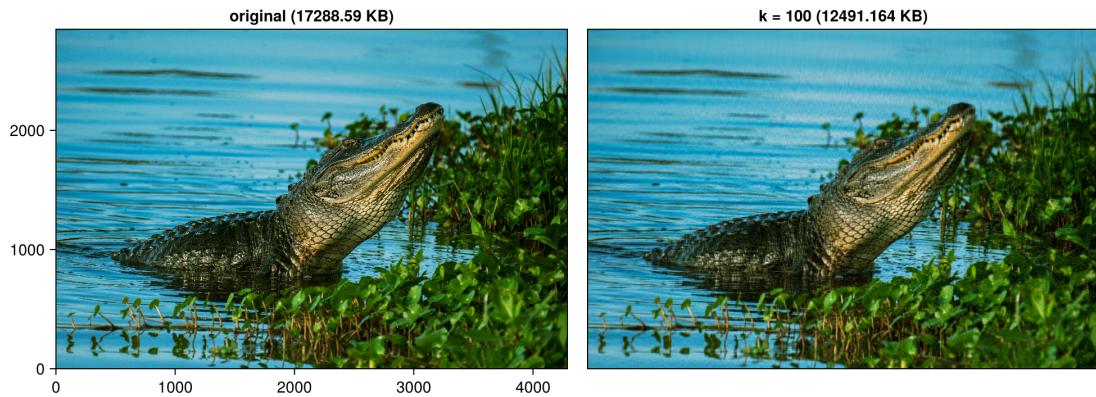


1.3.5 Ejemplo 5

```
[33]: wani = load("./../fig/wani.png")
s1 = stat("./../fig/wani.png").size
wani_k100, s2 = imageSVDcolor("./../fig/wani.png", 100)
wani_k50, s3 = imageSVDcolor("./../fig/wani.png", 50)
wani_k5, s4 = imageSVDcolor("./../fig/wani.png", 5);
```

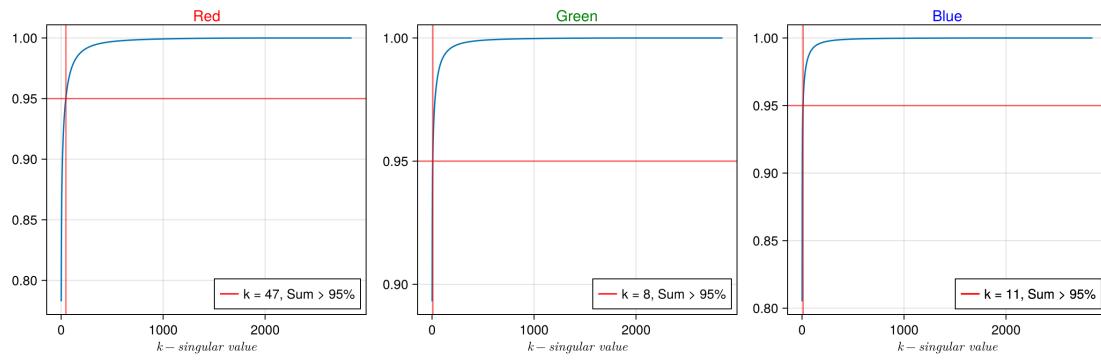
```
[34]: images = [wani, wani_k100, wani_k50, wani_k5]
labels = ["original ($s1/1000) KB", "k = 100 ($s2) KB", "k = 50 ($s3) KB",
          "k = 5 ($s4) KB"]
plotimages(images, labels)
```

```
[34]:
```



```
[35]: norm_sum_plot(wani)
```

```
[35] :
```



Algunas de las conclusiones que se pueden plantear a partir de las imágenes y gráficas son las siguientes:

1. La efectividad de la compresión depende claramente de las características de los valores singulares de la matriz que representa la imagen. Si estos valores decrecen rápidamente (o, de forma equivalente, si la suma acumulativa de los valores singulares converge rápidamente a 1), significa que gran parte de la información de la imagen está contenida en un pequeño subconjunto de valores singulares. En estos casos, es posible truncar los valores singulares más pequeños sin una pérdida significativa en la calidad visual. Esto se observa, por ejemplo, en las imágenes del gato naranja, el perro salchicha y el cocodrilo.

En general, las imágenes con grandes regiones de color uniforme o con menos elementos detallados presentan una rápida convergencia en la suma acumulativa de sus valores singulares. Por otro lado, en imágenes con mayor cantidad de elementos y detalles, como la primera imagen del gato, el índice del valor singular necesario para conservar más del 95% de la información es significativamente mayor que en las otras imágenes.

2. El tamaño ocupado al almacenar las imágenes también varía según las características de compresión. Convertir las imágenes a escala de grises, en varios casos, resultó en tamaños mayores al original. Para las imágenes comprimidas, aunque el tamaño disminuye a medida que se reduce el número de valores singulares, en ciertos casos, como el del gato negro o el pájaro, el tamaño resultante fue similar o incluso mayor al original, lo que sugiere que la compresión no se reflejó significativamente en términos de almacenamiento, pero sí en una menor calidad visual.

En contraste, las imágenes del gato naranja y el cocodrilo lograron una reducción efectiva en el tamaño. Un factor en común es que las dimensiones originales de estas imágenes eran mayores que las de las otras. Sin embargo, no queda del todo claro qué factores adicionales influyen para que se logre una disminución significativa en el tamaño de almacenamiento.

Como conclusión general: el método de descomposición en valores singulares (SVD) es sencillo y efectivo para la compresión de imágenes, pero su eficiencia depende de ciertas características específicas de las imágenes.