Álgebra Lineal
RC1
Pregrado
2023-2

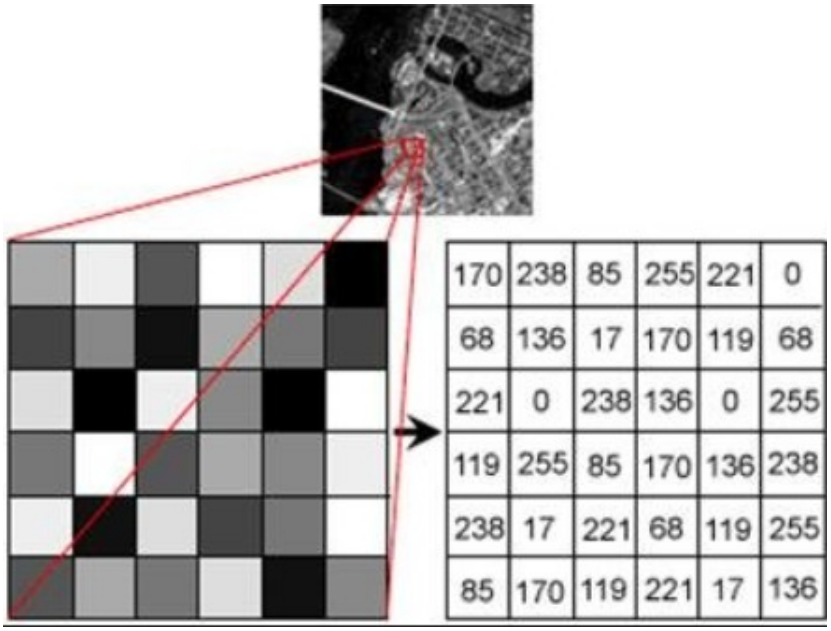
RC1- Caos en imagenes: Entrega 1/6

Esta resolución de caso 1 RC1 consta de cuatro secciones junto con algunas preguntas adicionales que son opcionales. El procedimiento implica la creación de un archivo LiveScript en MATLAB que contenga los códigos y los resultados obtenidos, y posteriormente, exportarlo en formato PDF. Para concluir la tarea, es necesario proporcionar dos elementos: el archivo .mxl y el archivo en formato PDF.

1. Convolución y procesamiento de imagenes

El procesamiento de imágenes se puede definir como el uso de operaciones matemáticas aplicadas a una imagen con el objetivo de mejorarla en algún sentido u obtener parámetros de la misma. Con la introducción de los ordenadores, el procesamiento se realiza por medio de algoritmos gráficos a las imágenes digitales, las cuales se obtienen mediante un proceso de digitalización o directamente utilizando cualquier dispositivo digital. A este procesamiento se le llama procesamiento digital de imágenes.

Toda imagen digital en blanco y negro no es más que una matriz numérica de NM píxeles, distribuidos en N píxeles por fila y M píxeles por columna. En cada píxel se asocia un número que indica su nivel de blanco (0=negro absoluto, pasando por varios niveles de gris, hasta llegar al blanco absoluto).



Por otro lado, una imagen digital en color es la superposición de 3 imágenes: R, G, B que contienen, respectivamente: nivel de rojo, verde y azul.

Así pues, toda fotografía digital es una matriz de M filas y N columnas, o bien la superposición de 3 de este tipo. En otras palabras: toda manipulación de fotografía digital no es más que una operación entre matrices. Quien domine la teoría de matrices, dominará las transformaciones de imágenes digitales .

2. Convolución

El primer procesamiento de imágenes se basó en filtros que permitían, por ejemplo, obtener los bordes de un objeto en una imagen utilizando la combinación de filtros de borde vertical y borde horizontal. Estos filtros utilizan la operación matricial conocida como "producto convolucional". A continuación pasaremos a definir de forma sencilla esta operación. Nótese que en todo momento estaremos hablando de una imagen como si fuera una matriz según la representación que explicamos anteriormente.

Producto convolucional de matrices.
Considere dos matrices cuadradas $A = [a_{ij}]$ y $B = [b_{ij}]$ de orden 3. Diremos que el paso del producto convolucional entre ellas es el número

$$\sum_{ij} a_{ij} * b_{ij}$$

Esto es, la suma de los productos entrada por entrada de las mencionadas matrices. Sabiendo

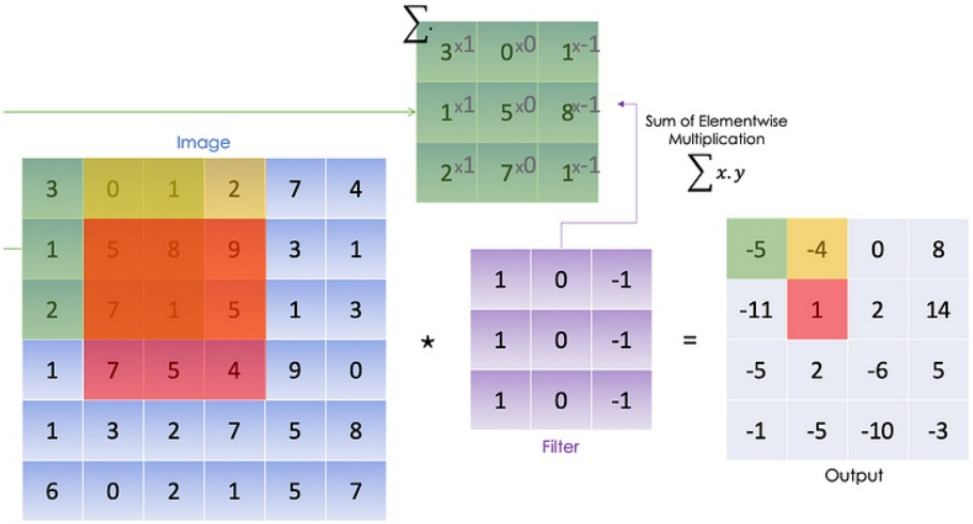


Figura 1: Representación convolucional de una matriz de A de orden 6×6 usando una matriz B de orden 3×3. La matriz de salida muestra tres de sus elementos coloreados que son los resultados de la convolución de los elementos correspondientes de A con B

.

esto, ahora podemos definir el producto convolucional aplicado a una imagen (vista como matriz). Para ello tomemos una matriz cuadrada B de orden n la cual se mantendrá fija durante todo el proceso. La idea de realizar el producto convolucional es tomar submatrices cuadradas de orden n de la matriz de la imagen e ir "multiplicándola", usando el producto descrito anteriormente, con la matriz B. El resultado de este producto reemplazará el valor del centro de nuestra submatriz. Este paso se realiza para cada submatriz hasta modificar por completo la matriz original. En la Figura (1) se ilustra mejor esta operación.

3. Ejercicio

3.1. Parte 1: Conversión de Imagen a Escala de Grises

Deseas realizar una conversión de una imagen en color a una imagen en escala de grises. Para lograr esto, debes implementar un programa en MATLAB que realice el proceso de conversión siguiendo el método ponderado estándar para obtener la intensidad de gris a partir de los canales de color rojo, verde y azul (RGB) de cada píxel de la imagen.

Para completar la tarea, sigue los pasos a continuación:

1. Lee una imagen (JPG) a formato RGB.
2. Obtiene las dimensiones de la imagen (alto y ancho).
3. Inicializa una matriz para almacenar la imagen en escala de grises.
4. Calcula el valor de gris para cada píxel utilizando la siguiente fórmula:

$$Gris = 0{,}2989 * Rojo + 0{,}5870 * Verde + 0{,}1140 * Azul$$

   donde $Gris$, $Rojo$, $Verde$ y $Azul$ son las matrices correspondientes a los canales RGB de la imagen JPG.
5. Muestra la imagen en escala de grises resultante

Use: Los comandos de Matlab **imread, imshow, uint8**

3.2. Parte 2: Ajuste de Brillo y Contraste de una Imagen

Estás trabajando en la mejora de imágenes y necesitas implementar un programa en MATLAB que realice el ajuste de brillo y contraste en una imagen a color. El objetivo es aplicar transformaciones a los valores de los píxeles de manera que puedas controlar el brillo y el contraste de la imagen final.

Para completar la tarea, sigue los siguientes pasos:

1. Lee una imagen en formato RGB.
2. Define valores para el ajuste de brillo (`brightness` $\in [0; 1]$) y el ajuste de contraste (`contrast`$\in [1; 2]$).
3. Utiliza el ajuste de brillo y contraste para transformar los valores de los píxeles de la imagen original. Para ello, multiplica cada valor de píxel por el valor de contraste y luego agrega el valor de brillo multiplicado por 255.
4. Asegúrate de que los valores de píxeles ajustados estén dentro del rango válido de $[0, 255]$. Utiliza las funciones `max` y `min` para aplicar la limitación.
5. Convierte la matriz resultante en valores de tipo `uint8` para el formato adecuado de visualización.
6. Muestra la imagen resultante con el brillo y contraste ajustados.

Tu tarea consiste en implementar el código en MATLAB siguiendo estos pasos y verificar que el ajuste de brillo y contraste se realice de manera efectiva en la imagen.

Puedes utilizar el código que proporcionaste como referencia para llevar a cabo esta tarea.

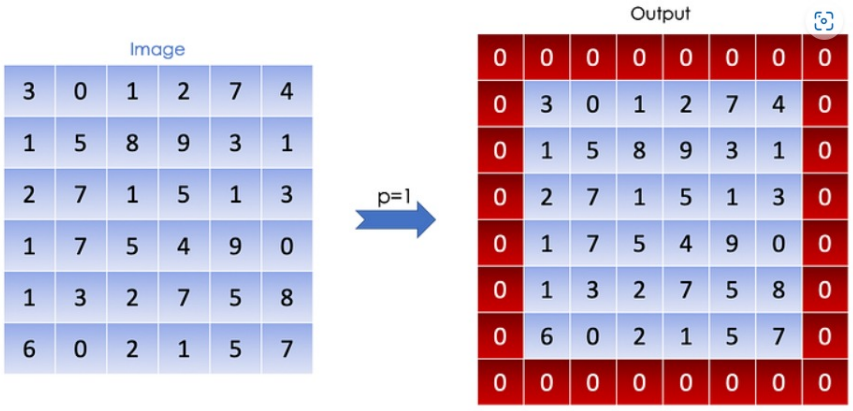
3.3. Parte 3: Convolución

Usa la imagen en escala de Grises de la Parte 2 Luego defina una matriz *B filtro* de orden 3x3 y realice el producto convolucional de la imagen con dicha matriz.

Para esto debes realizar los siguientes pasos

Relleno de imagen

Como los bordes de la imagen no pueden ser el centro de ninguna submatriz y por lo tanto no pueden seer modificadas, a menudo agregamos relleno alrededor de la imagen para tener en cuenta los píxeles de los bordes. Los valores así como el tamaño del borde son variables y dependen del tipo de procesamiento que se requiere hacer a la imagen. La siguiente figura ilustra el relleno de una imagen en escala de grises.



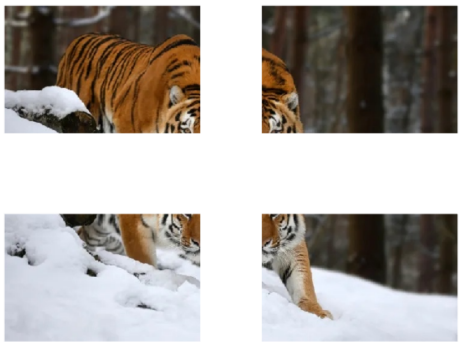
Ejercicio A continuación, se describen los pasos que debes seguir para completar la tarea:

1. Utiliza la matriz $ImageGris \in \mathbb{R}^{m \times n}$ de la Parte 2 como punto de partida.
2. Agrega un relleno de ceros en el borde de la matriz anterior y define una nueva matriz $ImageGrisAu \in \mathbb{R}^{(m+1)\times(n+1)}$.
3. Aplica una operación de convolución a la matriz $ImageGrisAu$ utilizando una matriz filtro $B \in \mathbb{R}^{3\times3}$.

Tu tarea consiste en seguir estos pasos y realizar la convolución de la matriz de imagen $ImageGrisAu$ utilizando la matriz filtro B. Asegúrate de comprender el proceso de convolución y cómo afecta la imagen resultante.

3.4. Parte 4: División de una Imagen en Subimágenes

En el contexto de procesamiento de imágenes, te han asignado la tarea de dividir una imagen en partes más pequeñas para facilitar su análisis y procesamiento. Para lograr esto, necesitas implementar un programa en MATLAB que divida una imagen a color en una cuadrícula de subimágenes de tamaño 2x2.



A continuación, se describen los pasos que debes seguir para completar la tarea:

1. Lee una imagen en formato RGB.
2. Define el número de filas y columnas en la cuadrícula de subimágenes (por ejemplo, `numRows = 2` y `numCols = 2`).
3. Calcula las dimensiones de cada subimagen dividiendo el alto y ancho de la imagen original por el número de filas y columnas.
4. Crea una matriz para almacenar las subimágenes. Utiliza la función `cell` para crear una matriz de celdas.
5. Utiliza bucles anidados para iterar a través de las filas y columnas de la cuadrícula. Para cada subimagen, calcula las coordenadas de inicio y fin en la imagen original y almacena la subimagen en la matriz de celdas.
6. Muestra las subimágenes en una cuadrícula en una nueva figura utilizando la función `subplot` y `imshow`.

Tu tarea consiste en implementar el código en MATLAB siguiendo estos pasos y verificar que la imagen se divida correctamente en subimágenes y se muestre en una cuadrícula.

Puedes utilizar el código que proporcionaste como referencia para llevar a cabo esta tarea.

4. Preguntas adicionales (opcional)

A continuación se presentan los códigos incompletos que permiten aplicar diversos tipos de filtro a nuestra imagen. Se pide completar dichos códigos en función de lo visto en clase. Utilizaremos esta imagen como referencia

2. Difuminado

Para obtener el difuminado deseado utilizaremos el método "desenfoque de caja". Este funciona tomando cada píxel y cambiando su valor por el promedio de sus píxeles vecinos. El nuevo valor de cada píxel sería la media de los valores de todos los píxeles que están dentro de 1 fila y columna del píxel original (formando una caja de 3x3). El valor del propio píxel también debe ser incluido en el cálculo del promedio.

Considere la siguiente representación de bits:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Para el píxel 6, se obtendría el nuevo valor promediando los valores de color originales de los píxeles 1, 2, 3, 5, 6, 7, 9, 10 y 11 (nótese que el propio píxel 6 está incluido en la media).

Asimismo, los valores de color del píxel 11 se obtendrían promediando los valores de color de los píxeles 6, 7, 8, 10, 11, 12, 14, 15 y 16. Para un píxel a lo largo del borde o de la esquina, como el píxel 15, seguiríamos buscando todos los píxeles dentro de 1 fila y columna: en este caso, los píxeles 10, 11, 12, 14, 15 y 16.

Ejercicio 1.
El siguiente código sirve para aplicar un difuminado utilizando un relleno adecuado. Primero aplicaremos un relleno de tamaño 25 y de color aleatorio a la imagen usando el siguiente código.

```
    function border = aplicar_relleno(obj)

[m n o] = size(obj) % para completar
border_width = 25; %para completar

border = zeros(m+(border_width*2),n+(border_width*2),3);

c = [randi(255) randi(255) randi(255)]; %para completar

border(:,:,1) = c(1);
border(:,:,2) = c(2);
border(:,:,3) = c(3);

for i = 1 : m
    for j = 1 : n
        border(i+border_width,j+border_width,:) = (obj(i,j,:));
    end
end
```

En este caso usaremos una matriz cuadrada A de orden 13 para el filtro.

```
    lista_3d = leer_imagen('https://th.bing.com/th/id/R.06ab389aa05724868c09d806abedd52

lista_3d
guardar_imagen('eg_utec.bmp', aplicar_difuminado(lista_3d));

imshow('eg_utec.bmp')

function result = aplicar_difuminado(obj)
```

```
            % SU SOLUCION EMPIEZA AQUI
            A = ones(13,13)/169 % Esto es para completar
            result = imfilter(obj,A)
            % SU SOLUCION TERMINA AQUI
end

function lista_3d = leer_imagen(ruta)
    % La función leer_imagen recibe una cadena con la ruta
    % de una imagen en formato BMP y devuelve una matriz
    % tridimensional con el mapa de bits de la imagen.
    % Convertimos la matriz a un tipo de dato común en lugar de usar la librería numpy.

    img = imread(ruta);
    lista_3d = int32(img);
end

function guardar_imagen(ruta, lista_3d)
    % La función guardar_imagen recibe una matriz tridimensional
    % con el mapa de bits de la imagen y la guarda en formato bmp.

    imwrite(uint8(lista_3d), ruta);
end
```

Ejercicio 2.

Sea $Filter \in \mathbb{R}^{3\times 3}$ y los vectores $v_1 = (1,0,1), v_2 = (1,2,0)$ y $v_3 = (-2,0,1)$. Si se cumple que

$$Filter * v_1' = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \quad Filter * v_2' = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad \text{y} \quad Filter * v_3' = \begin{pmatrix} -3 \\ -3 \\ -3 \end{pmatrix}.$$

Hallar la matriz $Filter$.

Luego aplicar este filtro a nuestra imagen de referencia dos veces. Para ello solo deben modificar la matriz utilizada en el código anterior.