reporte_PCA

December 2, 2021

1 Proyecto Final: Cálculo de PCA aplicado a compresión de imágenes.

Carlos Bautista*, Edgar Bazo*, Luz Hernández* e Ita Santiago*

* Alumnos de la materia ANCC

1.1 Introducción

Como se vio en clase de optimización, el cómputo matemático ha ayudado al desarrollo tecnológico e informático con un acelerado crecimiento, lo cual ha incrementado la cantidad de información (datos) a computar. Podemos asumir que este crecimiento es exponencial y la forma en la que se trata hoy, con respecto a hace algunos años, no es la misma, lo cual ha impulsado a la investigación y aplicación de matemáticas para optimizar los problemas que esto ha traído. En este contexto, tenemos muy claro que independientemente del avance tecnológico los recursos computacionales (hardware, software) son finitos y los costos de implementación y mantenimiento pueden ser elevados.

En este proyecto, trabajaremos en el análisis de componentes principales (PCA) para datos de imágenes. PCA es una famosa técnica de reducción de dimensionalidad no supervisada que viene a nuestro rescate cada vez que la maldición de la dimensionalidad nos persigue. Aunado a esto, vamos a calcular las componentes principales haciendo uso del algoritmo de rotaciones de Jacobi para obtener los SVD y a partir de estos los PCA.

Uno de los casos de uso de PCA es que se puede utilizar para la compresión de imágenes, una técnica que minimiza el tamaño en bytes de una imagen manteniendo la mayor calidad de imagen posible. Una imagen de color típica se compone de píxeles, muchos píxeles se unen en una matriz para formar una imagen digital. Dicha imágen digital típica se crea apilando matrices de píxeles rojo, azul y verde de intensidades que van de 0 a 255.

1.2 Teoría

1.2.1 Repaso de PCA

PCA es uno de los métodos más utilizados para encontrar patrones en los datos y es usado frecuentemente cuando cada observación contiene muchas características y no todas ellas son significativas, pero también se usa cuando existe mucha covarianza entre las características. En pocas palabras, describe los datos resumiéndolos en patrones típicos llamados componentes principales, en donde estos nos permitirán explicar los valores a través de una combinación de ellos.

Las componentes principales encuentran una proyección lineal de datos en un sistema de base ortogonal que tiene la redundancia mínima y conserva la variación en los datos. Aplicaciones: * Identificar la dimensionalidad intrínseca de los datos. * Representación dimensional más baja de datos con el menor error de reconstrucción.

1.2.2 PCA para reducción de dimensionalidad

- Si los datos viven en un espacio dimensional inferior, entonces algunos de los valores propios en la matriz se establecen en 0.
- Si queremos reducir la dimensionalidad de los datos de d a algunos fijos k elegimos los eigenvectores correspondientes a los k eigenvalor más altos, i.e. las dimensiones que conservan la mayor parte de la varianza en los datos.
- Esta selección también minimiza el error de reconstrucción de datos (por lo que las mejores k dimensiones conducen al mejor error).

1.2.3 Algoritmo PCA

Pasos para realizar el PCA: de una matriz $X \in \mathbb{R}_{N\times a}$ a una matriz $X \in \mathbb{R}_{N\times b}$:

1. Normalización: Centrar la matriz, substrayendo la media.
2. Calcular la matriz de covarianzas de la matriz centrada $C = \frac{1}{N-1}X^T X$.
3. Calcular los eigenvectores de la matriz de covarianzas.
4. Seleccionar los m eigenvectores correspondientes a los eigenvalores más grandes.

Obtenemos los **Componentes Principales de X** al multiplicar estos eigenvectores por $X^T_{centrada}$.

El proceso de PCA identifica aquellas direcciones en las que la varianza es mayor. Como la varianza de una variable se mide en su misma escala elevada al cuadrado, si antes de calcular las componentes no se estandarizan todas las variables para que tengan media 0 y desviación estándar 1, aquellas variables cuya escala sea mayor dominarán al resto. De ahí que sea recomendable estandarizar siempre los datos.

1.2.4 Obtención de valores y vectores propios

El número σ se denomina valor singular de A si $\sigma = \sqrt{\lambda A^T A} = \sqrt{\lambda A A^T}$ donde: $\lambda A^T A$ y $\lambda A A^T$ es eigenvalor de $A^T A$ y AA^T respectivamente.

1.2.5 Uso de rotaciones de Jacobi

Este método produce una secuencia de transformaciones ortogonales de la forma $J_k^T A J_k$ con el objetivo de hacer "más diagonal" a la matriz $A \in \mathbb{R}_{n\times n}$.

Si la matriz A es simétrica y J_0 es una transformación de rotación de Jacobi, ver transformaciones de rotación, entonces el esquema iterativo:

$$A_{k+1} = (J_0 J_1 \dots J_k)^T A (J_0 J_1 \dots J_k)$$

converge a una matriz diagonal en la que se encuentran los eigenvalores de A.

1.2.6 Algoritmo rotaciones de Jacobi

Dados A simétrica y $tol > 0$ definir $A_0 = A$, $Q_0 = I_n$.

Repetir el siguiente bloque para $k = 0, 1, 2, \ldots$

1. Elegir un par de índices $(idx1, idx2)$ según las metodologías vistas en clase,
2. Calcular las entradas $cos(\theta), sin(\theta)$ de la matriz de rotación J_k,
3. $A_{k+1} = J_k^T A_k J_k$,
4. $Q_{k+1} = Q_k J_k$,

hasta convergencia: satisfacer criterio de paro en el que se utiliza tol y $maxsweeps$.

La matriz J_k se utiliza para eliminar un par de entradas (simétricas) en la matriz A_k, esto preserva la simetría de la matriz original. En las columnas de la matriz Q_k se encuentran aproximaciones a los eigenvectores de A y en la diagonal de A_k se tienen aproximaciones a los eigenvalores de A.

Una vez analizado todo lo anterior, crearemos las funciones necesarias para llevar a cabo la compresión de imágenes.

Haremos el cálculo de **eigenvalores** y **eigenvectores** de nuestra matriz X de datos utilizando **método de rotaciones de Jacobi y ordenamiento cíclico por renglones.**

Se toman las funciones definidas en la nota 2.3 Algoritmos y aplicaciones de eigenvalores y eigenvectores de una matriz del libro de Optimización.

Definimos las funciones para "diagonalizar" la matriz $X^T X$ con rotaciones de Jacobi a través del método de ordenamiento cíclico por renglones.

Primero, definimos funciones auxiliares `off()` y `max_sweeps()` para el criterio de paro

Definimos función que realiza las iteraciones de Jacobi:

Ahora realizaremos el cálculo con una función que implementamos y que nos ayuda a calcular las PCA, basándonos en el método de Jacobi

Recordamos que las "direcciones principales están dadas por las columnas de V (salvo signos positivos o negativos)", y que las "**componentes principales están dadas por la multiplicación matricial de** XV (salvo signos positivos o negativos)"

Siguiendo un método más directo y que también funciona, con base en nota 2.4 del libro de Optimización. Podemos obtener PCA multiplicando $X_{centrada}$ por V (obtenida a partir de Q_k resultante de aplicar Jacobi a $X^T X$).

1.3 PCA para compresión de imágenes

Como vimos, el análisis de componentes principales, o PCA, es una técnica estadística para convertir datos de alta dimensión en datos de baja dimensión, mediante la selección de las características más importantes que capturan la máxima información (varianza) sobre el conjunto de datos [6].

Basamos la idea de nuestras funciones en las características que se seleccionan sobre la base de la varianza que causan en la salida. Como también aprendimos en clase, la característica que causa la mayor variación es el primer componente principal. La característica que es responsable de la

segunda varianza más alta se considera el segundo componente principal, y así sucesivamente. Es importante mencionar que los componentes principales no tienen ninguna correlación entre sí.

Aparte de las múltiples aplicaciones de PCA, otra aplicación interesante es la compresión de imágenes, nuestra motivación para usar esta herramienta para eso, se explica en la siguiente sección.

Echemos un vistazo a cómo podemos lograr esto con Python.

1.4 Motivación

Como se mencionó en la introducción, una de las aplicaciones generales más importantes del PCA es la reducción de dimensionalidad. Ésta a su vez es de múltiple utilidad en diversos contextos de la ciencia de datos que van desde la hasta la compresión de imágenes.

En el caso de este presente trabajo nos interesa probar el funcionamiento de algoritmos utilizados en el curso para reducir dimensionalidad a través de PCA, aplicados a la compresión de imágenes. Se busca lograr los resultados consistentes con lo obtenido con las paqueterías de python comúmnente utilizadas para resolver estos problemas (linalg y scikit-learn).

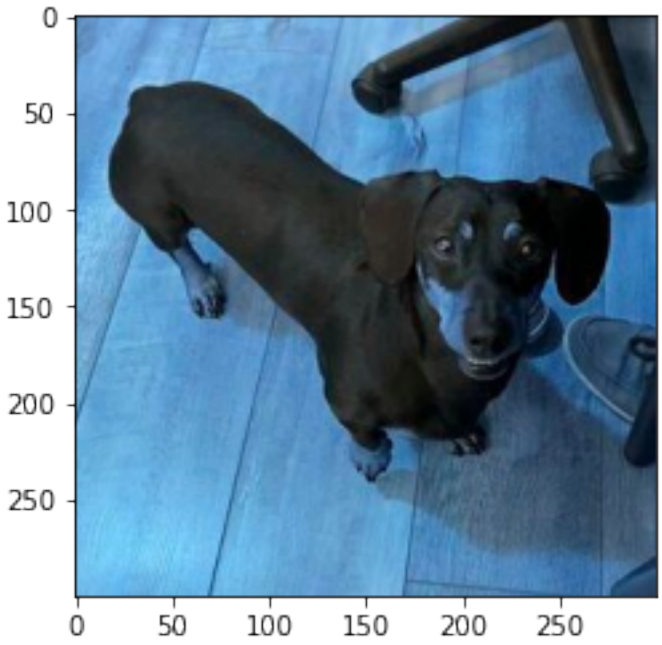
En la práctica del DeepLearning, es común utilizar imágenes para entrenar modelos de redes neuronales, para reconocimiento de imágenes, entre otros. Para ello se utilizan una gran cantidad de imagénes voluminosas para entrenar los modelos, lo cual afecta los tiempos de procesamiento.

En un contexto de recursos limitados, resulta de gran utilidad procesar las imágenes para comprimirlas (reducir la dimensionalidad) y lograr eficiencia en los procesos sin afectar la calidad de las imágenes, obteniendo así resultados muy similares con mayor eficiencia.

1.4.1 Cargando la imagen

Vamos a usar una foto de una de nuestras mascotas.

```
<matplotlib.image.AxesImage at 0x7fd8c42eb670>
```

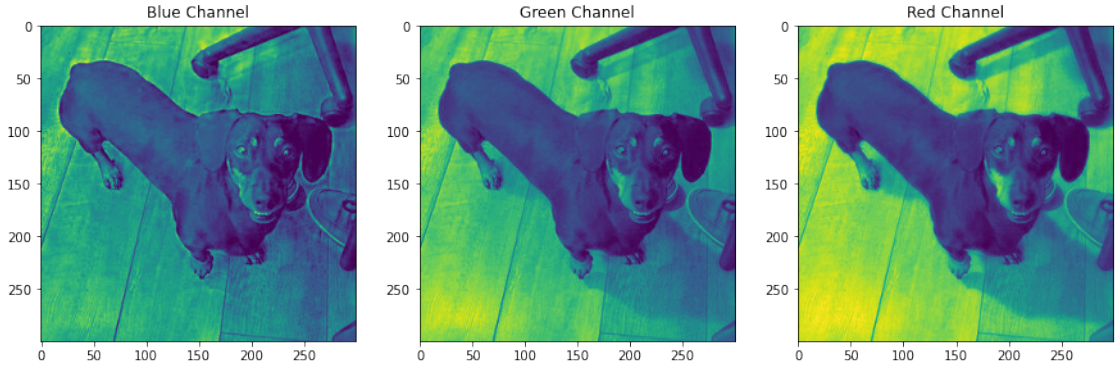


Veamos el tamaño de la imagen

```
(300, 300, 3)
```

1.4.2 División de la imagen en formato RGB

Sabemos que una imagen digital en color es una combinación de matrices R, G y B (rojo, verde, azul) apiladas unas sobre otras. Es necesario dividir cada canal de la imagen y extraer los componentes principales de cada uno de ellos.



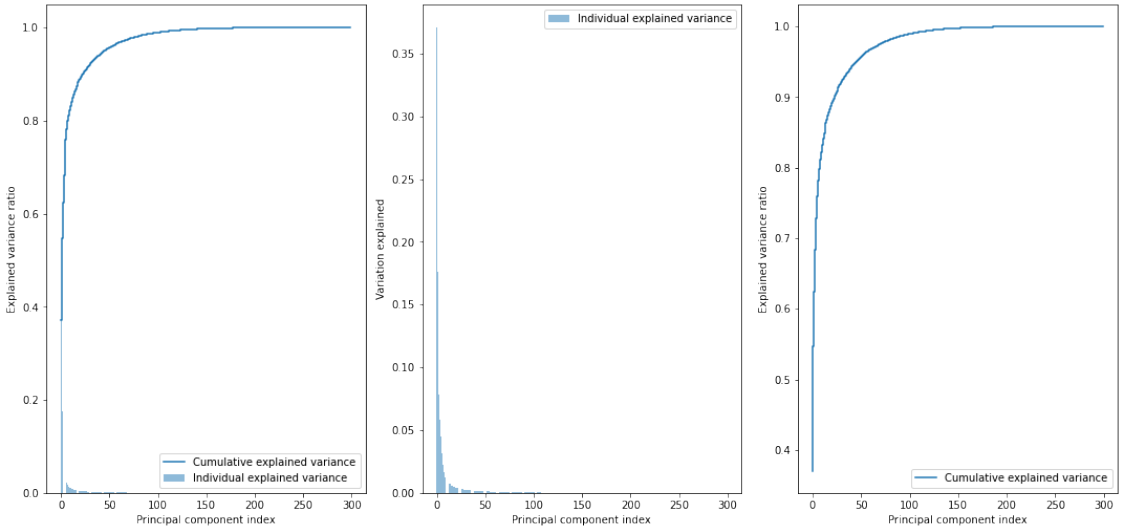
Escalamos los datos

1.4.3 PCA con Jacobi

```
Sweeps 12
------------------------
Off(x_k) = 1.1127400598700876e-08
tolerancia = 1.5977985427208198e-08


La matriz q_k queda como:
[[ 0.    -0.014 -0.028 … -0.08  -0.08  -0.009]
 [-0.001 -0.015 -0.02  …  0.02   0.07  -0.047]
 [-0.001 -0.016 -0.02  … -0.05   0.016  0.096]
 …
 [ 0.029  0.015 -0.114 …  0.129 -0.179  0.026]
 [ 0.022  0.024 -0.111 … -0.     0.149 -0.019]
 [ 0.016  0.029 -0.1   … -0.024 -0.081  0.049]]
------------------------
```

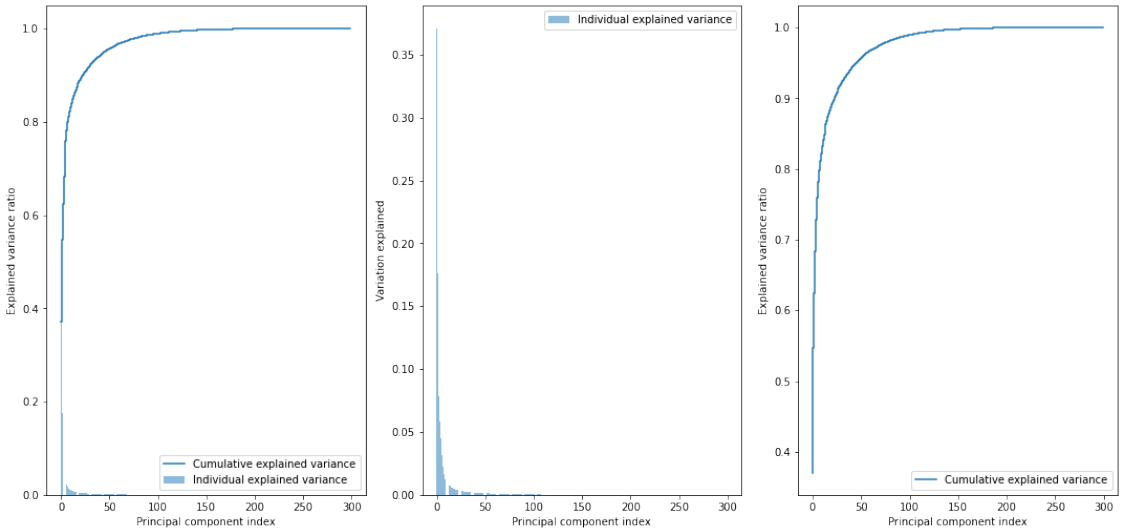
```
Sweeps 12
------------------------
Off(x_k) = 3.545663803727385e-08
tolerancia = 5.7644703209919294e-08


La matriz q_k queda como:
[[ 0.004 -0.012  0.029 …  0.032  0.016 -0.056]
 [ 0.003 -0.012  0.023 … -0.     0.003  0.053]
 [ 0.003 -0.012  0.025 …  0.006 -0.036  0.033]
 …
 [-0.012 -0.001  0.112 … -0.042 -0.153  0.044]
 [-0.016  0.004  0.111 …  0.101  0.154 -0.009]
 [-0.019  0.007  0.106 … -0.056 -0.073 -0.008]]
------------------------


Sweeps 12
------------------------
Off(x_k) = 4.062419189351485e-08
tolerancia = 1.1282751158192279e-07


La matriz q_k queda como:
[[ 0.002  0.002  0.019 …  0.047 -0.006 -0.047]
 [ 0.001  0.002  0.015 … -0.021 -0.048 -0.036]
 [ 0.001  0.002  0.016 … -0.016  0.021  0.   ]
 …
 [-0.026  0.004  0.106 …  0.055 -0.03  -0.15 ]
 [-0.029 -0.001  0.105 … -0.012  0.088  0.128]
 [-0.031 -0.003  0.102 … -0.056 -0.041  0.005]]
------------------------
```

1.4.4 Varianza explicada

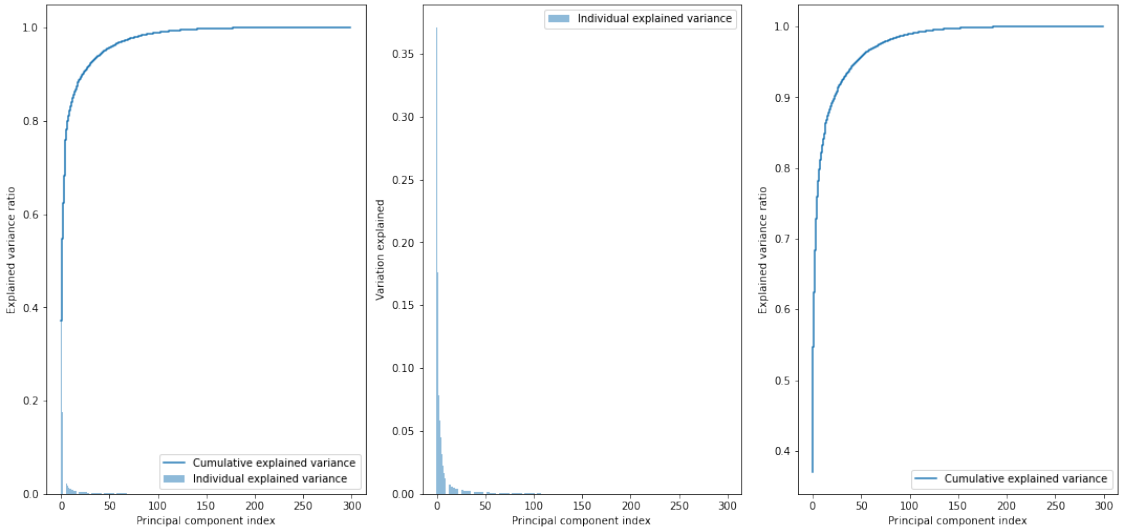


Podemos ver que con 50 componentes principales ya se explica más del 95% y con 72 componentes (de 300) se explica el 99%.

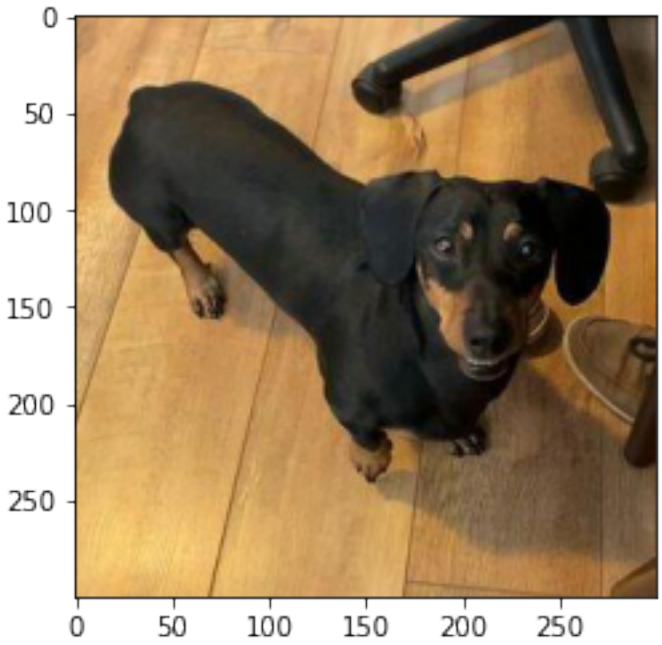
Canal verde

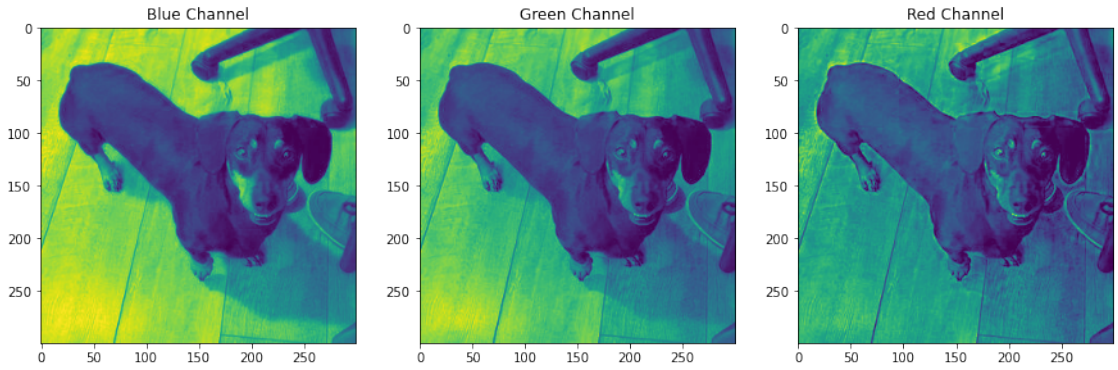


Canal rojo

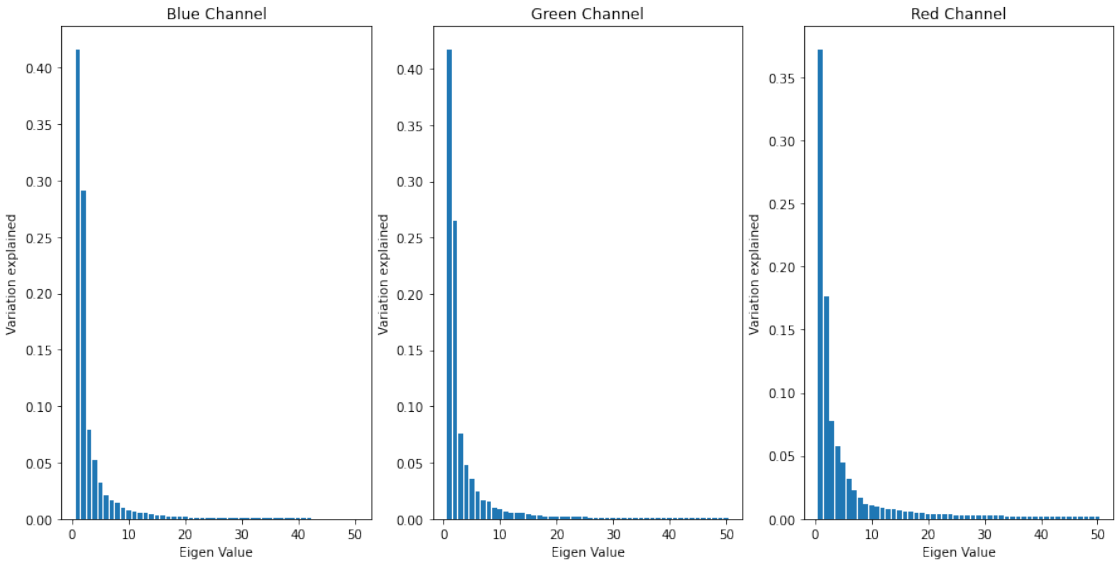


En el caso de los tres canales tenemos un desempeño similar, con menos de la cuarta parte de los datos.






```
Blue Channel : 0.991735312788283
Green Channel: 0.9846153727870853
Red Channel  : 0.955207320440407
```




```
array([[0.425, 0.401, 0.383, …, 0.304, 0.3  , 0.271],
       [0.426, 0.386, 0.361, …, 0.289, 0.297, 0.267],
       [0.409, 0.379, 0.357, …, 0.296, 0.303, 0.271],
       …,
       [0.461, 0.453, 0.441, …, 0.166, 0.162, 0.158],
       [0.452, 0.445, 0.418, …, 0.154, 0.16 , 0.151],
       [0.454, 0.434, 0.404, …, 0.142, 0.161, 0.146]])
```

1.4.5 Recontrucción de la imagen

Ahora que tenemos una reducción de dimensionalidad usando nuestra función PCA, sabemos que tenemos un más del 95% de varianza explicada, ahora visualizaremos la imagen nuevamente y para eso, primero tenemos que invertir la transformación de los datos y luego fusionar los datos de los 3 canales en uno.

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers).
```

Como nuestro ojo humano puede apreciar, la diferencia entre las dos imágenes es mínima, sin embargo, hemos usado la sexta parte de la información.

1.5 Conclusiones

Como se vio durante el curso y que fue uno de los propósitos a demostrar en este proyecto, el tener demasiadas características de una observación se traduce en demasiadas dimensiones que definen a un objeto o registro y este, a su vez, genera un costo computacional muy alto al querer transformar, manipular o computar datos para efectos de cualquier estudio o análisis. Sim embargo, la compresión de imágenes con análisis de componentes principales es una aplicación útil y relativamente sencilla de implementar gracias a que podemos ver a las imágenes como matrices hechas de valores del color de los píxeles. Además, tuvimos la oportunidad de extender esta implementación usando las rotaciones de Jacobi para crear nuestra propia función para obtener las PCA y a partir de estas, poder reducir la dimensión de una imagen en color dividiéndola en 3 canales y luego reconstruyéndola para su visualización.

La reducción de la dimensionalidad actualmente es aplicada en distintos campos, por ejemplo: procesamiento de señales de video como remasterización, procesamiento de señales de audio como reducción de ruido, reconocimiento de voz como en las IA Alexa, Siri o Cortana, visualización de grandes volúmenes de datos (Big Data), en modelado de datos para predicción, Machine learning, compresión de imágenes con la menor pérdida de calidad como lo expuesto en este proyecto, etc.

En estos casos es posible que nos enfrentaríamos a problemas con la convergencia de los algoritmos que desarrollamos para este trabajo, por lo que esperamos en un futuro mejorar nuestras implementaciones para poder sobrellevar este tipo de retos.

Vimos también que, debido a que los recursos computacionales son finitos, es imperativo optimizar los problemas que ayudan a reducir características mediante la reducción de dimensionalidad. Esto tiene importantes beneficios como optimizar el tiempo y costo computacional en el entrenamiento de modelos; transformar datos no lineales en una forma linealmente separables; facilidad para visualizar, analizar y entender los datos; eliminación de características redundantes en los datos; etc. Nosotros pensamos que, en específico, uno de los grandes beneficios es el tema de la reducción en el espacio de almacenamiento de los datos que pueden generar altos costos, monetarios y ambientales, de almacenaje y manutención de la infraestructura.

1.6 Bilbiografía

- [1] Palacios E. (2021) Libro de Optimización
- [2] Golub, G.H., and Van Loan, C.F. (1989) Matrix Computations, 3er ed. (Johns Hopkins University Press).
- [3] Alter O, Brown PO, Botstein D. (2000) Singular value decomposition for genome-wide expression data processing and modeling
- [4] Amat J. (2017) Análisis de Componentes Principales
- [5] Batal I., Strobl E., Hauskrecht M. (2014) Principal Component Analysis (PCA)and Singular Value Decomposition (SVD)
- [6] Hurtado R. (2021) Compresión de Imágenes Mediante Rediucción de Dimensionalidad con Técnica de Análisis de Componentes Principales (PCA)
- [7] Fu D., Guimaraes G. (2016) Using Compression to Speed Up Image Classification in Artificial Neural Networks

1.7 Referencias código

- [1] How to Calculate Principal Component Analysis (PCA) from Scratch in Python
- [2] numpy.column_stack
- [3] numpy.triu_indices
- [4] Dimensionality reduction of color image using PCA
- [5] How to reverse PCA and reconstruct original variables from several principal components?
- [5] On the Applications of Robust PCA in Image and Video Processing
- [6] Application of Principal Component Analysis to Image Compression
- [7] Principal Component Analysis in Image Processing