ÁLGEBRA APLICADA

Prof: Maglis Mujica

Autores: Juan Nocetti - 5.390.966-5

Renzo Giacometti - 5.590.459-8

Ionas Josponis - 5.242.903-0

Año 2023

Índice

Introducción

El presente proyecto busca aplicar los conocimientos adquiridos en la unidad 4 del curso sobre Transformaciones Lineales, y valores y vectores propios de una matriz. Enfocaremos estos conceptos en el procesamiento de imágenes, explorando transformaciones lineales y la compresión de imágenes mediante la descomposición SVD.

Objetivos

Transformaciones Lineales:

- Desarrollar una comprensión profunda de las matrices correspondientes a cada transformación lineal, incluyendo rotación, escalado y deformación.

- Implementar funciones en Python que permitan la aplicación eficiente de estas transformaciones a una imagen seleccionada.
- Realizar experimentos con diversos valores de parámetros en cada transformación con el fin de entender y documentar sus efectos visuales.

Compresión de Imágenes:

- Crear un algoritmo en Python para la compresión de imágenes utilizando la descomposición SVD.
- Ejecutar pruebas exhaustivas con diferentes valores de k (al menos 4) en el proceso de compresión de la imagen seleccionada.
- Analizar y comentar los resultados obtenidos, observando cómo varía la calidad de la imagen al modificar el valor de k.

Marco Teórico

Transformaciones lineales:

Describen las relaciones y cambios en espacios vectoriales mediante operaciones matriciales o lineales.
En el contexto de imágenes, se aplican transformaciones como rotación, escalado, deformación, reflexión, y proyección.

Escalado (Scaling): El escalado es aquella operación que aumenta/disminuye el tamaño de los objetos . En un escalado, cada vector v , se escala por un factor k, es decir $T(v)=kv$, para cada $v \in R^2$. Esto permite cambiar el tamaño o la proporción de una imagen.

Rotación: La rotación implica girar una imagen alrededor de un punto o eje específico que representamos con la letra θ . Las matrices de rotación son ortonormales.

Deformación (Shearing): La deformación implica cambiar la forma de una imagen de manera que no necesariamente se conservan las distancias y ángulos entre píxeles, alterando localmente la posición de los mismos. Los píxeles individuales se desplazan a nuevas ubicaciones, lo que cambia la forma general de la imagen. Cuando ejecutamos una deformación, la imagen se puede inclinar tanto en sentido horizontal como en sentido vertical. Lo que se hace es mapear un punto P en un correspondiente punto P' y un ángulo α se forma entre PP' y el eje x.

Matriz de Escalado:

$$S = \begin{| s_x & 0 \\ 0 & s_y \end|}$$

Matriz de Rotación:

$$R = \begin{| \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end|}$$

Matriz de Deformación:

$$D = \begin{| 1 & d_x \\ d_y & 1 \end|}$$

Compresión de imágenes utilizando SVD:

Comprende técnicas para reducir el volumen de datos en imágenes. SVD se puede aplicar a una imagen para descomponerla en sus componentes principales y conservar los componentes que almacenan mayor información.

Entonces, tenemos que una matriz M de tamaño $m \times n$ y de rango r puede descomponerse como el producto de tres matrices de la forma $M=U.S.V^T$, lo que se denomina Descomposición en valores singulares de la matriz M (SVD - Singular Value Decomposition) dónde:

U es una matriz $m \times r$ ortonormal por columnas, es decir, cada una de sus columnas es un vector de norma 1 y el producto punto de dos columnas cualesquiera es 0.Esta matriz representa las relaciones entre filas (píxeles) de la imagen. Cada columna de U se denomina

"vector singular izquierdo" y está relacionada con una característica o patrón particular en la imagen.

S es una matriz diagonal que contiene los valores singulares de M. Los valores singulares son las raíces cuadradas de los valores propios de $Mt*M$. Estos valores le indican qué tan importante es cada vector singular izquierdo y derecho correspondiente para representar la imagen. Los valores singulares están ordenados desde el más importante (arriba a la izquierda) al menos importante (abajo a la derecha).

V es una matriz $r \times n$ ortonormal por filas, es decir, cada una de sus filas es un vector de norma 1 y el producto punto de dos filas cualesquiera es 0. Esta matriz representa las relaciones entre columnas (píxeles) de la imagen. Cada columna de VT se denomina "vector singular derecho" y está relacionada con una característica o patrón particular en la imagen.

Una imagen contiene información redundante, es decir, que puede ser eliminada sin que el efecto visual sea notable, y se podría sustituir M por otra matriz $M1$ de menor rango. Para esto debemos descomponer la matriz de imagen y luego comprimir la información tomando solo cierta cantidad k, con $1<k<r$, de valores singulares.

Herramientas

- IDE: Es un entorno de desarrollo para poder programar, en nuestra aplicación usamos PyCharm de JetBrains y Visual Studio Code.

- **Python:** Lenguaje de desarrollo.

-GitHub: Es una herramienta para desarrollar software colectivamente.

-Numpy: Librería de Python que nos facilita algunas operaciones entre matrices.

-CV2: Librería para trabajar con imágenes en Python.

Desarrollo

Para desarrollar este programa creamos dos clases principales, la clase utils y main.

Dentro de la clase utils se encuentra toda la lógica para poder realizar las diferentes partes del proyecto. Mientras tanto, la clase main conecta la interfaz gráfica con la lógica del programa.

El programa funciona en base a las siguientes funciones:

def show_image(image, title):

Esta función muestra una matriz de píxeles como una imagen con la librería matplotlib.

También muestra un título para dar contexto de lo que se trata la imagen.

```
def show_image(image, tittle):
    plt.imshow(image,"gray")
    plt.title(tittle)
    plt.show()
```

def compress_image(image, k):

Esta función toma una imagen y reduce su complejidad al retener sólo las k características más significativas. Utiliza la descomposición de valores singulares (SVD) para descomponer la matriz de píxeles de la imagen en tres matrices U, S y V.
Luego, conserva solo las primeras k columnas de estas matrices para obtener una aproximación comprimida de la imagen original. El resultado es una versión comprimida de la imagen con una pérdida controlada de información.

```
def compress_image(image, k):
    image = rgb2gray(image)
    imageToArray = np.array(image)
    U, S, V = np.linalg.svd(imageToArray, full_matrices=False)
    compressed_image = np.dot(U[:, :k], np.dot(np.diag(S[:k]), V[:k, :]))
    return compressed_image
```

rotate_image(image, angle):

La función rotate_image utiliza una imagen y un ángulo como parámetros de entrada para realizar una rotación bidimensional. La imagen otorgada se gira dependiendo del ángulo pasado por parámetro. La salida de la función es la imagen rotada.

```
def rotate_image(image, angle):
    image = np.array(image)
    rotation_rad = angle * np.pi / 180.0

    height, width, channels = image.shape

    max_len = int(math.sqrt(height*height + width*width))
    rotated_image = np.zeros((max_len, max_len, channels))

    rotated_height, rotated_width, _ = rotated_image.shape
    mid_row = int( (rotated_height+1)/2 )
    mid_col = int( (rotated_width+1)/2 )

    for r in range(rotated_height):
        for c in range(rotated_width):
            y = (r-mid_col)*math.cos(rotation_rad) + (c-mid_row)*math.sin(rotation_rad)
            x = -(r-mid_col)*math.sin(rotation_rad) + (c-mid_row)*math.cos(rotation_rad)

            y += mid_col
            x += mid_row

            x = round(x)
            y = round(y)
            if (x >= 0 and y >= 0 and x < width and y < height):
                rotated_image[r][c][:] = image[y][x][:]
    result = Image.fromarray(rotated_image.astype("uint8"))

    return result
```

scaling(image,k):

La función scaling acepta una imagen y un factor de escala k como parámetros de entrada, realizando un redimensionamiento proporcional de la imagen. El valor de k determina la escala del cambio en las dimensiones de la imagen, donde un k mayor que 1 amplía la imagen, mientras que un k entre 0 y 1 la reduce. La salida de la función es la imagen modificada.

```python
def scaling(image, k):
    scaled_image = image.resize(
        (int(image.width * k), int(image.height * k)))
    return scaled_image
```

shearing(image, shearInput):

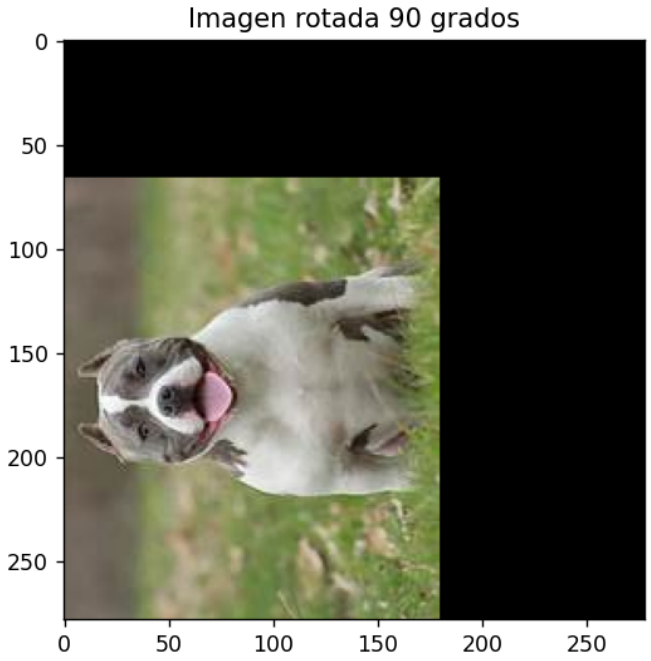
La función shearing utiliza una imagen y un valor numérico shearInput como parámetros de entrada para aplicar un sesgo a la imagen. El sesgo deformará la imagen en una dirección específica según el valor proporcionado. La salida de la función es la imagen transformada después de la distorsión.\

```python
def shearing(image, shearInput):
    image = np.array(image)
    afine_tf = tf.AffineTransform(shear=shearInput)
    modified = tf.warp(image, inverse_map=afine_tf)
    return modified
```

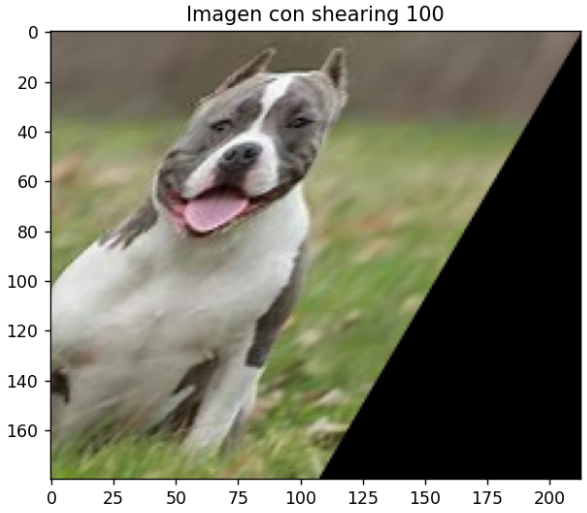
Respuestas a interrogantes planteadas:

1. *Explicar qué observa en cada transformación.*

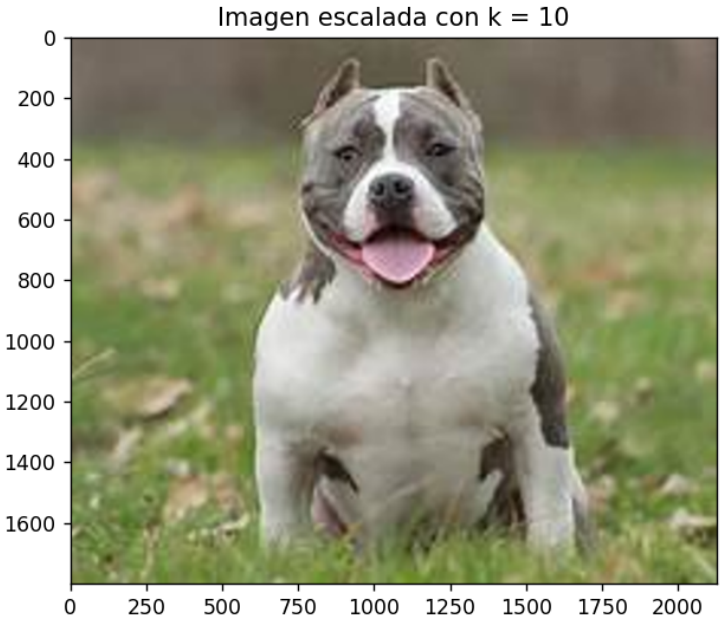
Se puede observar, en el caso de la rotación, como la imagen queda girada:



En el caso del shearing, la imagen se deforma en el plano horizontal, haciendo que quede torcida:

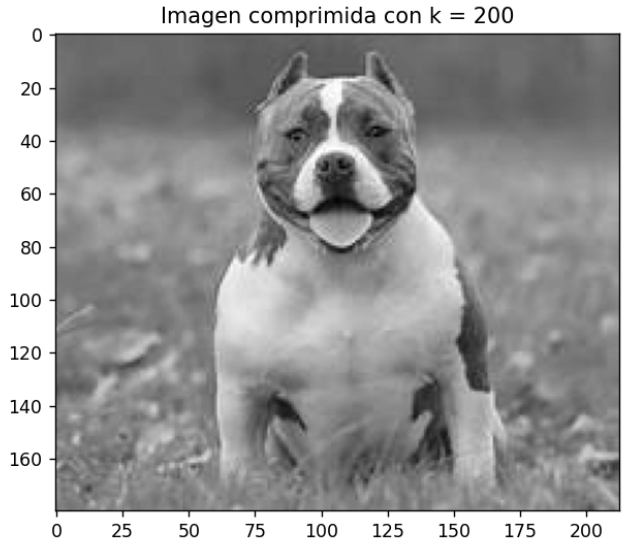


Por último, el caso del escalado, se puede observar el cambio en el tamaño al mirar los ejes que indican la cantidad de pixeles:



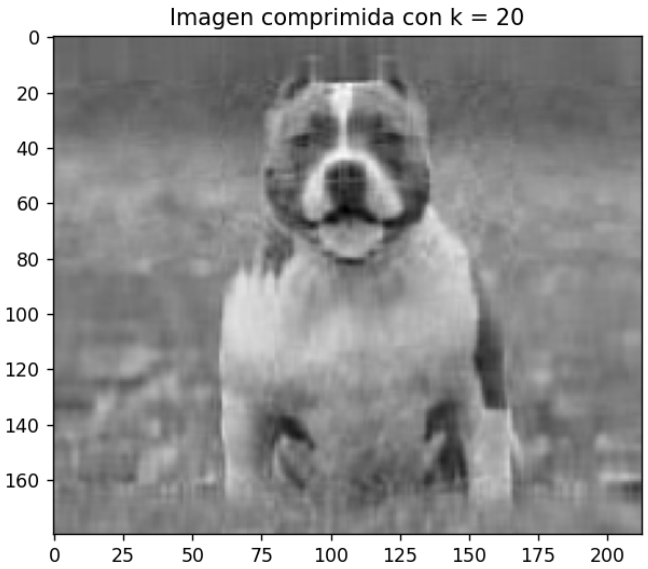
2. *Probar con distintos valores de k (al menos 4) y mostrar los resultados de las imágenes comprimidas. Comentar los resultados.*

Aquí vemos una imagen comprimida con k = 200:



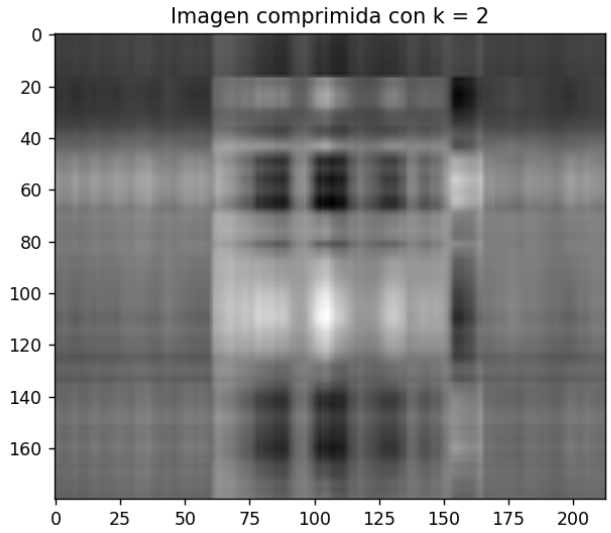
En este caso, la imagen pierde un nivel notable de calidad, pero se puede seguir apreciando de forma clara la imagen original.

Luego probamos con k = 20:



En este caso, la pérdida de calidad es mucho más notoria, y se puede observar como, aunque reconocible, la imagen comienza a quedar mucho más borrosa y difuminada.

Por último, y solo por hacer la prueba, quisimos utilizar k = 2:



En este caso, y como se había explicado anteriormente en clase, la imagen queda completamente irreconocible, ya que utilizar dos tonos de muestreo para recrear toda la imagen se torna inviable a la hora de querer obtener cualquier tipo de detalle o calidad.

3. *¿Qué ocurre al aumentar el valor de k? ¿Por qué sucede eso? ¿Qué representa el valor k en el contexto de la Descomposición de Valor Singular (SVD)?*

¿Qué ocurre al aumentar el valor de k?

Aumentar el valor de k tiene un impacto directo en la calidad de la imagen comprimida. A medida que k aumenta, se retienen más componentes principales de la imagen original durante el proceso de compresión. En términos prácticos, esto significa que la imagen comprimida conserva más detalles finos y sutilezas presentes en la imagen original.

¿Por qué sucede eso?

La descomposición SVD descompone una matriz en tres componentes principales:

U, S, y V T. La matriz S contiene los valores singulares, ordenados de mayor a menor. Al seleccionar los primeros k valores singulares más grandes y sus correspondientes vectores singulares izquierdos y derechos de las matrices U y V T, respectivamente, estamos capturando las componentes más importantes de la imagen original. Por lo tanto, al aumentar k, estamos reteniendo más información relevante y preservando una representación más fiel de la imagen.

¿Qué representa el valor k en el contexto de la Descomposición de Valor Singular (SVD)?

En el contexto de la Descomposición de Valor Singular, el valor k representa la cantidad de componentes principales que se retienen durante la compresión de la imagen. Cada componente principal captura una porción de la información total de la imagen. Por lo tanto, al elegir un valor k más alto, estamos incluyendo más componentes principales y, por ende, conservando más detalles de la imagen original. Sin embargo, esto también significa que el tamaño del archivo comprimido será mayor, ya que se retiene más información.

4. *¿Por qué efectivamente el procedimiento realizado resulta en una compresión de la imagen?*

La descomposición SVD permite identificar y separar la información redundante de la imagen. Los valores singulares más pequeños en la matriz S representan componentes de

menor importancia en la imagen, y al truncar estos valores (utilizando un valor k menor), eliminamos información redundante que contribuye menos a la representación visual haciendo que tenga menor cantidad de información y consecuentemente ocupe menos espacio.

Conclusiones

El proyecto resalta la importancia de los conceptos de álgebra lineal en el procesamiento de imágenes, ofreciendo herramientas poderosas para manipular y representar datos visuales.
La combinación de transformaciones lineales y compresión de imágenes abre posibilidades significativas en áreas como la corrección de imágenes, mejora visual y eficiencia en el almacenamiento de datos visuales.

Pasos a futuro

- Se podría implementar una interfaz gráfica que permita ingresar al usuario por su cuenta los distintos valores de k mostrando su imagen resultante al aplicar la función seleccionada.
- Crear algoritmos de scaling y shearing utilizando solo operaciones matriciales.

Bibliografía

-Presentaciones de clase de Google Colabs.