# Laboratorio 1: Comenzando con OpenCV

Alumno: Josue Samuel Philco Puma

21 de abril de 2025

## 1. Mostrando una imagen en formato RGB

Al comenzar a usar la biblioteca **OpenCV**, debemos tener en cuenta que este puede procesar imágenes, entonces primero debemos saber que usar para leer y mostrar una imagen.

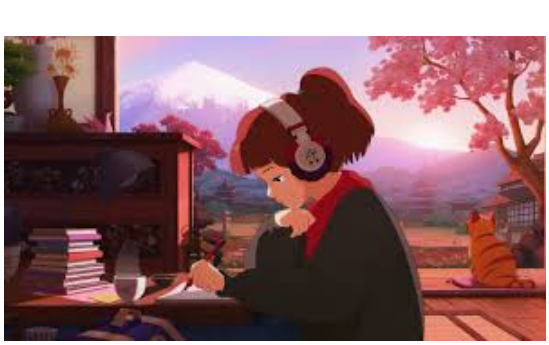
1. Usaremos una imagen de prueba para que se lea correctamente y muestre usando las siguientes funciones de **OpenCV** que son: **cv::imread** y **cv::imshow**.

2. Al usar la función **cv::imread** debemos pasarle como parámetro la ruta de nuestra imagen, esto lo que hará es procesar la imagen con la que se va a trabajar.

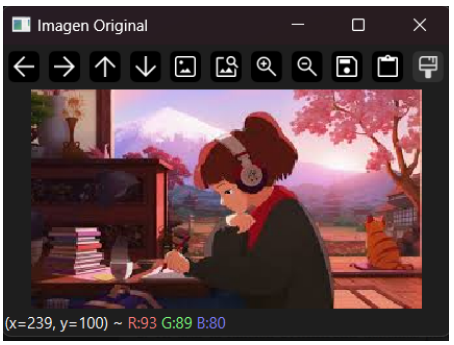
3. Para mostrar la imagen se usa la función **cv::imshow**, lo que hace es mostrar la imagen en una ventana y para que no se cierre rápido se usa la instrucción **waitKey(0)** para que no se cierre la ventana (se cerrará al presionar cualquier tecla).

```cpp
#include <iostream>
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;

int main() {
    string image_file;
    cout << "Ingrese el nombre de la imagen (con extension): ";
    cin >> image_file;
    string image_path = "D:/UNSA EPCC/7mo semestre/Computacion Grafica/Unidad 1/Imagenes con OpenCV/Imagenes/" + image_file;

    Mat image_original = imread(image_path);
    if (image_original.empty()) {
        cout << "No se pudo cargar la imagen." << endl;
        return -1;
    }

    imshow("Imagen Original", image_original);
    waitKey(0);
    return 0;
}
```

Entonces, con el código anterior veremos el resultado que es mostrar la imagen en formato **RGB**:



(a) Imagen original.




(b) Imagen mostrada con OpenCV.

Figura 1: Mostrando la imagen original (a) y la imagen con OpenCV (b)

## 2. Función para extraer y mostrar los canales de la imagen

Ahora que tenemos la imagen cargada y mostrándose debemos separar cada uno de sus canales de color y mostrar cada canal de forma separada. Esto se puede hacer explícitamente con las funciones que nos ofrece la biblioteca **OpenCV** que sería la función **cv::split**, pero también podemos hacerlo de forma manual accediendo a cada píxel y extraer los valores correspondientes. Para ello debemos saber que vamos a realizar:

1. **Primero debemos encontrar la forma de acceder a las filas y columnas de la imagen**: Esto lo haremos usando lo siguiente: **.rows** y **.cols**, esto lo que hace es obtener el número de filas y columnas de una imagen.
2. **Crear las matrices para cada canal:** Para canal debemos crear su propia matriz de color, entonces esto se hace con la instrucción **(rows, columns, CV_8UC1)**, este creará un canal de un color (escala gris), esto lo haremos para los canales **Azul, Verde y Rojo**.
3. **Recorrer las filas y columnas y acceder a cada píxel para obtener su valor:** Ahora que tenemos las filas y columnas podemos recorrerlas con dos bucles for y con la instrucción **image.at<Vec3b>(i, j)**, esto obtendrá el valor de cada pixel recorrido, en cada canal pondríamos un [0], [1] y [2] que representa los canales **Azul, Verde y Rojo**.
4. **Almacenar los valores en cada canal de color y mostrar la imagen:** Con los valores de los pixeles solo se va a almacenar cada valor en su respectivo canal, y con los valores almacenados solo mostrar la imagen en cada canal dividido.

Ahora con los pasos numerados que se debe realizar se muestra el código que lo realiza:

```
void show_channels(const Mat& image) {
    int columns = image.cols;
    int rows = image.rows;

    Mat blue_channel(rows, columns, CV_8UC1);
    Mat green_channel(rows, columns, CV_8UC1);
    Mat red_channel(rows, columns, CV_8UC1);

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            int channel_blue = image.at<Vec3b>(i, j)[0];
            int channel_green = image.at<Vec3b>(i, j)[1];
            int channel_red = image.at<Vec3b>(i, j)[2];

            blue_channel.at<uchar>(i, j) = channel_blue;
            green_channel.at<uchar>(i, j) = channel_green;
            red_channel.at<uchar>(i, j) = channel_red;
        }
    }

    imshow("Canal Azul", blue_channel);
    imshow("Canal Verde", green_channel);
    imshow("Canal Rojo", red_channel);
    waitKey(0);
}
```

Ahora se muestran los resultados obtenidos de la imagen en cada canal de color:

(a) Imagen original (b) Canal Azul (c) Canal verde (d) Canal Rojo

Figura 2: Extracción de canales de la imagen

# 3. Almacenar los valores obtenidos de cada canal

Ahora debemos almacenar los valores obtenidos de cada canal, entonces vamos a ver que hay que realizar:

1. **Inicializar valores mínimos y máximos en cada canal:** Como sabemos, debemos tener un valor mínimo y máximo para cada canal, por razonamiento se pensaría que el mínimo es 0 y el máximo es 255, pero como vamos a comparar cada valor, debemos inicializar el mínimo en 255 y el máximo en 0.
2. **Recorrer la imagen y en cada valor obtenido comparar con el mínimo y máximo:** Cuando estamos recorriendo la imagen, obtenemos los valores de los píxeles y se va comparando con el mínimo y máximo valor y actualizando su valor.
3. **Mostrar los resultados:** Cuando se termine de recorrer toda la imagen se muestran los resultados del mínimo y máximo entre el rango de 0 a 255.

Ahora con los pasos indicados hacemos el código respectivo:

```cpp
void show_channels(const Mat& image) {
    int columns = image.cols;
    int rows = image.rows;
    int minimun_value_blue = 255, maximun_value_blue = 0;
    int minimun_value_green = 255, maximun_value_green = 0;
    int minimun_value_red = 255, maximun_value_red = 0;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            int channel_blue = image.at<Vec3b>(i, j)[0];
            int channel_green = image.at<Vec3b>(i, j)[1];
            int channel_red = image.at<Vec3b>(i, j)[2];

            if (channel_blue < minimun_value_blue) {
                minimun_value_blue = channel_blue;
            }
            if (channel_blue > maximun_value_blue) {
                maximun_value_blue = channel_blue;
            }
            if (channel_green < minimun_value_green) {
                minimun_value_green = channel_green;
            }
            if (channel_green > maximun_value_green) {
                maximun_value_green = channel_green;
            }
            if (channel_red < minimun_value_red) {
                minimun_value_red = channel_red;
            }
            if (channel_red > maximun_value_red) {
                maximun_value_red = channel_red;
            }
        }
    }

    cout << "Valores de los canales:" << endl;
    cout << "Canal Azul: Minimo: " << minimun_value_blue << ", Maximo: " <<
        maximun_value_blue << endl;
    cout << "Canal Verde: Minimo: " << minimun_value_green << ", Maximo: " <<
        maximun_value_green << endl;
    cout << "Canal Rojo: Minimo: " << minimun_value_red << ", Maximo: " <<
        maximun_value_red << endl;
}
```

Con esto podemos obtener los valores mínimos y máximos de cada canal y están compredidas desde 0 a 255, esto se visualiza en terminal:

```
LENOVO@DESKTOP-QBBTJM9 MINGW64 /d/UNSA EPCC/7mo semestre/Computacion G
penCV
$ ./Extract_Show_Channels.exe
Ingrese el nombre de la imagen (con extension): image_lofi.jpg
Valores de los canales:
Canal Azul: Minimo: 0, Maximo: 255
Canal Verde: Minimo: 0, Maximo: 255
Canal Rojo: Minimo: 8, Maximo: 255
QThreadStorage: entry 1 destroyed before end of thread 0x247ee0bcdb0
QThreadStorage: entry 0 destroyed before end of thread 0x247ee0bcdb0
```

Figura 3: Resultado de valores obtenidos en cada canal.

## 4. Código completo

Entonces con todas las actividades desarrolladas se presenta el código completo con todos los pasos indicados:

```cpp
#include <iostream>
#include <opencv2/opencv.hpp>
using namespace std;
using namespace cv;

void show_channels(const Mat& image) {
    int columns = image.cols;
    int rows = image.rows;
    int minimun_value_blue = 255, maximun_value_blue = 0;
    int minimun_value_green = 255, maximun_value_green = 0;
    int minimun_value_red = 255, maximun_value_red = 0;

    Mat blue_channel(rows, columns, CV_8UC1);
    Mat green_channel(rows, columns, CV_8UC1);
    Mat red_channel(rows, columns, CV_8UC1);

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            int channel_blue = image.at<Vec3b>(i, j)[0];
            int channel_green = image.at<Vec3b>(i, j)[1];
            int channel_red = image.at<Vec3b>(i, j)[2];

            blue_channel.at<uchar>(i, j) = channel_blue;
            green_channel.at<uchar>(i, j) = channel_green;
            red_channel.at<uchar>(i, j) = channel_red;

            if (channel_blue < minimun_value_blue) {
                minimun_value_blue = channel_blue;
            }
            if (channel_blue > maximun_value_blue) {
                maximun_value_blue = channel_blue;
            }
            if (channel_green < minimun_value_green) {
                minimun_value_green = channel_green;
            }
            if (channel_green > maximun_value_green) {
                maximun_value_green = channel_green;
            }
            if (channel_red < minimun_value_red) {
                minimun_value_red = channel_red;
            }
            if (channel_red > maximun_value_red) {
                maximun_value_red = channel_red;
            }
        }
    }
```

```
47
48      imshow("Canal Azul", blue_channel);
49      imshow("Canal Verde", green_channel);
50      imshow("Canal Rojo", red_channel);
51      waitKey(0);
52
53      cout << "Valores de los canales:" << endl;
54      cout << "Canal Azul: Minimo: " << minimun_value_blue << ", Maximo: " <<
            maximun_value_blue << endl;
55      cout << "Canal Verde: Minimo: " << minimun_value_green << ", Maximo: " <<
            maximun_value_green << endl;
56      cout << "Canal Rojo: Minimo: " << minimun_value_red << ", Maximo: " <<
            maximun_value_red << endl;
57  }
58
59  int main() {
60      string image_file;
61      cout << "Ingrese el nombre de la imagen (con extension): ";
62      cin >> image_file;
63      string image_path = "D:/UNSA EPCC/7mo semestre/Computacion Grafica/Unidad
            1/Imagenes con OpenCV/Imagenes/" + image_file;
64
65      Mat image_original = imread(image_path);
66      if (image_original.empty()) {
67          cout << "No se pudo cargar la imagen." << endl;
68          return -1;
69      }
70
71      imshow("Imagen Original", image_original);
72      waitKey(0);
73
74      show_channels(image_original);
75      return 0;
76  }
```

# Referencias

[1] OpenCV. *Reading and Writing Images and Video*. Disponible en: https://docs.opencv.org/2.4/modules/highgui/doc/reading_and_writing_images_and_video.html#imread

[2] StackOverflow. *How to check whether my image is RGB or BGR format in Python?*. Disponible en: https://stackoverflow.com/questions/59581538/how-to-check-whether-my-image-is-rgb-format-or-bgr-format-in-python-how-do-i-co

[3] HetPro. *OpenCV cvtColor*. Disponible en: https://hetpro-store.com/TUTORIALES/opencv-cvtcolor/

[4] StackOverflow. *OpenCV get pixel channel value from Mat image*. Disponible en: https://stackoverflow.com/questions/7899108/opencv-get-pixel-channel-value-from-mat-image

[5] OpenCV Answers. *How to get pixels value from a picture?*. Disponible en: https://answers.opencv.org/question/12963/how-to-get-pixels-value-from-a-picture/

[6] TutorialsPoint. *How to get the value of a specific pixel in OpenCV using C++?*. Disponible en: https://www.tutorialspoint.com/how-to-get-the-value-of-a-specific-pixel-in-opencv-using-cplusplus

[7] OpenCV Blog. *Read, Display and Write an Image Using OpenCV*. Disponible en: https://opencv.org/blog/read-display-and-write-an-image-using-opencv/