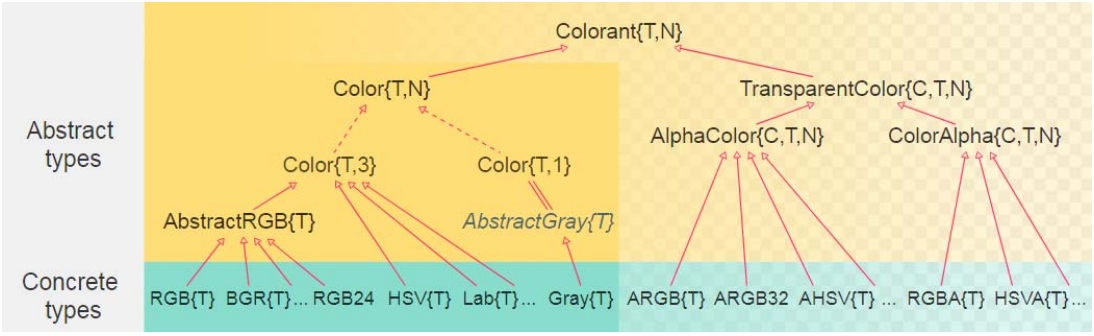
Imágenes en Julia

Aprendizaje Automático

En este tutorial se explican unos conocimientos básicos de procesado de imágenes. En primer lugar, para poder manipular imágenes será necesario tener instalado el paquete *Images*, e importarlo de la forma habitual con *using*.

Una imagen se puede cargar con la función *load*, indicando como argumento el nombre del archivo, para lo cual previamente habrá que haber importado el paquete *FileIO*. Una vez cargada, se puede mostrar utilizando la función *display*.

En Julia, una imagen típica será una variable de tipo *Array{RGB{Normed{UInt8,8}},2}*, es decir, un *array* bidimensional donde cada elemento es de tipo *RGB{Normed{UInt8,8}}*. Esto quiere decir que cada valor del *array* será de tipo RGB, con tres valores (rojo, verde y azul), y teniendo cada uno de ellos un valor normalizados entre 0 y 1. Sin embargo, existen más tipos para especificar otras formas de almacenar colores además de RGB, por ejemplo: BGR, XRGB, RGBA, RGBX, RGB24, HSV, HSL, XYZ, YIQ, Gray, etc. Según la imagen cargada y el espacio de colores en el que se esté trabajando, se usará un tipo u otro de color. En todos los casos, una imagen será una matriz del color correspondiente. Todos estos tipos de color forman una jerarquía como se puede ver en la siguiente imagen (Fuente: https://github.com/JuliaGraphics/ColorTypes.jl):



Como se puede ver, todos los tipos de colores son subtipos de *Colorant*, que agrupa tanto colores con transparencia (como RGBA), como sin transparencia (bien sean colores propiamente dichos como RGB o en escala de grises como Gray). Para asegurarse de que las funciones que se proveen con esta práctica son compatibles con cualquier tipo de color, los argumentos pasados no se especifican como *Array{RGB{Normed{UInt8,8}},2}*, sino como *Array{<:Colorant,2}*, de igual manera que al trabajar con matrices de valores reales en las prácticas anteriores los argumentos de las funciones en lugar de ser *Array{Float32,2}* o

Array{Float64,2} fueron *Array{<:Real,2}*.

En todos los casos, una imagen (bien sea en color o escala de grises, con transparencia o sin ella) es un array de un objeto subtipo de Colorant. El tamaño de este *array* será el tamaño de la imagen. Para acceder al valor de un pixel en las coordenadas nxm, se puede hacer sencillamente mediante

```
imagen[n,m]
```

Y para referenciar una ventana, se puede hacer como si fuera una matriz:

```
imagen[n1:n2,m1:m2]
```

Para cada píxel de una imagen en color, los valores de rojo, verde y azul se pueden referenciar como se puede ver en el siguiente ejemplo, en el que se toman las componentes del píxel situado en las coordenadas (1,1):

```
imagen[1,1].r

imagen[1,1].g

imagen[1,1].b
```

Otra forma de hacerlo, es mediante las funciones *red*, *green* y *blue*, por ejemplo:

```
red(imagen[1,1])

green(imagen[1,1])

blue(imagen[1,1])
```

Por lo tanto, si por ejemplo se quisiera crear una matriz con los valores del canal rojo de una imagen en color, esto se podría hacer mediante una operación de *broadcast* con la función *red*, de la siguiente manera:

```
red.(imagen)
```

- ¿Por qué daría error escribir *red(imagen)*?

Esto devolvería una matriz de valores, pero no una imagen, por lo que no se podría visualizar. Para poder verla, es necesario convertirla en un valor RGB, para lo cual se puede usar la función *RGB*, que recibe los valores de rojo, verde y azul. Por ejemplo, para crear un

valor de RGB (que será un píxel) que contenga el color verde, esto se hará de la siguiente manera:

```
RGB(0, 1, 0)
```

Y por tanto para mostrar únicamente el canal rojo de una imagen, esto se puede hacer de la siguiente manera:

```
matrizRojos = red.(imagen);

imagenRojo = RGB.(matrizRojos,0,0);

display(imagenRojos)
```

Es interesante darse cuenta de que en la segunda línea, cuando se hace un *broadcast* de la función *RGB* sobre todos los valores de la matriz *matrizRojos*, el primer argumento es efectivamente una matriz, mientras que el segundo y el tercero son constantes. En general, las operaciones de *broadcast* se pueden hacer de dos maneras: pasando como argumentos matrices de un determinado tamaño y dimensionalidad (todas del mismo tamaño y dimensionalidad), o poniendo algún o varios argumentos como constantes. En este último caso, estos valores serán tratados como si fueran una matriz correspondiente donde todos los valores son igual a ese.

Otro ejemplo de uso de la función RGB es esta, en la que se crea una imagen igual que la que se tiene:

```
RGB.(red.(imagen), green.(imagen), blue.(imagen))
```

Para convertir una imagen en color a escala de grises, esto se realiza mediante la función *Gray*, que recibe como argumento un objeto de la clase RGB, y por lo tanto para aplicarla a una imagen hay que realizar una operación de *broadcast*, por ejemplo:

```
imagenBN = Gray.(imagen);
```

En este caso, esta variable es de tipo *Array{Gray{Normed{UInt8,8}},2}*, y cada elemento ya no tiene componentes en RGB, sino una única componente, que para acceder a ella se puede realizar de cualquier de las siguientes dos maneras:

```
imagenBN[1,1].val
```

```
gray(imagenBN[1,1])
```

En consecuencia, para convertir una imagen en escala de grises en una matriz con los valores de gris, esto se puede realizar mediante:

```
matrizBN = gray.(imagenBN);
```