

FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES /CENTROS DE SIMULACIÓN – PARA DOCENTES

CARRERA: COMPUTACIÓN/INGENIERÍA DE SISTEMAS

ASIGNATURA: VISIÓN POR COMPUTADOR

**NRO.
PRÁCTICA:**

1-1

TÍTULO PRÁCTICA: Fundamentos del procesamiento digital de imágenes: Espacios de Color, Histogramas y Clasificación Básica

OBJETIVO:

Reforzar los conocimientos adquiridos en clase sobre la conversión de espacios de color, cálculo de histogramas y el procesamiento de vídeo.

INSTRUCCIONES:

- 1.** Revisar el contenido teórico del tema
- 2.** Profundizar los conocimientos revisando los libros guías, los enlaces contenidos en los objetos de aprendizaje y la documentación disponible en fuentes académicas en línea
- 3.** Deberá desarrollar un conjunto de *scripts* y programas que permitan realizar el procesamiento digital de imágenes y llevar a cabo la clasificación de imágenes empleando operaciones de resta de imágenes tanto en escala de grises como a color.
- 4.** La práctica se subdividirá en tareas específicas que guardarán relación con la representación de varios conceptos como: el cálculo de histogramas, representación de imágenes en distintos espacios de color y la operación sobre secuencias de imágenes (vídeos).

ACTIVIDADES POR DESARROLLAR

• **Parte 1.** Desarrollar un programa en C++ que permita procesar de forma iterativa un directorio dado usando OpenCV a fin de realizar la conversión del espacio de color RGB (Red, Green, Blue) a Gris. Para ello, se debe ejecutar las siguientes actividades (empleando el conjunto de imágenes “*imágenes-práctica-1.zip*”):

1. Desarrollar un programa para realizar la conversión del espacio de color RGB a Gris de un conjunto de imágenes y calcular los tiempos que se requiere para realizar la operación de conversión.

Para la resolución del problema se ha realizado la lectura de las imágenes directamente transformado al leerlas usando el comando **IMREAD_GRAYSCALE**.

```
void convertirRGB2GRAY(int i){
    DIR *direccion;
    struct dirent *elementos;
    if(direccion=opendir(carpetas[i].c_str())){
        while (elementos=readdir(direccion)){
            string elemento = elementos->d_name;
            if(elemento!="." && elemento!=".." && elemento.length() > 2 ){
                string pathImagen = carpetas[1] + elemento;
                string gray_elemento = elemento.substr(0,elemento.length()-4)+".png";
                Mat img = imread(pathImagen, IMREAD_GRAYSCALE);
                imwrite(carpetas_gray[i]+gray_elemento,img);
            }
        }
        closedir(direccion);
    }
}
```

Fig 1. Método para obtención de imagen a escala de grises y guardarla.

```
*****
| UNIVERSIDAD POLITÉCNICA SALESIANA
*****
Proceso: conversión de color a gray-scale
Carpeta: ant
Tiempo demorado: 0.091176
-----
Carpeta: motorbikes
Tiempo demorado: 0.056744
-----
Carpeta: sunflower
Tiempo demorado: 0.042508
-----
Carpeta: watch
Tiempo demorado: 0.033441
-----
```

Fig 2. Tiempo transcurrido para la transformación de imágenes.



Fig 3. Carpetas contenedoras de imágenes en escala de grises

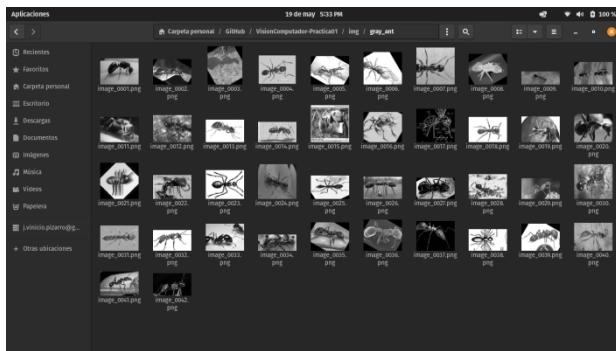


Fig 4. Carpeta gray_ant



Fig 5. Carpeta gray_motorbikes

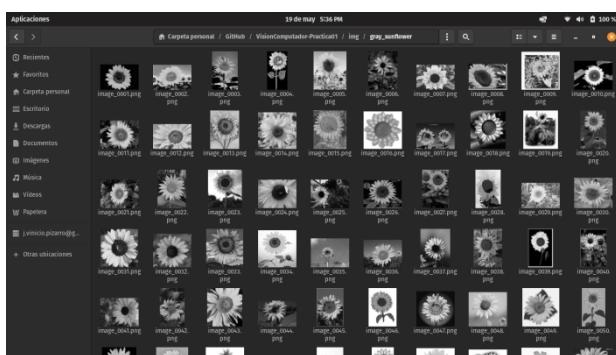


Fig 6. Carpeta gray_sunflower

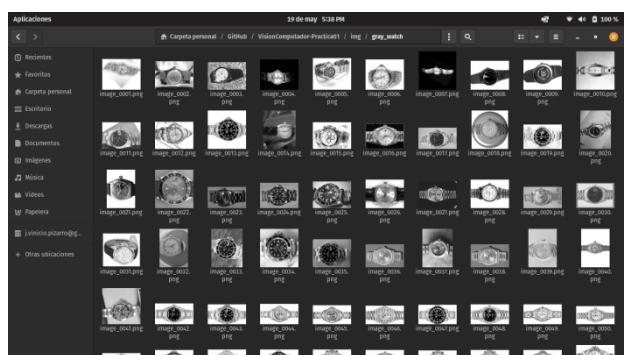


Fig 7. Carpeta gray_watch

- Desarrollar un programa para calcular el histograma (a color y en escala de grises) del conjunto de imágenes y calcular los tiempos que se requiere para la operación.

Resolvimos este punto creando un 4 vectores (1 escala de grises, 3 para los canales BGR; 1 para cada canal) en los cuales se almacena el numero de veces que aparece un pixel en el canal inspeccionado. Posteriormente se grafican cada canal con su color característico para posteriormente almacenarlas en una carpeta.

Según el enunciado nos solicita realizar el histrograma del conjunto de imágenes por carpeta por lo que se procese en estos vectores hacer la suma de todas las imagenes por carpeta.

```

void histograma(int histo[], int size, Mat imagen){
    for(int i=0;i<size;i++){
        histo[i] = 0;
    }
    int pixel = 0;
    for(int i=0;i<imagen.rows;i++){
        for(int j=0;j<imagen.cols;j++){
            pixel = imagen.at<uchar>(i,j);
            histo[pixel]++;
        }
    }
}

```

Fig 8. Método para contar pixeles de una imagen de 1 canal.

```

void dibujarHistograma(int histo[], int size, Scalar color, string nombre){
    namedWindow(nombre, WINDOW_AUTOSIZE);
    int maximo_gray = maximo(histo, size);
    int ancho = 1024;
    int alto = 320;
    Mat histo_img = Mat(Size(ancho, alto), CV_8UC3, Scalar(0,0,0));
    double escala = ((double)alto) / ((double) maximo_gray);
    for(int i=0;i<256;i++){
        line(histo_img, Point(i*3,alto), Point(i*3, alto-escala*histo[i]), color, 2);
    }
    imwrite("./img/histogramas/"+nombre + ".png",histo_img);
    imshow(nombre, histo_img);
}

```

Fig 9. Método para graficar un histograma.

```

void calculoHistograma(int i){
    int size = 256;
    int histograma_gray[256];
    int histograma_color_b[256], histograma_color_g[256], histograma_color_r[256];
    DIR *direccion;
    struct dirent *elementos;
    if(direccion=opendir(carpetas[i].c_str())){
        while (elementos=readdir(direccion)){
            string elemento = elementos->d_name;
            if(elemento!="." && elemento!=".." && elemento.length() > 2 ){
                string pathImagen = carpetas[i] + elemento;
                Mat imagen_color = imread(pathImagen);
                vector<Mat> canales;
                split(imagen_color,canales);
                histograma(histograma_color_b, size, canales[0]);
                histograma(histograma_color_g, size, canales[1]);
                histograma(histograma_color_r, size, canales[2]);
                Mat imagen_gray = imread(pathImagen, IMREAD_GRAYSCALE);
                histograma(histograma_gray, size, imagen_gray);
            }
        }
    }
    closedir(direccion);
    dibujarHistograma(histograma_gray, 256, Scalar(255,255,255), "Histograma - Gray Scale "+nombre_carpetas[i]);
    dibujarHistograma(histograma_color_b, 256, Scalar(255,0,0), "Histograma - Azul "+nombre_carpetas[i]);
    dibujarHistograma(histograma_color_g, 256, Scalar(0,255,0), "Histograma - Verde "+nombre_carpetas[i]);
    dibujarHistograma(histograma_color_r, 256, Scalar(0,0,255), "Histograma - Rojo "+nombre_carpetas[i]);
    waitKey();
    destroyAllWindows();
}

```

Fig 10. Método metodo principal para calcular los histogramas de *escala de grises y BGR*.

3. Generar un informe (de forma automática) en formato de texto donde se indiquen los tiempos requeridos para procesar la N imágenes y realizar las operaciones de conversión de color y cálculo del histograma.

Se arma un ***string*** para agregar los tiempos de cada proceso, para guardar el archivo en .txt se usa la librería ***ofstream***.

```

string reporte;
reporte = "*****\n";
reporte += "\t UNIVERSIDAD POLITÉCNICA SALESIANA\n";
reporte += "*****\n";
reporte += "Proceso: conversión de color a gray-scale\n";
for(int i = 0; i<4; i++){
    reporte += "Carpeta: " + nombre_carpetas[i] + "\n" +
               "Tiempo demorado: " + to_string(tiempos[0][i]) + "\n" +
               "-----\n";
}
reporte+="*****\n" + "Proceso: calculo histograma";
for(int i = 0; i<4; i++){
    reporte += "Carpeta: " + nombre_carpetas[i] + "\n" +
               "Tiempo demorado: " + to_string(tiempos[0][i]) + "\n" +
               "-----\n";
}
reporte+="Realizado por:\n \twilliam Mendiera \n\tJorge Pizarro\n";
reporte+="*****\n";
cout << reporte << endl;
ofstream file;
file.open("./partel/Reporte.txt", std::fstream::in | std::fstream::out | std::fstream::app);
file << reporte;
file.close();

return EXIT_SUCCESS;

```

Fig 11. Código para generar reporte .txt .

4. Debe incluir un informe en formato PDF con el código desarrollado y los resultados de tiempos de comparación y reflexionar sobre los resultados obtenidos.

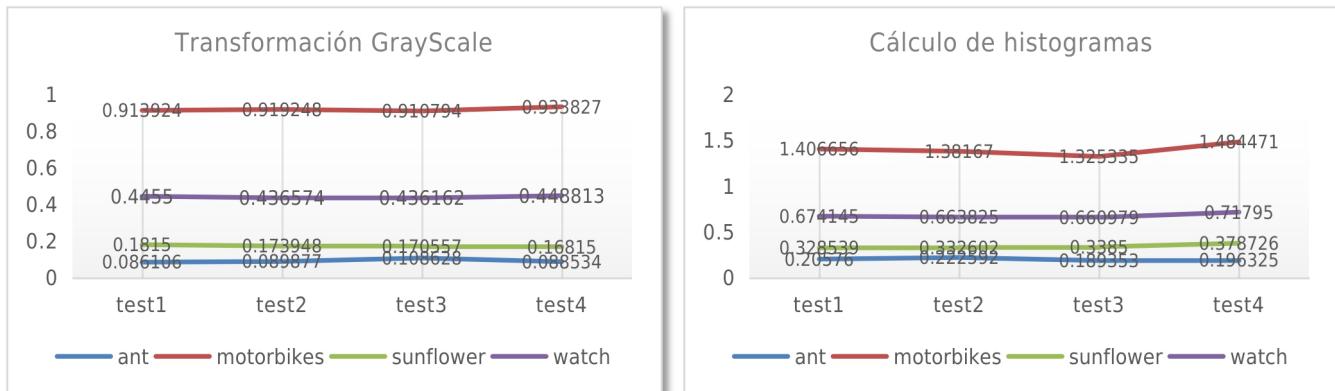


Fig 12. Presentación de tiempos de cada operación por carpeta.

Conclusiones: hemos realizado 4 pruebas para tener respaldo de tiempo de ejecución, así que el tiempo máximo de ejecución llega al minuto y medio, y como muestran las gráficas mientras mas archivos mayor en tiempo de ejecución.

• **Parte 2.** Desarrollar un programa que permita realizar un proceso de clasificación de imágenes en base a la resta de imágenes. Para ello, deberá considerar los siguientes aspectos:

- Separar las imágenes en dos partes (respetando las carpetas que representan las clases): entrenamiento (que se empleará para formar los clústers) y test (para determinar el nivel de precisión del programa).

```
vector<Imagen> trainAnt;
vector<Imagen> testAnt;
cargarImagenes(trainAnt, testAnt, dirAnt);

vector<Imagen> trainMotorBike;
vector<Imagen> testMotorBike;
cargarImagenes(trainMotorBike, testMotorBike, dirMotorBikes);

vector<Imagen> trainSunFlower;
vector<Imagen> testSunFlower;
cargarImagenes(trainSunFlower, testSunFlower, dirSunFlower);

vector<Imagen> trainWacth;
vector<Imagen> testWacth;
cargarImagenes(trainWacth, testWacth, dirWacth);
```

Fig 13. Vectores para almacenar las iamgenes de Train y Test.

```
void cargarImagenes(vector<Imagen>& train, vector<Imagen>& test,
                    vector<string> dir){
    int limit =(int)(dir.size()*0.8);
    for(int i = 0; i < dir.size() ; i++){
        Imagen imagen;
        imagen.nombre=dir[i];
        Mat img = imread(dir[i]);
        resize(img, img, Size(ancho, alto), INTER_LINEAR);
        imagen.imagenColor=img;
        vector<Mat> bgr_canales;
        split( img, bgr_canales );
        imagen.b=bgr_canales[0];
        imagen.g=bgr_canales[1];
        imagen.r=bgr_canales[2];
        cvtColor(img,img,COLOR_BGR2GRAY);
        imagen.imagenGray=img;
        if(i < limit){
            train.push_back(imagen);
        }else{
            test.push_back(imagen);
        }
    }
}
```

Fig 14. Función para distribución de las imagenes de Train y Test.

- Trabajar con un conjunto de imágenes y escalarlas a un tamaño en el que se pueda realizar la resta de forma correcta.

```
int ancho = 300;
int alto = 200;
string carpetas[4] = {"./img/ant/*", "./img/motorbikes/*", "./img/sunflower/*", "./img/watch/*"};
string reporte = "";
```

Fig 15. Variables locales para definición de ancho y largo de imagenes.

```
resize(img, img, Size(ancho, alto), INTER_LINEAR);
```

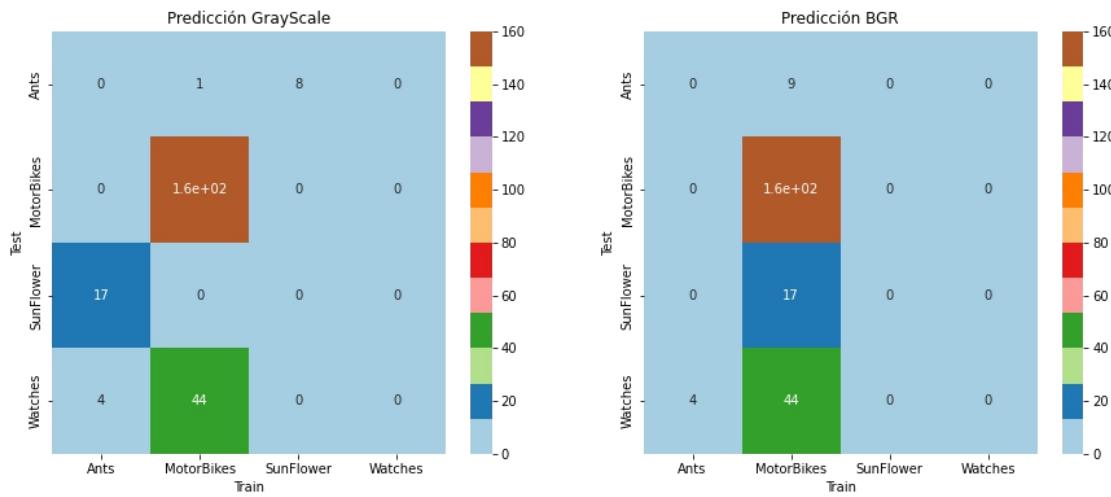
Fig 16. Uso de las variables de *largo* y *ancho*

```
int restarImagen(Mat& img1, Mat& img2){
    Mat resta;
    //subtract(img1,img2,resta);
    absdiff(img1,img2,resta);
    int resultado=0;
    for(int i = 0; i< resta.rows; i++){
        for (int j = 0; j < resta.cols; j++)
        {
            resultado+=(int)resta.at<uchar>(i,j);
            //cout << (int) resta.at<uchar>(i,j) << endl;
        }
    }
    //cout << resultado << endl;
    return resultado;
}
```

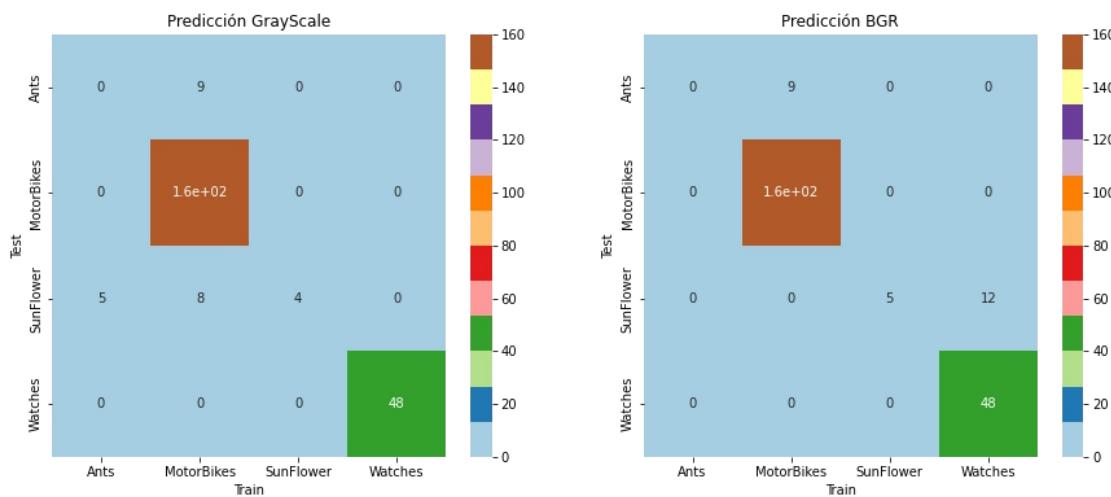
Fig 17. Función para calcular la resta de imagenes(canales) y retornar el valor de la sumatoria.

3. Comparar el nivel de precisión que se tiene cuando se clasifica imágenes a colores frente a imágenes en escala de grises. Para ello, se tomará cada imagen del test y se calculará la resta para ver con qué cluster se parece más. Si el cluster corresponde a la misma categoría de la imagen, será un acierto, caso contrario un fallo.

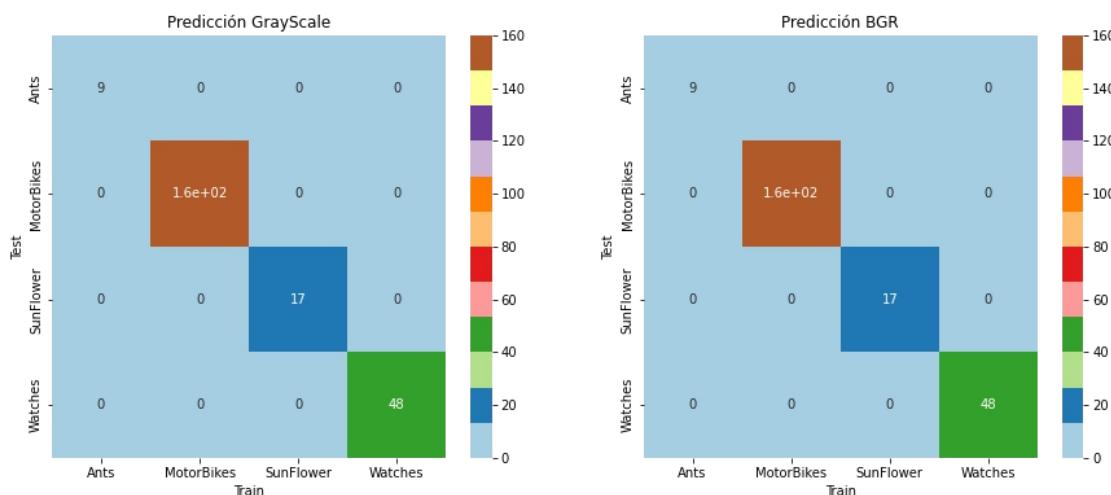
Prueba 1

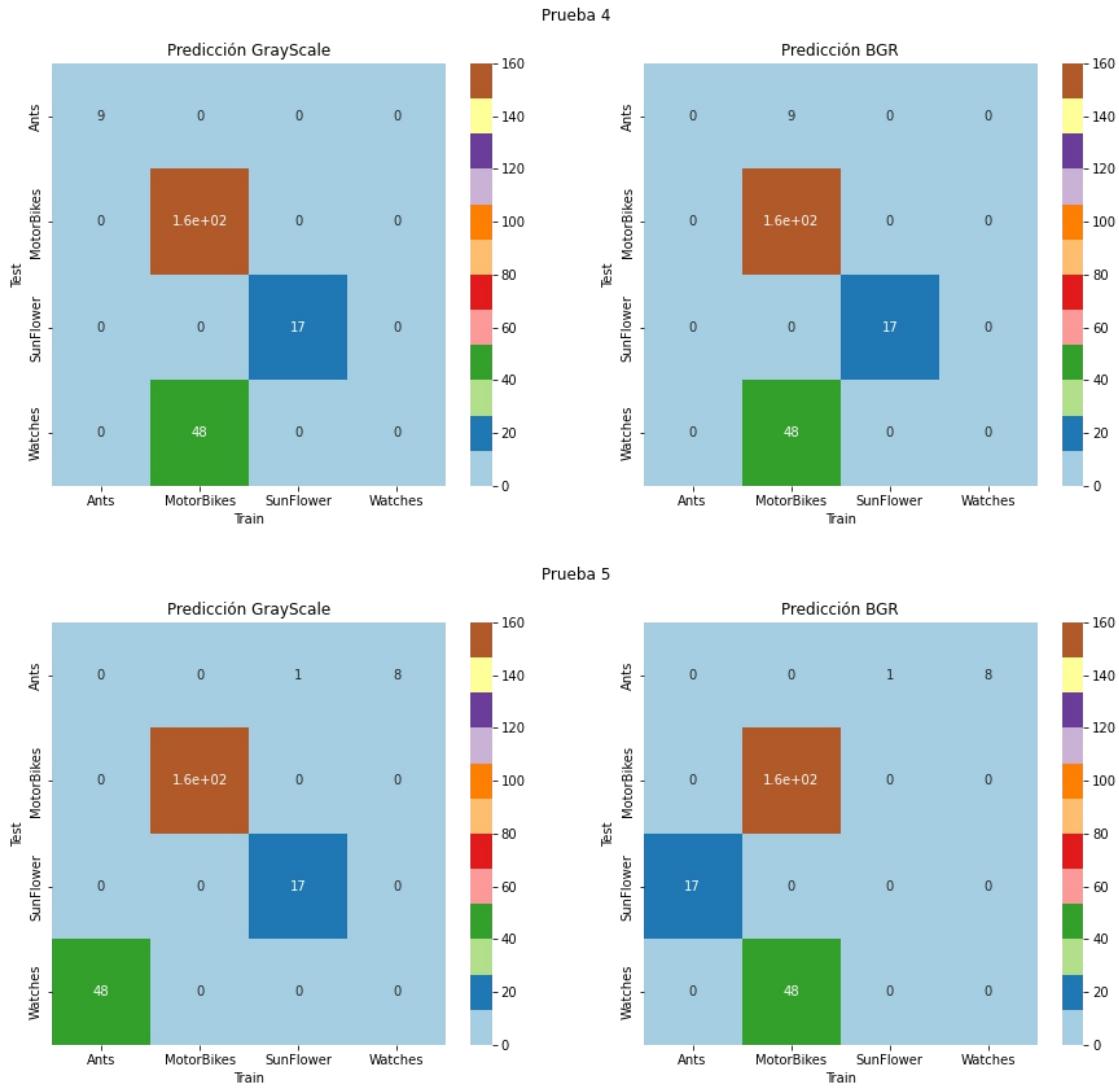


Prueba 2



Prueba 3





Una vez terminado la implementación de nuestro código se han realizado las 5 pruebas con cluster de train y test aleatorios en cada ejecución, lo que nos ha permitido visualizar variaciones diferencias de precisión del cual la prueba 3 tiene una presición del 100% tanto en GrayScale como a Color usando RBG como espacio de color. Sin embargo, en todas las prueba se puede verificar que se tiene un 100% de eficiencia en las imágenes de **MotorBikes**. La diferencia de entre presición de debe a la cantidad de imágenes disponibles para el train ya que estos tiene el mayor numero de imágenes a diferencia de los otros conjuntos de datos.

Por su lado en cuando a espacio de colores tomando en cuenta la 5ta prueba y refiriendo al conjunto de datos de SunFlower el espacio de color que tiene una mejor presición es la de GrayScale.

- **Parte 3.** Desarrollar un programa que permita realizar la conversión dinámica del espacio de color imágenes que se captan desde la cámara. Para ello es necesario considerar los siguientes aspectos:

1. Deberá existir una ventana donde se muestra la imagen original y otra donde se muestra el resultado de realizar al conversión a los siguientes espacios de color: HSV, LAB, YcrCb y Escala de Grises.

```

int main(){
    VideoCapture video("/dev/video0");
    if(video.isOpened()){
        namedWindow("Video", WINDOW_AUTOSIZE);
        while(3==3){
            video >> frame;
            if(frame.empty()){break;}
            resize(frame, frame, Size(), 1.2, 1.2);
            flip(frame, frame, 1); //Utilizado para invertir la imagen
            createTrackbar("Espacio", "Video", &thres, 3, eventoTrack, NULL);
            setMouseCallback("Cambio", eventoRaton, NULL);
            if(thres==0){
                cvtColor(frame, nuevoEspacio, COLOR_BGR2GRAY);
            }else if(thres==1){
                cvtColor(frame, nuevoEspacio, COLOR_BGR2HSV);
            }else if(thres==2){
                cvtColor(frame, nuevoEspacio, COLOR_BGR2Lab);
            }else if(thres==3){
                cvtColor(frame, nuevoEspacio, COLOR_BGR2YCrCb);
            }
            imshow("Video", frame);
            imshow("Cambio", nuevoEspacio);
            if(waitKey(33)==27){break;}
        }
        video.release();
        destroyAllWindows();
    }
    return EXIT_SUCCESS;
}

```

2. La ventana deberá tener un botón o una barra de desplazamiento que permita seleccionar el espacio de color al que se convertirá la imagen original. Al seleccionar el espacio de color se muestra una imagen copia en una segunda ventana con el espacio de color seleccionado.

```

void eventoTrack(int v, void *data){}
createTrackbar("Espacio", "Video", &thres, 3, eventoTrack, NULL);

```

3. Al dar click en la ventana donde está representada la imagen en los otros espacios de color (HSV, LAB, YcrCB), deberá guardarse la imagen actual.

```

void eventoRaton(int e, int x, int y, int b, void *data){
    //cout << "x: " << x << " y: " << y << endl;
    if(b==33){
        imwrite("./"+espacios.at(thres)+".png",nuevoEspacio);
    }
}

```



Pudimos apreciar que al aplicar un determinado espacio de color obtenemos como imagen resultante una variación de colores con respecto a la imagen original, en el caso de aplicar HSV podemos aplicar como una saturación, tinte y brillo a la imagen que se proyecta, así mismo con los demás espacios de colores, en el caso de LAB nos ayuda a resaltar un objeto en específico determinando su color.

Cuando aplicamos YcrCB lo que hacemos es obtener una visión clara de un objeto en comparación con su alrededor.

Código: <https://github.com/JPizarro92/VisionComputador-Practica01.git>

Integrantes: Willam Mendieta - Jorge Pizarro

RESULTADO(S) OBTENIDO(S):

Obtiene el histograma de una imagen digital, transforma imágenes digitales entre los distintos espacios de color, comprende las características de los distintos espacios de color

CONCLUSIONES:

Los estudiantes comprenden los principales fundamentos de la formación de imágenes digitales, cómo obtener los histogramas, realizar operaciones de conversión de espacios de color y cómo emplearestas características para distintas tareas de la visión artificial, como la clasificación de imágenes.

RECOMENDACIONES:

Revisar la información proporcionada por el docente previo a la práctica.
Haber asistido a las sesiones de clase.
Consultar con el docente las dudas que puedan surgir al momento de realizar la práctica.

Docente / Técnico Docente: Ing. Vladimir Robles, Bykbaev

Firma: _____