TV Digital
y
Procesamiento de Imagenes

Manipulacion de Imagenes por medio de Software

Autor : Rodolfo Echenique

Indice

Marco Teórico

Las imágenes (o gráficos) son una parte importante de cualquier comunicación de medios.

Las imágenes se representan en una computadora (en su mayoría como imágenes de mapas de bits: cada punto o pixel se representa por separado) y pueden manipularse. Las imágenes son un arreglo bidimensional de píxeles.Una imagen es una ilustración almacenada en un archivo JPEG. JP E G es un estándar internacional para indicar cómo almacenar ilustraciones de alta calidad pero con poco espacio. JPEG es un formato de compresión con pérdidas. Esto significa que se comprime, o sea que se hace más pequeño, pero no con el 100 % de la calidad del formato original. Aunque por lo general, lo que se pierde es la calidad o nitidez que no se ve o que no podemos ver de todas formas. Una imagen JPEG es adecuada para casi todos los propósitos. El formato BMP es sin pérdida pero no está comprimido. Un archivo BMP será mucho mayor que un archivo JPEG para la misma imagen. El formato PNG es sin pérdida y está comprimido.

Un arreglo unidimensional es una secuencia de elementos del mismo tipo. Podemos asignar un nombre a un arreglo y después usar números de índice para acceder a los elementos del arreglo.

A un arreglo bidimensional también se le conoce como matriz. Una matriz es una colección de elementos ordenados en filas y columnas. Esto significa que es posible especificar tanto el índice de fila como el de columna, para poder acceder a un valor en la matriz.

Lo que se almacena en cada elemento en la imagen es un pixel. La palabra “pixel” es la abreviación de “elemento de imagen” . En sentido literal es un punto, y la imagen en general está compuesta de muchos de estos puntos. ¿Alguna vez ha colocado una lupa sobre las imágenes en un periódico o revista, en una televisión o incluso en su propio monitor? Cuando vemos la imagen en la revista o en la televisión no parece que estuviera dividida en millones de puntos discretos, pero lo está.

Nuestro aparato sensorial humano no puede distinguir (sin ampliación u otro equipo especial) los pequeños bits en toda la imagen. Los humanos tienen una agudeza visual baja: no vemos tantos detalles como, por ejemplo, un águila. En realidad tenemos más de un tipo de sistema de visión en uso en nuestro cerebro y ojos. Nuestro sistema para procesar el color
es diferente de nuestro sistema para procesar blanco y negro (o lum inancia). Por ejemplo, usamos la luminancia para detectar el movimiento y los tamaños de los objetos. En realidad detectamos mejor los detalles de luminancia con los lados de nuestros ojos que con el centro.

Ésa es una ventaja evolutiva, ya que nos permite detectar cuando el tigre dientes de sable se esconde a nuestra derecha detrás de algunos arbustos, por ejemplo. La falta de resolución en la visión humana es lo que hace posible la digitalización de imágenes. Los animales que perciben un mayor detalle que los humanos (por ejemplo, las águilas o los gatos) podrían incluso ver los píxeles individuales. Descomponemos la imagen en ele mentos más pequeños (píxeles), pero hay suficientes de ellos y son lo bastante pequeños como para que la imagen no se vea entrecortada. Si podemos ver los efectos de la

digitalización (es decir, si podemos ver pequeños rectángulos en algunos puntos), entonces tenemos lo que se
conoce como pixelización: el efecto cuando el proceso de digitalización se vuelve obvio.

La luz visible es continua: la luz visible tiene cualquier longitud de onda entre 370 y 730 nanómetros (0.00000037 y 0.00000073 metros). Pero nuestra percepción de la luz se limita en cuanto a la forma en que funcionan nuestros sensores de colores. Nuestros ojos tienen sensores que se activan (llegan a un máximo) cerca de los 425 nanómetros (azul), 550 nanómetros (verde) y 560 nanómetros (rojo). Nuestro cerebro determina que un color específico se basa en la retroalimentación de estos tres sensores en nuestros ojos. Hay algunos animales con sólo dos tipos de sensores, como los perros. Estos animales pueden percibir el color, pero no los mismos colores ni de la misma forma que los humanos. Una de las implicaciones interesantes de nuestro aparato sensorial visual limitado es que en realidad percibimos dos tipos de naranja. Existe un naranja espectral: una longitud de onda específica que es el color naranja natural. Existe también una mezcla de rojo y amarillo que choca con nuestros sensores de colores de tal forma que lo percibimos como el mismo color naranja.

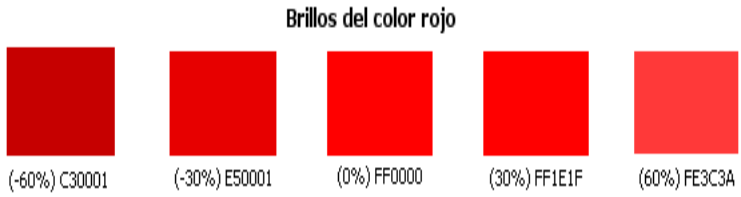
Mientras sigamos codificando lo que llegue a nuestros tres tipos de sensores de colores, estaremos registrando nuestra percepción humana del color. Así, codificamos cada pixel como una tercia de números. El primer número representa la cantidad de rojo en el pixel, el segundo es la cantidad de verde y el tercero es la cantidad de azul. Podemos crear cualquier color visible por el humano si combinamos la luz roja, verde y azul. Al combinar las tres luces obtenemos el color blanco puro. Si apagamos las tres obtenemos negro. A esto le llamamos el modelo de colores RGB.

Modificacion del Brillo

Es un término que se usa para describir que tan claro u oscuro parece un color, y se refiere a la cantidad de luz percibida. El brillo se puede definir como la cantidad de "oscuridad" que tiene un color, es decir, representa lo claro u oscuro que es un color respecto de su color patrón.

Es una propiedad importante, ya que va a crear sensaciones espaciales por medio del color. Así, porciones de un mismo color con un fuertes diferencias de valor (contraste de valor) definen porciones diferentes en el espacio, mientras que un cambio gradual en el valor de un color (gradación) da va a dar sensación de contorno, de continuidad de un objeto en el espacio.

El valor es el mayor grado de claridad u oscuridad de un color. Un azul, por ejemplo, mezclado con blanco, da como resultado un azul más claro, es decir, de un valor más alto. También denominado tono, es distinto al color, ya que se obtiene del agregado de blanco o negro a un color base.



A medida que a un color se le agrega mas negro, se intensifica dicha oscuridad y se obtiene un valor más bajo. A medida que a un color se le agrega más blanco se intensifica la claridad del mismo por lo que se obtienen valores más altos. Dos colores diferentes(como el rojo y el azul) pueden llegar a tener el mismo tono, si consideramos el concepto como el mismo grado de claridad u oscuridad con relación a la misma cantidad de blanco o negro que contengan, según cada caso.

La descripción clásica de los valores corresponde a claro (cuando contiene cantidades de blanco), medio (cuando contiene cantidades de gris) y oscuro (cuando contiene cantidades de negro). Cuanto más brillante es el color, mayor es la impresión de que el objeto está más cerca de lo que en realidad está.

Aumentar el brillo de una imagen consiste en sumar una constante a los colores que constituyen un pixel cuidando siempre de no rebasar los limites de 0 y 255. Dicho de otra forma, aumentar o disminuir el brillo consiste en aumentar o disminuir la ordenada al origen de la linea recta con pendiente 45º que representa los grises.

Codigo

El codigo para la modificacion de brillo al igual que los siguientes otros metodos de manipulaciones de imagenes estas escritos en lenguaje python.

```
from PIL import Image
import random
def bright(image,n):
	image = Image.open(image)
	px = image.load()
	width, height = image.size
	for y in range(height):
		for x in range(width):
			pxnew = []
			list(px[x,y])
			for i in range(3):
				pxnew.append(cambio(px[x,y][i],n))
			px[x,y] = (pxnew[0],pxnew[1],pxnew[2])
			tuple(px[x,y])
	image.save("{} - brillo - {}.jpg".format(random.randrange(100),n))


def cambio(value,cambio):
	new_value = value + cambio
	if new_value > 255:
		new_value = 255
		return new_value
	elif new_value < 0:
		new_value = 0
		return new_value
	else:
		return new_value
file = input("Ingrese la direccion de la imagen : ")
```

```
brillo = int(input("Ingrese la cantidad de brillo que desea : "))
bright(file, brillo)
```

Este codigo ejecuta un comando donde pide la ubicación de la imagen que se desea manipular, y la cantidad de brillo que se desea aumentar (valores positivos) o disminuir (valores negativos).

Una vez realizado todo el procedimiento del script, devolvera una imagen con los valores modificados.

Ejemplo

Izquierda : Original – Derecha : Brillo de +50

Izquierda : Original – Derecha : Brillo de -50

Modificacion del contraste

El contraste es la diferencia de luminosidad y tono entre las zonas claras y zonas oscuras de una imagen. Si la diferencia es grande, el contraste es mayor. Si la imagen tiene los tonos o brillos muy igualados el contraste es pequeño. El contraste “0” haria imperceptible una imagen, pues necesitariamos captar diferencias de tonos y color para reconocer las formas. El contraste maximo nos ofrece una imagen casi compuesta por dos zonas de color, una blanca y otra negra.

Aumentar o disminuir el contraste de una imagen consiste en aumentar o disminuir la pendiente de la linea recta con pendiente 45º que representa los grises cuidando de no rebasar los limites de 0 y 255.

Codigo

```
from PIL import Image, ImageEnhance
import random

def contraste(imagen,n):
        image = Image.open(imagen)
        contrast = ImageEnhance.Contrast(image)
        contrast.enhance(n).show()
        image.save("{} - contraste - {}.jpg".format(random.randrange(100),n))

file = input("Ingrese la direccion de la imagen : ")
contrast = int(input("Ingrese la cantidad de brillo que desea : "))
contraste(file, contrast)
```

Ejemplo

Izquierda : Original – Derecha : Contraste de +2

Izquierda : Original – Derecha : Contraste de -2

Inversion

Como el valor mas grande que puede tomar un color es de 255 y el mas pequeño es de 0, entonces si se desea invertir una imagen, se debe invertir las contribuciones de los diferentes pixeles a la formacion de dicha imagen, para esto se debe restar su color de 255 y esta diferencia tomarla como la contribucion de un color a la nueva imagen.

Codigo

```
from PIL import Image
import random

def inversion(image):
	image = Image.open(image)
#	image.show()
	px = image.load()
	width, height = image.size
	for y in range(height):
		for x in range(width):
			pxnew = []
			list(px[x,y])
			for i in range(3):
				pxnew.append(cambio(px[x,y][i]))
			px[x,y] = (pxnew[0],pxnew[1],pxnew[2])
			tuple(px[x,y])
	image.save("{} - inversion.jpg".format(random.randrange(100)))

def cambio(value):
	new_value = 255 - value
	return new_value

file = input("Ingrese la direccion de la imagen : ")
inversion(file)
```

Ejemplo

Escala de Grises

Cualquier pixel RGB de una imagen proyectada sobre el vector de grises nos data su contribucion gris a una nueva imagen que formaremos con todas las proyecciones de los pixeles originales.

$$CU=(r,g,b)*(1,1,1)$$

$$CU=|C|*|U|*\cos(\varphi)$$

(1,1,1) representa el vector de escala de grises

$$r+g+b=\sqrt{3(r^2+g^2+b^2)}*\cos(\varphi)$$

$$Proyeccion\,CU=\sqrt{r^2+g^2+b^2}*\cos(\varphi)$$

$$Proyeccion\,CU=\frac{r+g+b}{\sqrt{3}}$$

$$Proyeccion\,CU\,Normalizada=\frac{r+g+b}{\sqrt{3}}$$

Codigo

```
from PIL import Image, ImageOps
import random

def grey_scale(imagen):
        imagen = Image.open(imagen)
        imagen = ImageOps.grayscale(imagen)
        imagen.save("{} - Grey_Scale.jpg".format(random.randrange(100)))


file = input("Ingrese la direccion de la imagen : ")
grey_scale(file)
```

Ejemplo

Deteccion de Borde

Se puede cuantificar la diferencia entre 2 colores calculando la distancia geometrica de los vectores que la representan.

El objetivo de esta tecnica es la deteccion de la forma de una imagen y ser capaz de dibujar un bitmap resultante donde las orillas estan en blancos sobre un fondo negro.

Procedimiento : consiste en desplazarse por la imagen pixel por pixel comparando el color de cada uno con su vecino de la derecha y el de abajo. Si la diferencia es muy grande, el pixel corresponde a un borde y debe ser pintado de blanco.

$$C_1 = R_1 + G_1 + B_1$$

$$C_2 = R_2 + G_2 + B_2$$

$$(C_1, C_2) = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}$$

Codigo

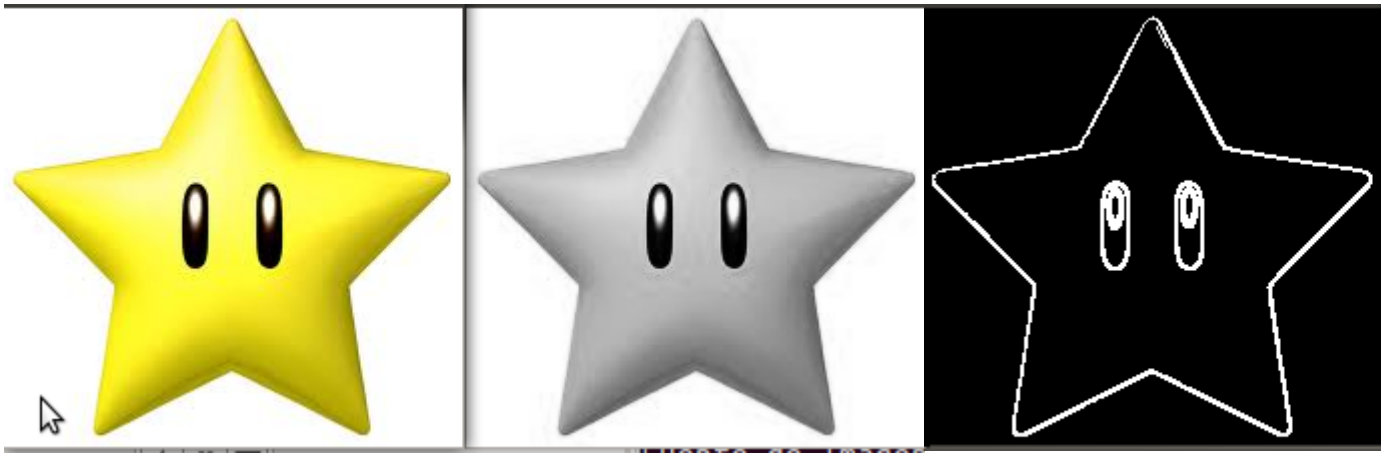
```
#!/usr/bin/python
#coding: utf-8
from PIL import Image
from numpy import array
import cv
import numpy as np

def main():
    im = Image.open('figuras.png')
    im.show() #mostramos imagen cargada con PIL
    arr_rgb = array(im) #convertimos en arreglo la imagen im
    arr_gry = np.zeros(shape = (arr_rgb.shape[0],arr_rgb.shape[1]))
    for n in (range(arr_rgb.shape[0])):    #Barremos por Filas
      for m in (range(arr_rgb.shape[1])):  #Barremos por Columnas
        arr_gry[n,m] = int((np.sum(arr_rgb[n,m]))/3)

    cv.SaveImage("Gray_figuras.png",cv.fromarray(arr_gry))
    im_gry = cv.LoadImage("Gray_figuras.png")
    cv.ShowImage('imagen',im_gry) #la visualizo con openCV
    cv.WaitKey(0)

if __name__ == '__main__':
    main()
```

Ejemplo



Bibliografia

- Introduccion a la compuatacion y Programacion – Mark J. Guzdial
- http://www.desarrolloweb.com/articulos/1503.php
- http://contenidos.educarex.es/mci/2002/24/actividades/6_9.html
- http://python-para-impacientes.blogspot.com.ar/2014/12/fundamentos-para-procesar-imagenes-con_18.html
- http://playingwpythonandvision.blogspot.com.ar/2013/07/deteccion-de-bordes.html