Utilization of linear algebra in image compression through PCA

Student code: jxl547

Words: 3983

The Beginning:

A PhD student of biology was invited to present his research about fungi growth one lunch period in grade 11. That was my first exposure to the concept of eigenvectors in which he used to analyze his data. Couple month later, while playing video games, a sudden question intrigued me: How does the flat computer screen project something that looked 3D? This question led me into the world of vectors and matrices in which computers used to manipulate the images from 2D to 3D. I was absolutely fascinated by linear algebra, and remembering eigenvectors from before, I was interested in the application of linear algebra and specifically eigenvectors. Leading me to the topic of PCA, specifically how can computers compress images using PCA?

Introduction:

In this paper, example calculations and explanations will be done using small matrices composed to artificial data for the sake of simplicity, while actual calculations with relevant values will be computed using a computer due to the large amount of data for PCA. However, methods described in the paper applies to all matrices for all dimensions, therefore is applicable in the actual performance of PCA. Before performing PCA, some background information is needed.

Vectors and Matrices:

In mathematics, a vector is a quantity with both magnitude(length) and direction.[1] They can be notated as a column vector or a row vector of any size:

$$\vec{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} or\ \vec{v} = [v_1 \quad \dots \quad v_n]$$

Vectors can also be notated in lowercase bold letters:

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

geometrically, they are represented by an arrow in the plane(s) that they belong in.[2] The definition of matrix is a rectangular array of numbers arranged in rows and columns.[3] They can also be thought of as a collection of vectors.

$$\vec{v}\ and\ \vec{w} = \begin{bmatrix} v_1 & w_1 \\ v_2 & w_2 \end{bmatrix}$$

Similarly, matrix *A* that has m rows and n columns is represented as such:

[1] Alain Malouin, *Introduction To Vectors*, trans. Claudia de Fulviis (Québec: P.P.I. inc., 2008) 1.2.
[2] Malouin, *Introduction To Vectors*, 1.3.
[3] "Introduction to Matrices," Lumen, Accessed January 6, 2022, https://courses.lumenlearning.com/boundless-algebra/chapter/introduction-to-matrices/.

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

addition, subtraction, and multiplication are the three operations that can be performed on matrices.

Addition:

Addition of two vectors is the addition of the vectors' coordinates[4]

$$\vec{a} + \vec{b} = \begin{bmatrix} a_1 + b_2 \\ a_2 + b_2 \end{bmatrix}$$

Geometrically,

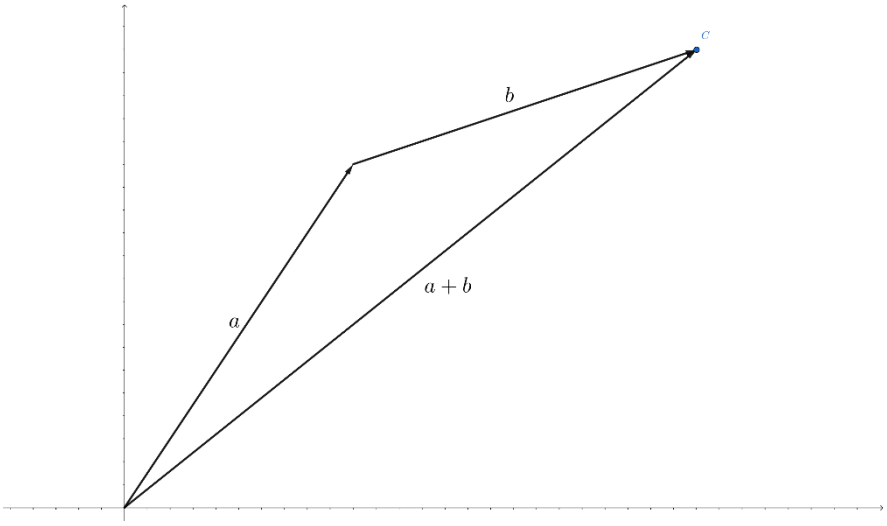

Figure 1 vector addition drawn using Geogebra

Subtraction:[5]

$$\vec{a} - \vec{b} = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \end{bmatrix}$$

Magnitude:

The magnitude, the length, of the vector is derived from the Pythagorean theorem: [6]

$$||\vec{v}|| = \sqrt{{a_1}^2 + {a_2}^2 \ldots + {a_n}^2}$$

Example:

$$\vec{v} = [3 \quad 5]$$

$$||\vec{v}|| = \sqrt{3^2 + 5^2} = \sqrt{19}$$

[4] Stephen Boyd and Lieven Vandenberghe, *Introduction to Applied Linear Algebra* (Cambridge: Cambridge University Press, 2018), 11.
[5] Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 11.
[6] Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 45.

Multiplication:

Given A_{ij} and B_{mn}, the two matrices can multiply only if $j = m$. The resulting matrix will have a dimension of $i \times n$.[7]

$$A = [a_1 \quad a_2]$$

$$B = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$AB = [a_1 \times b_1 + a_2 \times b_2]$$

However, matrix multiplication is that commutative meaning $AB \neq BA$

Proof:

Given matrix A_{2x2} and B_{2x2}

$$AB = \begin{bmatrix} a_{11} \times b_{11} + a_{12} \times b_{21} & a_{11} \times b_{12} + a_{12} \times b_{22} \\ a_{21} \times b_{11} + a_{22} \times b_{21} & a_{21} \times b_{12} + a_{22} \times b_{22} \end{bmatrix}$$

$$BA = \begin{bmatrix} b_{11} \times a_{11} + b_{12} \times a_{21} & b_{11} \times a_{12} + b_{12} \times a_{22} \\ b_{21} \times a_{11} + b_{22} \times a_{21} & b_{21} \times a_{12} + b_{22} \times a_{22} \end{bmatrix}$$

$$\therefore AB \neq BA$$

Vector multiplication is a special case since $V_{1n} = V_{n1}$, therefore it is commutative.

Dot product:

Geometrically, the dot product finds how much a vector points in the direction of another vector.

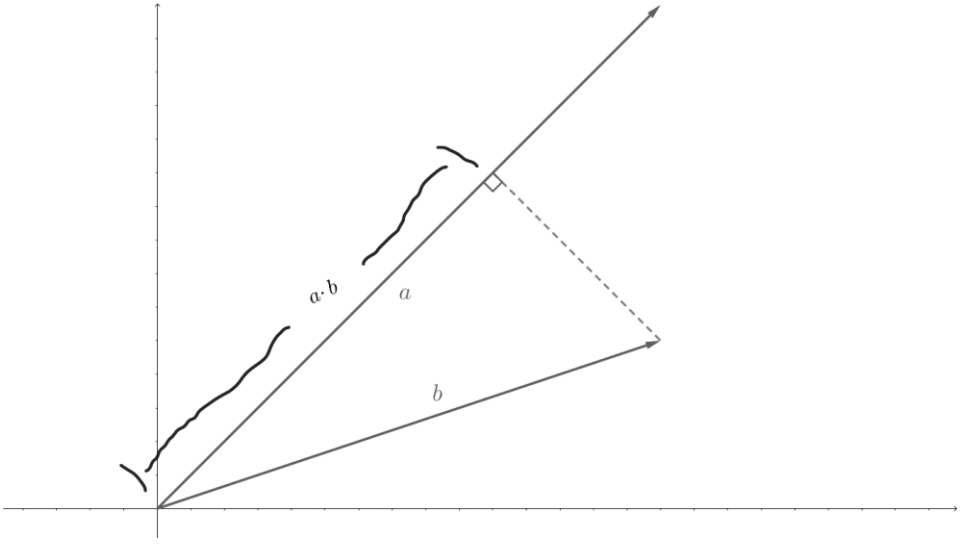

Figure 2 Geometric interpretation of dot product drawn using GeoGebra

[7] Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 177.

Algebraically, they are computed as:[8]

$$\boldsymbol{a} \cdot \boldsymbol{b} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = a_1 b_1 + a_2 b_2$$

with a trigonometry definition of:[9]

$$\boldsymbol{a} \cdot \boldsymbol{b} = \|\boldsymbol{a}\| \|\boldsymbol{b}\| \cos\theta$$

Proof using the cosine law:[10]

$$\begin{aligned} \|\boldsymbol{a} - \boldsymbol{b}\|^2 &= \|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2 - 2\|\boldsymbol{a}\|\|\boldsymbol{b}\| cos\theta \\ (a - b) \cdot (a - b) &= \|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2 - 2\|\boldsymbol{a}\|\|\boldsymbol{b}\| cos\theta \\ \boldsymbol{a} \cdot \boldsymbol{a} - \boldsymbol{2}(\boldsymbol{a} \cdot \boldsymbol{b}) + \boldsymbol{b} \cdot \boldsymbol{b} &= \|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2 - 2\|\boldsymbol{a}\|\|\boldsymbol{b}\| cos\theta \end{aligned}$$

$$\begin{aligned} \because \boldsymbol{a} \cdot \boldsymbol{a} &= \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = {a_1}^2 + {a_2}^2 = \|\boldsymbol{a}\|^2 \\ \|\boldsymbol{a}\|^2 - 2(\boldsymbol{a} \cdot \boldsymbol{b}) + \|\boldsymbol{b}\|^2 &= \|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2 - 2\|\boldsymbol{a}\|\|\boldsymbol{b}\| cos\theta \\ \therefore \boldsymbol{a} \cdot \boldsymbol{b} &= \|\boldsymbol{a}\|\|\boldsymbol{b}\| \cos\theta \end{aligned}$$

this means that if $\boldsymbol{a} \cdot \boldsymbol{b} = 0$, $\boldsymbol{a} \perp \boldsymbol{b}$.

Matrix multiplication is also known as finding the inner dot product because

$$\boldsymbol{a} \cdot \boldsymbol{b} = \boldsymbol{a}^T \boldsymbol{b}$$

Determinant:

Geometrically, the determinant is the signed volume of the parallelogram (2D) generated by its vectors.[11] And only square matrices can have determinants since logically, if a matrix is non square, some vectors would have too many elements while others have too little. The determinant of a 2×2 square matrix is noted and calculated using the formula[12]

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

[8] Ron Larson and David C. Falvo, *Elementary Linear Algebra*, 6th ed. (Boston: Richard Stratton, 2009), 282.
[9] Larson and Falvo, *Elementary Linear Algebra*, 282.
[10] Larson and Falvo, *Elementary Linear Algebra,* 282.; Virtually Passed, "Dot product and angle between two vectors proof," YouTube, October 11, 2016, video, 7:52, https://www.youtube.com/watch?v=bbBGgHDhmVg.
[11] Mark Demers, "A Derivation of Determinants," Accessed January 6, 2022, faculty.fairfield.edu/mdemers/linearalgebra/documents/2019.03.25.detalt.pdf, 1.
[12] Demers, "A Derivation of Determinants," 1.

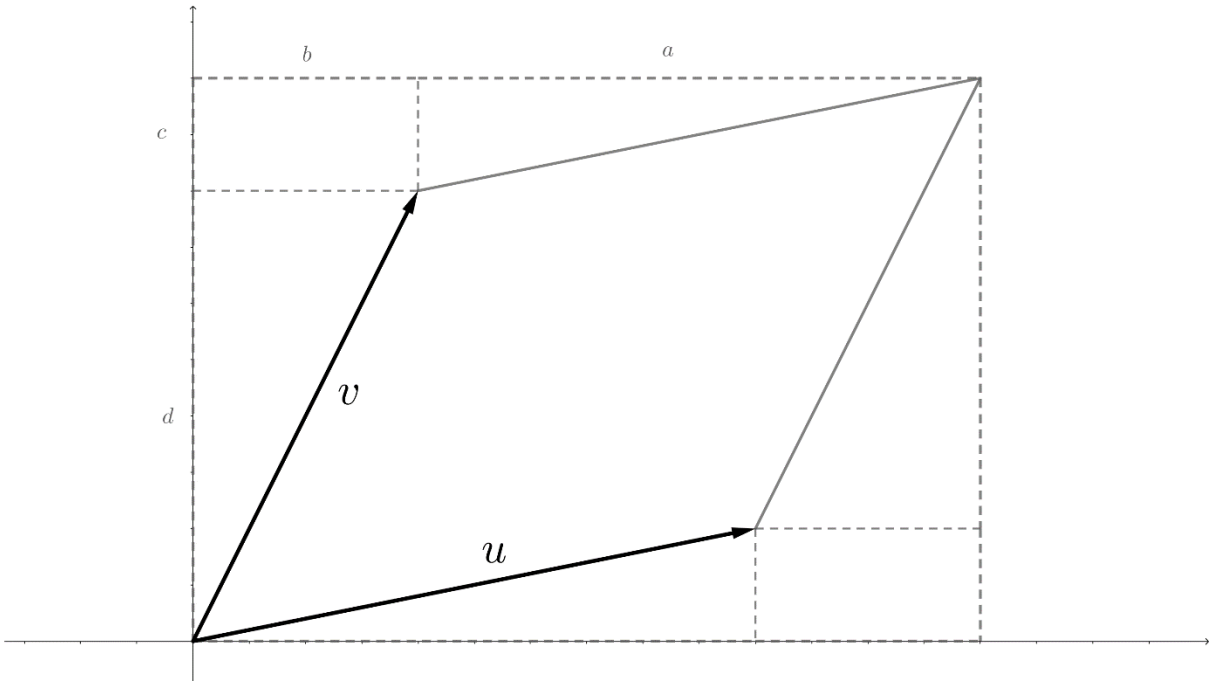

Figure 3 2x2 determinant drawn using GeoGebra

Proof:[13]

$$\begin{aligned} |A_{2\times 2}| &= (a+b)(c+d) - ac - bd - 2bc \\ &= ac + ad + bc + bd - ac - bd - 2bc \\ &= ad - bc \end{aligned}$$

The full exploration of determinants deserves a paper of its own and a complete derivation is therefore outside the scope of this paper. For the sake of this paper only the method of calculation will be touched upon. The Laplace expansion for a $n \times n$ determinant is given by[14]

$$|A| = \sum_{j=1}^{K} C_{ij} A_{ij}$$

Where C_{ij} is the cofactor given by: [15]

$$C_{ij} = (-1)^{i+j} M_{ij}$$

M_{ij} is the $(n-1) \times (n-1)$ matrix obtained from A by removing its corresponding i-th row

[13] Khan Academy, "Proof: Matrix determinant gives area of image of unit square under mapping | Matrices | Khan Academy," YouTube, March 10, 2021, video, 4:18, https://www.youtube.com/watch?v=_OiMiQGKvvc.
[14] Marco Taboga, "The Laplace expansion, minors, cofactors and adjoints," 2021, https://www.statlect.com/matrix-algebra/Laplace-expansion-minors-cofactors-adjoints.
[15] Taboga, "The Laplace expansion, minors, cofactors and adjoints."; Samuel R. Buss, "Some proofs about determinants," 2003, https://mathweb.ucsd.edu/~sbuss/CourseWeb/Math20F_2003S/determinants.pdf, 1.

and j-th column.[16]

Example for $|A_{3\times3}|$:

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = +a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

Example for $|A_{4\times4}|$:

$$\begin{vmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{vmatrix} = a\begin{vmatrix} f & g & h \\ j & k & l \\ n & o & p \end{vmatrix} - b\begin{vmatrix} e & g & h \\ i & k & l \\ m & o & p \end{vmatrix} + c\begin{vmatrix} e & f & h \\ i & j & l \\ m & n & p \end{vmatrix} - d\begin{vmatrix} e & f & g \\ i & j & k \\ m & n & o \end{vmatrix}$$

Transpose of matrix:

The transpose of a matrix is noted using a superscript T. The transpose of a matrix turns the rows of a matrix into its columns and vice versa:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

Identity matrix:

The identity matrix is a special matrix usually noted as *I*. It is a square matrix with 1 as its diagonal elements and its off-diagonal elements are all 0.[17]

$$I_n = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 1 \end{bmatrix}$$

The identity matrix functions similar to 1 as any matrices multiplied by the identity matrix will equal itself, however expanded into the dimension that I belongs in. E.g., $1x = x \; and \; A_{2\times2}I_{3\times3} = I_{3\times3}A_{2\times2} = A_{3\times3}$.

Inverse of matrix:

Since division is not a valid operation for matrices, matrix inversion is used to replace division. The specifics of inversing a matrix will not be touched on since it is not needed for PCA. But there are certain properties that needed be introduced. Similar to inverses with fractions, $\frac{1}{x}\cdot\frac{x}{1} = 1$, $AA^{-1} = A^{-1}A = \; I$ where A is any $m \times n$ matrix and I is the identity matrix. Since matrix multiplication is not commutative, if there is a case where $AB = BA = \; I$, then B is also the inverse of A^{-1}.

[16] Taboga, "The Laplace expansion, minors, cofactors and adjoints."; Buss, "Some proofs about determinants," 1.
[17] Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 113.

There are also cases where a matrix does not have an inverse. For example, $\frac{0}{n}$ is undefined. A matrix with a determinant of 0 also do not have an inverse. We know that[18]

$$AA^{-1} = I$$

then it follows

$$\det(AA^{-1}) = \det(I)$$
$$\det(A)\det(A^{-1}) = \det(I)$$

$\det(AA^{-1}) = \det(A)\det(A^{-1})$ because finding the determinant before or after linear transformation yield the same result. The full proof for the distributive property of determinants can be found on proof wiki.[19]

$$\det(A^{-1}) = \frac{1}{\det(A)}$$

Therefore if $|A| = 0$ the answer is undefined.

Dimensionally, invertible matrices must also be square.[20] Due to the non-commutative property of matrix multiplication, tall matrices are not right invertible while wide matrices are not left invertible.[21] When a matrix is noninvertible, it is called a singular matrix.[22]

Eigenvalues and eigenvectors:

According to the Merriam-webster dictionary, eigenvector is a nonzero vector which when linearly transformed, simply equals to a scalar multiplied to the original vector.[23] By the definition of eigenvectors, we reach the formula $A\vec{v} = \lambda\vec{v}$. Where the linear transformation applied to $\vec{v}$ by $A_{n\times n}$ is simply a scalar multiple λ, the eigenvalue, applied onto $\vec{v}$. From the formula, we can solve for λ.

$$A\vec{v} = \lambda\vec{v}$$
$$A\vec{v} - \lambda\vec{v} = \vec{0}$$
$$(A - \lambda)\vec{v} = \vec{0}$$

Since $A_{n\times n}$ cannot subtract by a single value, the value must be a matrix of the same dimensions. Hence λI.

$$(A - \lambda I)\vec{v} = \vec{0}$$

If $(A - \lambda I)^{-1}$ exists then,

---

[18] Jared Ronning, "What is the determinant of the inverse of a matrix?," Quora, Accessed January 6, 2022, https://qr.ae/pG63sw.
[19] Proofwiki, s.v. "Determinant of Matrix Product," Accessed December 30, 2021, https://www.proofwiki.org/wiki/Determinant_of_Matrix_Product.
[20] Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 202.
[21] Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 202.
[22] Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 202.
[23] *Merriam-Webster.com Dictionary*, s.v. "eigenvector," accessed January 7, 2022, https://www.merriam-webster.com/dictionary/eigenvector.

$$(\mathrm{A}-\lambda\ I)^{-1}(\mathrm{A}-\lambda\ I)\vec{v} = (\mathrm{A}-\lambda\ I)^{-1}\vec{0}$$

$$\therefore\ \vec{v} = \vec{0}$$

This means that $(A-\lambda\ I)$ must be a singular matrix for a nontrivial solution. Therefore, a nontrivial solution will be reached only when $det(A-\lambda\ I) = 0$, resulting to the characteristic polynomial of the square matrix.[24]

Example:

$$A = \begin{bmatrix} 1 & 0 \\ 1 & -4 \end{bmatrix}$$
$$\lambda\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

Since we are primarily interested in finding $det(A-\lambda\ I) = 0$, we will ignore the rest of the equation and focus on $A - \lambda$ for the time being.

$$\begin{bmatrix} 1 & 0 \\ 1 & -4 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \vec{0}$$

$$\begin{bmatrix} 1-\lambda & 0 \\ 1 & -4-\lambda \end{bmatrix} = \vec{0}$$

$$\begin{vmatrix} 1-\lambda & 0 \\ 1 & -4-\lambda \end{vmatrix} = 0$$

Finding the determinant using the formula $ad - bc$, we get the characteristic polynomial:

$$(1-\lambda)(-4-\lambda) - (0)(1) = 0$$
$$\lambda^2 + 3\lambda - 4 - 0 = 0$$
$$(\lambda + 4)(\lambda - 1) = 0$$
$$\lambda = -4, 1$$

Since there are two different eigenvalues, there would be two eigenvectors. Replacing λ with the calculated eigenvalues, eigenvectors are calculated.

$$(\mathrm{A}-\lambda\ I)\vec{v} = \vec{0}$$
$$(\mathrm{A}-(-4\ I))\vec{v} = \vec{0}$$
$$\begin{bmatrix} 1 & 0 \\ 1 & -4 \end{bmatrix} - \begin{bmatrix} -4 & 0 \\ 0 & -4 \end{bmatrix} = 0$$
$$\begin{bmatrix} 5 & 0 \\ 1 & 0 \end{bmatrix} = 0$$

Gaussian elimination is simply solving a linear system of equations using a different notation, the process and result are the same.[25]

---

[24] James B. Carrell, *Fundamentals of Linear Algebra* (self-pub., UBC, 2005), https://www.math.ubc.ca/~carrell/NB.pdf, 263-264.

[25] Jim Hefferon, *LINEAR ALGEBRA*, 4th ed. (Vermont: Saint Michael's College Colchester, 2020), 2 - 3.; Carrel, *Fundamentals of Linear Algebra*, 26, 30.

$$\begin{bmatrix} 5 & 0 & |0 \\ 1 & 0 & |0 \end{bmatrix} \xrightarrow[R_2 - \frac{1}{5}R_1 \to R_2]{} \begin{bmatrix} 1 & 0 & |0 \\ 0 & 0 & |0 \end{bmatrix}$$

$$[1 \quad 0 \quad |0]$$

$$x_1 + 0x_2 = 0$$
$$x_1 = 0$$
$$x_2 = x_2$$
$$\overrightarrow{v_1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

And using the same process

$$\overrightarrow{v_2} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$$

PCA

PCA is the process of compressing multidimensional data through a reduction of dimensions while retaining maximum variance by projecting each data point of a higher dimension onto principal components (PCs) of a lower dimension through linear transformation.[26] Linear transformation in PCA is achieved through matrix multiplication. The projection is achieved by eigen decomposition of either the data's covariance matrix or correlation matrix. In the case of image processing, PCA is more often done through singular value decomposition (SVD) because it is computationally faster than PCA.[27] Although computationally SVD is faster for finding PCs, I chose not to use SVD because SVD is a more advanced branch under PCA, so PCA is thus more conceptually simpler to understand and also much easier to program with my limited programming skills.

[26] I.T. Jollife, *Principal Component Analysis*, 2nd ed. (New York: Springer, 2002), 1.
[27] Jollife, *Principal Component Analysis,* 45.

Image:

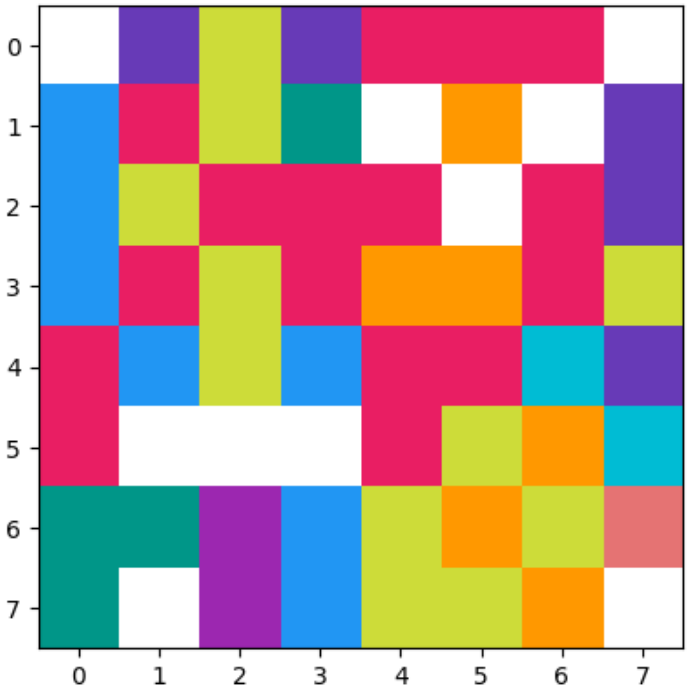

Figure 4 8x8 image drawn using pixilart for PCA

This is an 8×8 image I drew on pixilart.com. I drew this image because it is a smaller image compared to an online image (FHD), 8×8 vs. 1920×1080 (FHD), this allows for easier debugging since the data set is very small, so I am able to manually check the computed values. The faster computational time that comes with a smaller data set also speeds up my workflow in the numerous iterations of my program while debugging. Naturally, after completing the program, the program can perform PCA for a larger image, but would take significantly longer.

Colors in computer:

The computer stores color based on the primary colors, but instead of RYB (red, yellow, blue), it uses RGB instead since light is additive.[28] For each color, the computer stores 3 different values each representing one part of RGB with values ranging from $0 - 255$ with 0 being the lightest and 255 the darkest. For example, $[0, 0, 0]$ makes white and $[255, 255, 255]$ black. Therefore, the data used to perform PCA will come from the RGB matrices.

Feature scaling:

When we get a set of data, one important step is feature scaling. Feature scaling refers to the act of either standardizing or normalizing the data. Standardization organizes the data around 0 with a standard deviation of 1.[29] This method of feature scaling is good for comparing

[28] John Steinmetz, "WHY IS WEB COLOR RGB AND NOT RYB?," April 27, 2016, https://medium.com/@austincoding/why-is-web-color-rgb-and-not-ryb-47f4aea0f557.; Boyd and Vandenberghe, *Introduction to Applied Linear Algebra*, 6.
[29] Swetha Lakshmanan, "How, When, and Why Should You Normalize / Standardize / Rescale Your Data?,"

data sets when they do not follow the same scaling.[30] For example, comparing a data set that varies from $0-100$ and another that varies from $0-45$. Standardization of data is helpful when the data follows a Gaussian distribution (bell curve), but it is still possible to perform standardization on a non-normally distributed data.[31] However, performing standardization on a skewed dataset would still result in a skewed curve. To standardize the data, one would find the z-score given by the formula:

$$Z = \frac{X_i - \mu}{\sigma}$$

where X is the original data, μ is the mean of X and σ is the variance of X.

But since all of the data are within the same scale, $0-255$, there is no need or reason to perform standardization.

On the other hand, normalization places all the data points between $0-1$. Achieved by the formula:[32]

$$X_{normalized} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

where X is the raw data.

Applying normalization thus does not change the relationship between the variables, it only affects the scale in which the data is presented in. Before normalization, values will be between $0-255$ and after, $0-1$. Therefore, since there is no reason to perform standardization because the data is using the same scaling while normalization only changes the values presented in, there is no reason to standardize or normalize the RGB data set. However, after some experimentation, it appears that the cv2 library in which I am using to reconstruct the image after applying PCA requires the data to be normalized to be properly displayed, therefore all the RGB data will be normalized post PCA.

Two approaches

I will be experimenting with two different approaches when performing PCA. They are mathematically identical in calculation. But different in how the data is manipulated. The two approaches that I will be using are the pixels approach and the colors approach.

The Pixels approach

Since the computer stores 3 separate color value (RGB) for each pixel on the screen, for a 8×8

May 16, 2019, https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff.

[30] Aniruddha Bhandari, "Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization," April 3, 2020, https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

[31] Lakshmanan, "How, When, and Why Should You Normalize / Standardize / Rescale Your Data?."

[32] Bhandari, "Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization."

image that would mean the image would return three 8×8 matrix each representing red, green or blue. And each element of the three RGB matrix would correspond to the color intensity of a single pixel. The pixels approach is based on the idea of reorganizing the three separate RGB matrix by combining them into a larger 64×3 matrix where each row vector represents the color value of an entire pixel. The actual RGB values are obtained through the python library cv2. Refer to Appendix 1 for the full RGB matrices. This can be demonstrated by the table below:

Table 1 reorganizing RGB matrices into 64x3 matrix

C	B	G	R
Row 1	b_{11}	g_{11}	r_{11}
Row 2	b_{12}	g_{12}	r_{12}
		⋮	
Row 64	b_{88}	g_{88}	r_{88}

And converting table 1 into matrix notation we get:

$$C_{64x3} = \begin{bmatrix} 255 & 255 & 255 \\ 103 & 58 & 183 \\ & \vdots & \\ 255 & 255 & 255 \end{bmatrix}$$

see Appendix 2 for the full C_{64x3}.

Variance - covariance Matrix

The idea of PCA is to project a higher dimensional data onto a lower one while retaining as much information as possible. To apply PCA, we first need to find the covariance matrix which measures the variance between multiple variables. Finding the covariance matrix helps to identify the amount of variance between variables and when performing PCA it will help to separate data points with large variances from the small variances. In the end PCA will compress data by retaining only the information that has the largest variances, which in turn retains the 'shape' of the data.

Also known as the variance covariance matrix, the covariance matrix is denoted using Σ. For a covariance matrix of 2 variables, Σ is given by:[33]

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y}) = E[(X_i - E[X])(Y_i - E[Y])]$$

The covariance formula is also nearly identical to the variance formula

[33] "A geometric interpretation of the covariance matrix," Accessed January 7, 2022, https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/, 2 – 3.

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2$$

because they are mathematically the same formula, except one find the covariance between different variables while the other finds the variance of a single variable. Since the data set represents the entire population of the color values, Bessel's correction, dividing by $n-1$, is not necessary.[34] Therefore the data is divided by n.

The covariance formula can be extended for 3 variables as:

$$\Sigma = E[(X_i - E[X])(Y_i - E[Y])(Z_i - E[Z])]$$

and so on.

From the formula we can also see that the covariance matrix is also symmetric, meaning $\Sigma = \Sigma^T$. Because $E[(X_i - E[X])(Y_i - E[Y])] = E[(Y_i - E[Y])(X_i - E[X])]$. The covariance matrix would have variance of the variables along its diagonal with its off diagonals as covariance, hence the name variance – covariance matrix. Expanding the covariance matrix with two variables X and Y:

$$\Sigma = \begin{bmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{bmatrix} = \begin{bmatrix} var(x) & cov(x,y) \\ cov(y,x) & var(y) \end{bmatrix}$$

So, for $C_{64\times3}$, $cov(C_{64\times3}) = \Sigma_{64\times64}$.

To calculate the mean of a matrix for PCA we can use the formula:

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Example:

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 1 & 5 & 7 \end{bmatrix}$$

$$\bar{X} = \frac{1+3+4}{3} = \frac{8}{3}$$

$$\bar{Y} = \frac{13}{3}$$

$$cov(x,x) = \frac{\left(1-\frac{8}{3}\right)^2 + \left(3-\frac{8}{3}\right)^2 + \left(4-\frac{8}{3}\right)^2}{3} = \frac{14}{9}$$

$$cov(y,y) = \frac{\left(1-\frac{13}{3}\right)^2 + \left(5-\frac{13}{3}\right)^2 + \left(7-\frac{13}{3}\right)^2}{3} = \frac{56}{9}$$

$$cov(x,y) = \frac{\left(1-\frac{13}{3}\right)\left(1-\frac{8}{3}\right) + \left(5-\frac{13}{3}\right)\left(3-\frac{8}{3}\right) + \left(7-\frac{13}{3}\right)\left(4-\frac{8}{3}\right)}{3} = \frac{28}{9}$$

[34] Brayton Hall, "The Reasoning Behind Bessel's Correction: n – 1," August 22, 2020, https://towardsdatascience.com/the-reasoning-behind-bessels-correction-n-1-eeea25ec9bc9.

$$\because cov(x,y) = cov(y,x)$$

$$\Sigma = \begin{bmatrix} \frac{14}{9} & \frac{28}{9} \\ \frac{28}{9} & \frac{56}{9} \end{bmatrix}$$

Eigenvectors of a symmetric matrix:

After find the covariance matrix, we then need to find the eigenvectors of the covariance matrix. The reason that eigenvectors are used is because since the covariance matrix is symmetric, the eigenvectors of each distinct eigenvalue will be orthogonal,[35] meaning that they will be perpendicular to each other, which means that the direction of each eigenvector will signify the spread of the data.

Proof that eigenvector of a symmetrical matrix is orthogonal:[36]

For 2 distinct eigenvalues and their eigenvector of *A*

$$Ax_1 = \lambda_1 x_1 \text{ and } Ax_2 = \lambda_2 x_2$$

$$\begin{aligned} \lambda_1(x_1 \cdot x_2) &= (\lambda_1 x_1) \cdot x_2 \\ &= (Ax_1)^T x_2 \end{aligned}$$

distributing the transpose, $(AB)^T = B^T A^T$

$$(Ax_1)^T x_2 = {x_1}^T A^T x_2$$

since *A* is symmetric, $A^T = A$

$$\begin{aligned} {x_1}^T A^T x_2 &= {x_1}^T A x_2 \\ &= {x_1}^T \lambda_2 x_2 \\ &= x_1 \cdot (\lambda_2 x_2) \\ (\lambda_1 - \lambda_2)(x_1 \cdot x_2) &= 0 \\ \lambda_1 - \lambda_2 &\neq 0 \\ \therefore x_1 \cdot x_2 &= 0 \end{aligned}$$

$$x_1 \perp x_2$$

Then by looking at the magnitude of the eigenvector, it would tell us the amount of data spread in the direction of that eigenvector. By sorting the magnitudes, in other words the eigenvalues, from the largest to smallest, we would know the amount variance that each eigenvector represents, or in other words the importance of each eigenvector. This is true because the longer the eigenvector, the larger the data is spreading in the direction of that eigenvector. The percent variance can be calculated using the formula:

$$\frac{\lambda_i}{\sum_{i=1}^{n} \lambda_i} X100 = \%\, Variance$$

[35] Larson and Falvo, *Elementary Linear Algebra*, 452.
[36] Larson and Falvo, *Elementary Linear Algebra*, 452.

where λ are the eigenvalues.

After finding the % variance that each eigenvalue represents, they are then sorted from largest to smallest and graphed on to a cumulative graph. Ranking eigenvalues from largest to smallest helps to identify the relevant eigenvectors to keep during PCA and the cumulative graph is a good visual tool showing how much variance is explained by the eigenvectors.

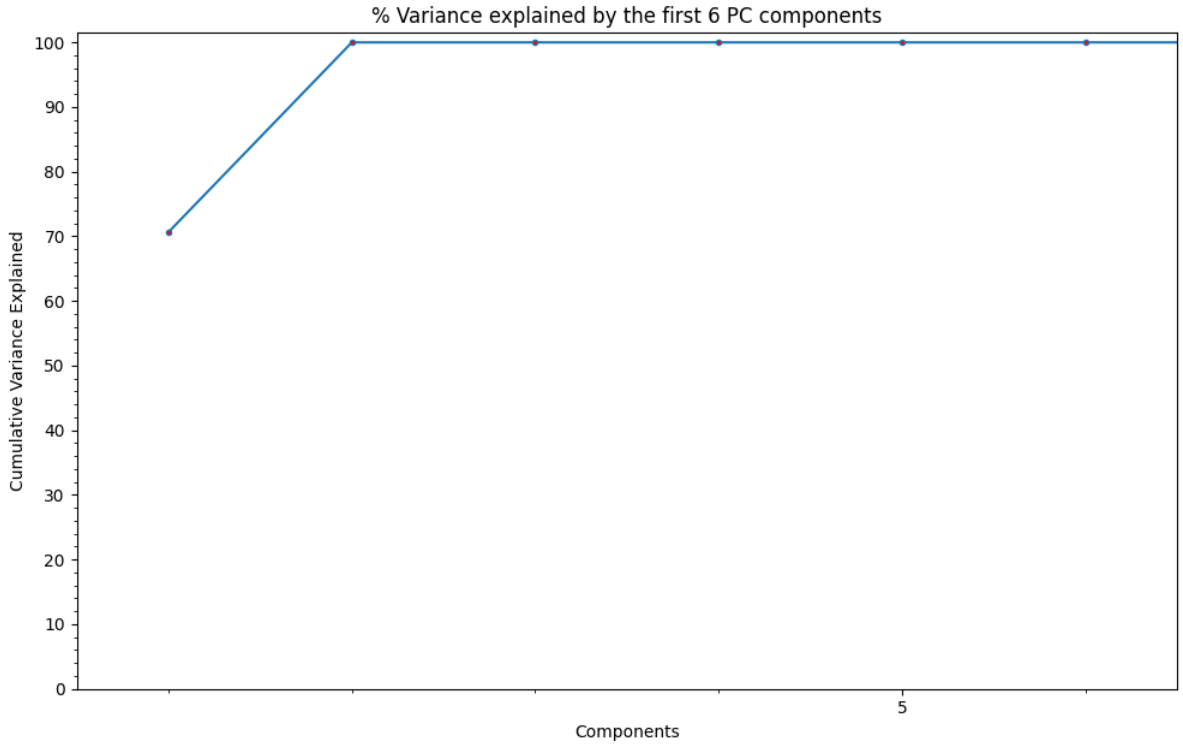

Figure 5 Variance explained by the first 6 components of the 64 X 3 matrix

From figure 4, we can see that the first eigenvector would account for ≈ 70.7% of the variance in the data while the second eigenvector would account for the rest. The amount of variance to retain depends on the application and use case while in some cases is dictated by personal preference as selecting the amount of variance to retain affects how much the data is compressed. The more variance one retains, the closer the compressed data is to the original. By selecting an arbitrary amount of variance one wants to retain, the feature matrix F can be obtained where each column corresponds to an eigenvector.

$$F = [eigenvector_1 \quad eigenvec_2 \quad \cdots \quad eigenvec_n]$$

Principal components (PCs):

After finding the amount of the variance that we wanted to retain by constructing the feature matrix, the PCs can be derived by applying linear transformation:[37]

$$P = FD = F^T \cdot D$$

[37] Lindsay I. Smith, "A tutorial on Principal Components Analysis," February 26, 2002, http://www.iro.umontreal.ca/~pift6080/H09/documents/papers/pca_tutorial.pdf, 16.

Where *P* is the compressed data, *F* is the feature matrix and *D* is the original data. This transformation maps the original data onto the selected eigenvectors. Image compression through PCA is also known as a *lossy* compression because only the most varied data are retained while less important information, the less varied, are discarded.[38]

Reverse PCA

To complete the image compression after finding its PCs, reverse PCA is needed to derive the proper dimensions for the 8×8 image, which is then redrawn using cv2. Solving for D:[39]

$$P = FD$$

$$F^{-1}P = F^{-1}FD$$

Since the feature matrix is composed of eigenvectors of a symmetrical matrix, therefore orthogonal, $F^{-1} = F^{T}$

$$F^{T}P = DI$$

$$F^{T}P = F \cdot P = D$$

this shows that for any given image compressed using PCA, the computer only needs to save the *F* and *P*, which are smaller than the original data, to be able to reconstruct the reduced image, achieving compression. However as previously mentioned, due to a need for the color values to line in between $0 - 1$ for proper visualization as required by cv2, *D* must be normalized after the algebraic reconstruction of the image.

$$D_{normalized} = \frac{D - D_{min}}{D_{max} - D_{min}}$$

Below is a graph comparing the pre compressed image to a few different compressions with varying degrees of variance retained using the pixel approach.

[38] Pamela Fox, "Lossy compression," Accessed January 7, 2022, https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:digital-information/xcae6f4a7ff015e7d:data-compression/a/lossy-compression.
[39] Smith, "A tutorial on Principal Components Analysis," 19.

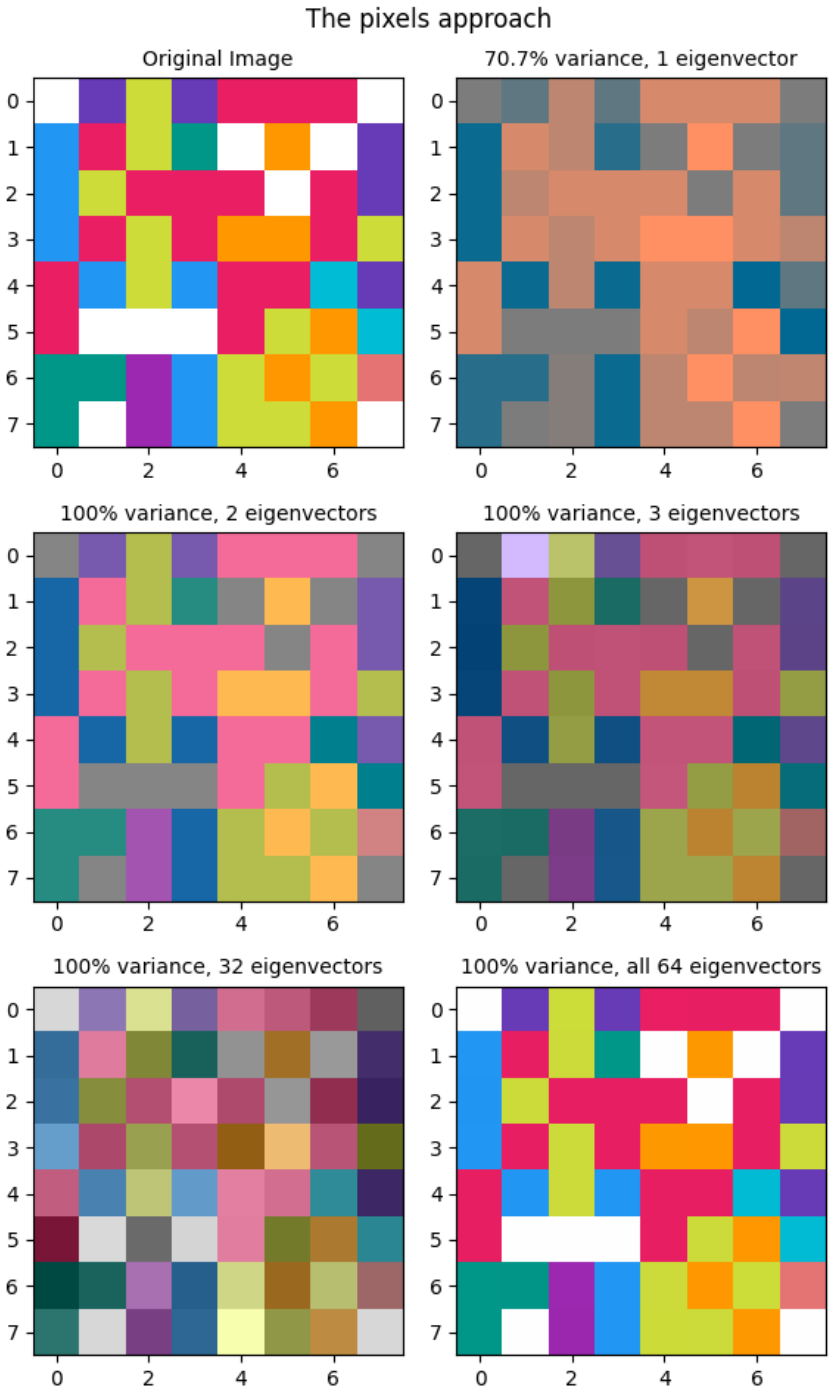

Figure 6 PCA Pixels approach

Since the sixth image which retained all of its eigenvectors is identical to the original, this indicates that PCA was performed correctly. From comparing the original image to the second image which retained 70.7% of the variance, we can see that the basic structure for the position of the pixels are already in place. Where the darker colors are represented by a darker blue while brighter and lighter colors are lighter and redder. From the second graph which retained 100% of the variance, the colors became more defined. Instead of only blue and red, now there are a variety of colors. Although already reached maximum variance retention, by retaining more eigenvectors, the colors became more varied and defined however beginning to

lose its overall structure. The fourth picture which retained three eigenvectors became darker while the fifth picture, which retained half of the total eigenvectors available, expressed even more colors than before but lost its pattern. The increase in color diversity even though retention of data variance has already reached 100% can be explained through the dimensions of the covariance matrix. Because the image started as a 64×64 covariance matrix, meaning that there are a total of 64 dimensions. Therefore, the more eigenvectors retained, the more dimensions are retained during compression, which could explain the increasing color diversity that led to decreasing structure.

Colors approach:

The colors approach is mathematically identical, but instead of organizing the three RGB matrices into a single matrix, PCA is performed on each separate matrix. The RGB matrices can be found in Appendix 1. Below is a graph showing the variance explained by each eigenvector of the three matrices:

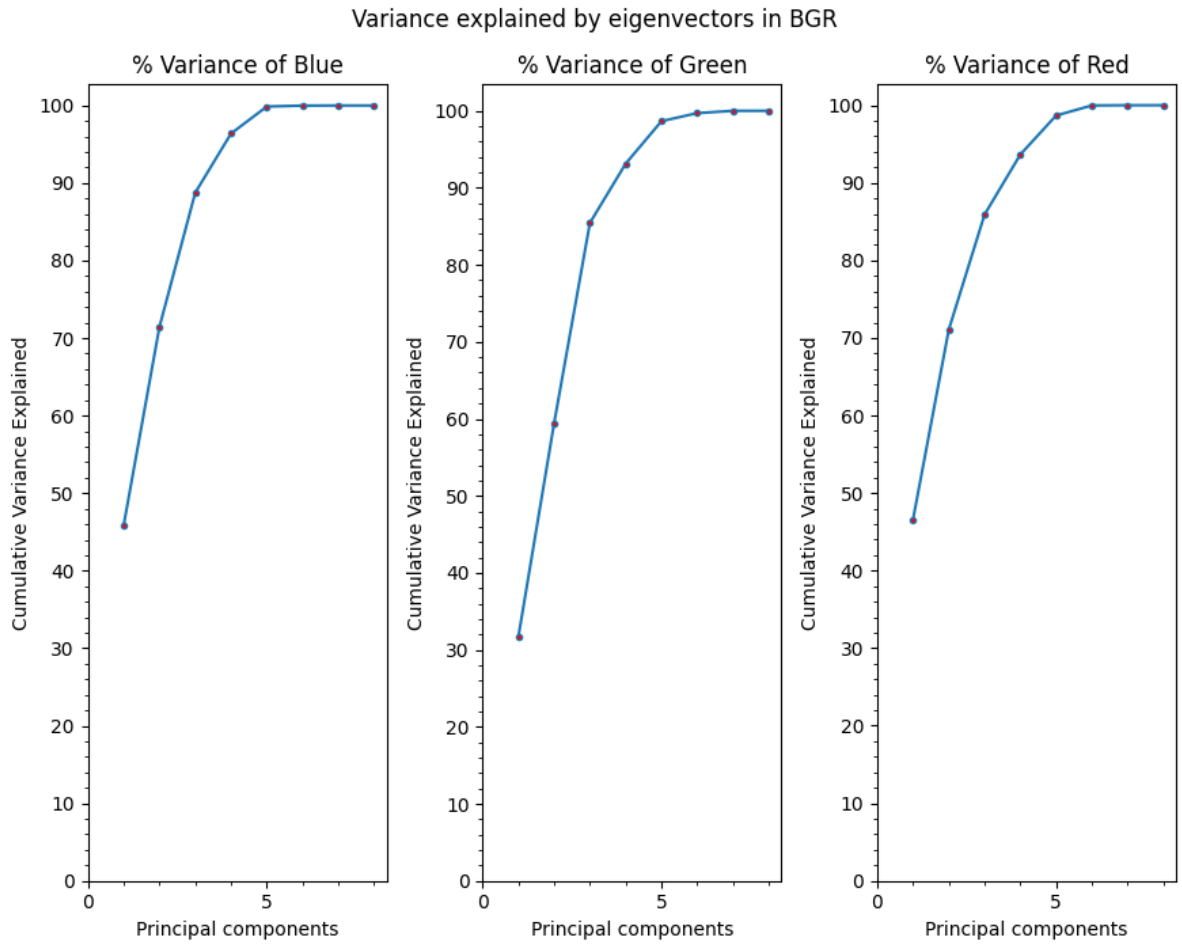

Figure 7 Variance explained by each RGB matrices

Performing PCA:

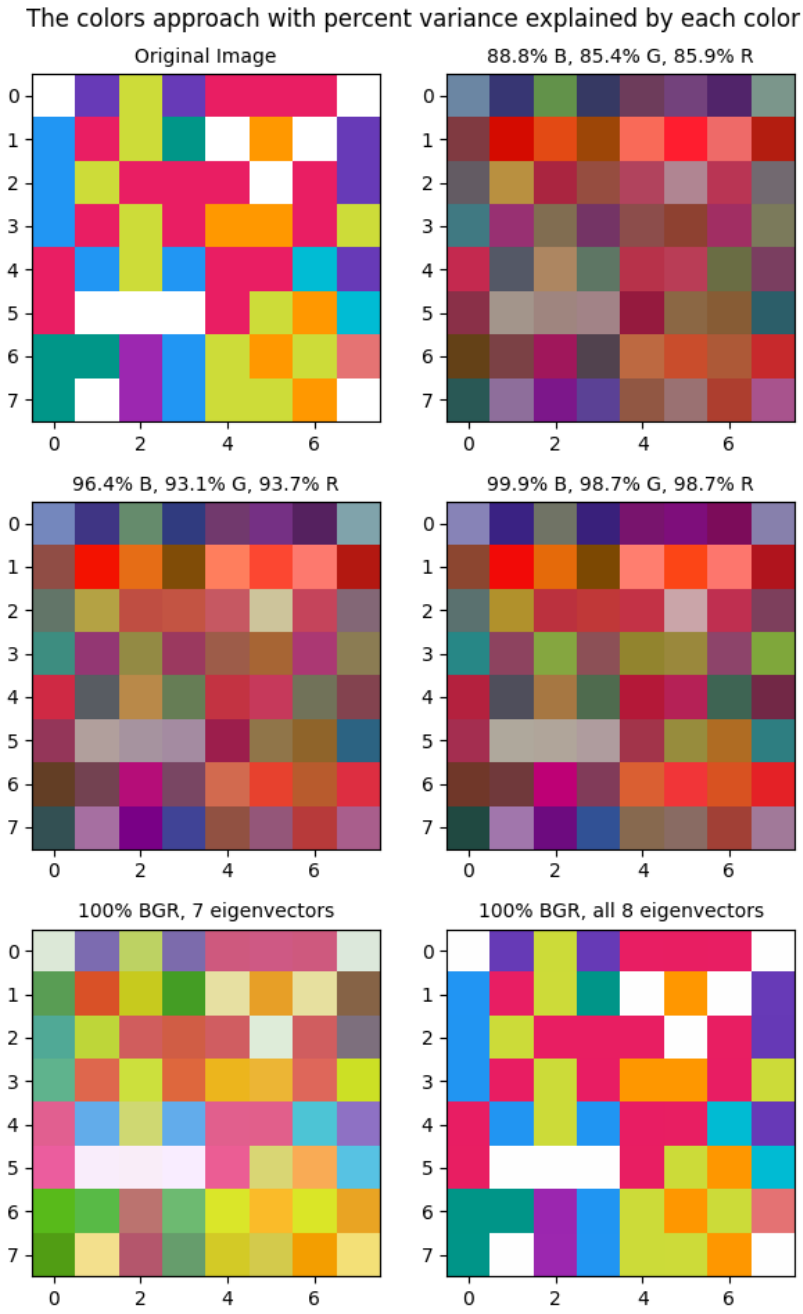

Figure 8 PCA Colors approach

Again, since the sixth image which retained all the eigenvectors is identical to the original, PCA was performed successfully. Different to the pixels approach, the first few compressions were not able to describe the structure of the image but expressed more colors than the pixels approach. What is interesting is that the fifth picture which retained 100% of the variance was close but did not match the original, the same explanation discussed in the pixels approach can be applied here since the fifth image retained only 7 eigenvectors for all three RGB matrices, missing an eigenvector direction.

Reflection:

Both approaches to applying PCA was successful, where the pixels approach was better at expressing the shape of the image but comes at a lower selection of distinct variance values while the colors approach offered a variety of variances to choose from and was good at retaining and expressing the color values at the cost of the image's overall shape. Practically, the pixel approach had a few problems. Not only does it offer less variances to choose from, it also suffers computationally. With the pixel approach, for a standard FHD image, the *C* matrix would be $C_{(1920\times1080)\times x3} = C_{2073600\times3}$. Calculation of the covariance matrix with a matrix this size is very slow, and when even larger images are introduced, matrix *C* will only get larger. In comparison, the colors approach not only offers more variance levels to choose from, it also does not suffer from such problem for a FHD image since only three separate covariance for $R_{1920\times1080}, B_{1920\times1080}\ and\ G_{1920\times1080}$ will need to be calculated. This approach will be a lot faster than the $C_{2073600\times3}$ alternative.

Extension:

An interesting extension would be to attempt to find a good default amount of variance to retain for PCA which maximizes the quality image while minimizing the file size.

Appendix 1

BGR color array values

$$B = \begin{bmatrix} 255 & 103 & 205 & 103 & 233 & 233 & 233 & 255 \\ 33 & 233 & 205 & 0 & 255 & 255 & 255 & 103 \\ 33 & 205 & 233 & 233 & 233 & 255 & 233 & 103 \\ 33 & 233 & 205 & 233 & 255 & 255 & 233 & 205 \\ 233 & 33 & 205 & 33 & 233 & 233 & 0 & 103 \\ 233 & 255 & 255 & 255 & 233 & 205 & 255 & 0 \\ 0 & 0 & 156 & 33 & 205 & 255 & 205 & 229 \\ 0 & 255 & 156 & 33 & 205 & 205 & 255 & 255 \end{bmatrix}$$

$$G = \begin{bmatrix} 255 & 58 & 220 & 58 & 30 & 30 & 30 & 255 \\ 150 & 30 & 220 & 150 & 255 & 152 & 255 & 58 \\ 150 & 220 & 30 & 30 & 30 & 255 & 30 & 58 \\ 150 & 30 & 220 & 30 & 152 & 152 & 30 & 220 \\ 30 & 150 & 220 & 150 & 30 & 30 & 188 & 58 \\ 30 & 255 & 255 & 255 & 30 & 220 & 152 & 188 \\ 150 & 150 & 39 & 150 & 220 & 152 & 220 & 115 \\ 150 & 255 & 39 & 150 & 220 & 220 & 152 & 255 \end{bmatrix}$$

$$R = \begin{bmatrix} 255 & 183 & 57 & 183 & 99 & 99 & 99 & 255 \\ 243 & 99 & 57 & 136 & 255 & 0 & 255 & 183 \\ 243 & 57 & 99 & 99 & 99 & 255 & 99 & 183 \\ 243 & 99 & 57 & 99 & 0 & 0 & 99 & 57 \\ 99 & 243 & 57 & 243 & 99 & 99 & 212 & 183 \\ 99 & 255 & 255 & 255 & 99 & 57 & 0 & 212 \\ 136 & 136 & 176 & 243 & 57 & 0 & 57 & 115 \\ 136 & 255 & 176 & 243 & 57 & 57 & 0 & 255 \end{bmatrix}$$

Appendix 2

Full color matrix $C_{64\times3}$

$$C_{64x3} = \begin{pmatrix}
255 & 255 & 255 \\
103 & 58 & 183 \\
205 & 220 & 57 \\
103 & 58 & 183 \\
233 & 30 & 99 \\
233 & 30 & 99 \\
233 & 30 & 99 \\
255 & 255 & 255 \\
33 & 150 & 243 \\
233 & 30 & 99 \\
205 & 220 & 57 \\
0 & 150 & 136 \\
255 & 255 & 255 \\
255 & 152 & 0 \\
255 & 255 & 255 \\
103 & 58 & 183 \\
33 & 150 & 243 \\
205 & 220 & 57 \\
233 & 30 & 99 \\
233 & 30 & 99 \\
233 & 30 & 99 \\
255 & 255 & 255 \\
233 & 30 & 99 \\
103 & 58 & 183 \\
33 & 150 & 243 \\
233 & 30 & 99 \\
205 & 220 & 57 \\
233 & 30 & 99 \\
255 & 152 & 0 \\
255 & 152 & 0 \\
233 & 30 & 99 \\
205 & 220 & 57 \\
233 & 30 & 99 \\
33 & 150 & 243 \\
205 & 220 & 57 \\
33 & 150 & 243 \\
233 & 30 & 99 \\
233 & 30 & 99 \\
0 & 188 & 212 \\
103 & 58 & 183 \\
233 & 30 & 99 \\
255 & 255 & 255 \\
255 & 255 & 255 \\
255 & 255 & 255 \\
233 & 30 & 99 \\
205 & 220 & 57 \\
255 & 152 & 0 \\
0 & 188 & 212 \\
0 & 150 & 136 \\
0 & 150 & 136 \\
156 & 39 & 176 \\
33 & 150 & 243 \\
205 & 220 & 57 \\
255 & 152 & 0 \\
205 & 220 & 57 \\
229 & 115 & 115 \\
0 & 150 & 136 \\
255 & 255 & 255 \\
156 & 39 & 176 \\
33 & 150 & 243 \\
205 & 220 & 57 \\
205 & 220 & 57 \\
255 & 152 & 0 \\
255 & 255 & 255
\end{pmatrix}$$

Appendix 3

PCA for the pixel approach

```
import numpy as np, pandas as pd, cv2, matplotlib.pyplot as plt
from numpy import float64, linalg as la

# read image
img = cv2.cvtColor(cv2.imread('PCA8X8PixelArt.png'), cv2.COLOR_BGR2RGB)

# split into BGR matrices
blue, green, red = cv2.split(img)
blue_df = pd.DataFrame(data = blue)
green_df = pd.DataFrame(data = green)
red_df = pd.DataFrame(data = red)

# dataframe to numpy array
blue_array = blue_df.to_numpy()
green_array = green_df.to_numpy()
red_array = red_df.to_numpy()

# make them 1d for easier manipulation
one_d_blue_array = blue_array.flatten()
one_d_green_array = green_array.flatten()
one_d_red_array = red_array.flatten()

# new 64X3 matrix representing all pixels
color_matrix = np.empty(shape = [0, 3], dtype = float64)
for i in range (len(one_d_blue_array)):
    color_matrix =
np.append(color_matrix,[[one_d_blue_array[i],one_d_green_array[i],one_d_red_array[i]]],
axis = 0)
color_matrix_array = np.array(color_matrix)
color_matrix_covariance = np.cov(color_matrix_array, rowvar = True, bias = True)

# split into eigenvals, eigenvecs
eigenValue, eigenVector = la.eigh(color_matrix_covariance)

# sort eigenvalue in decreasing order
idx = np.argsort(eigenValue)[::-1]
eigenValue = eigenValue[idx]

# sort eigenvectors according to same index
eigenVector = eigenVector[:,idx]

def img_reduced(data, eigenVector, pc_components):
    # pcomponents
    n_components = pc_components
    feature_vector = eigenVector[:,0:n_components]

    # linear transformation
    compressed = np.dot (feature_vector.T, data)

    # reverse PCA
    reconstruct = np.dot (feature_vector, compressed)

    # spliting into BGR channels
```

```
    final_blue_array = reconstruct[:,0]
    final_green_array = reconstruct[:,1]
    final_red_array = reconstruct[:,2]

    final_blue_array = final_blue_array.reshape(8,8)
    final_green_array = final_green_array.reshape(8,8)
    final_red_array = final_red_array.reshape(8,8)

    # combine BGR matrix
    img_reduced = cv2.merge((final_blue_array, final_green_array, final_red_array))
    # place color values between 0 - 1
    return (img_reduced - np.min(img_reduced)) / np.ptp(img_reduced)

# image comparison
fig = plt.figure(figsize=(8, 6))
fig.suptitle("The pixels approach")
fig.add_subplot(321)
plt.title("Original Image", size = 10)
plt.imshow(img)
fig.add_subplot(322)
plt.title("70.7% variance, 1 eigenvector", size = 10)
plt.imshow(img_reduced(color_matrix_array, eigenVector, 1))
fig.add_subplot(323)
plt.title("100% variance, 2 eigenvectors", size = 10)
plt.imshow(img_reduced(color_matrix_array, eigenVector, 2))
fig.add_subplot(324)
plt.title("100% variance, 3 eigenvectors", size = 10)
plt.imshow(img_reduced(color_matrix_array, eigenVector, 3))
fig.add_subplot(325)
plt.title("100% variance, 32 eigenvectors", size = 10)
plt.imshow(img_reduced(color_matrix_array, eigenVector, 32))
fig.add_subplot(326)
plt.title("100% variance, all eigenvectors", size = 10)
plt.imshow(img_reduced(color_matrix_array, eigenVector, 64))
plt.tight_layout()
plt.show()
```

Cumulative graph for showing percent variance explained by PCs

```
def CumulativeVariancePlot(Values):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(1, 1, 1)
    print((np.cumsum(Values)/np.sum(Values))*100)
    plt.plot(range(1, 65), (np.cumsum(Values)/np.sum(Values))*100, marker = '.',
markerfacecolor='red')
    plt.title("% Variance explained by the first 6 PC components")
    plt.xlabel("Components")
    xmajor_tics = np.arange(0, len(Values) + 1, 5)
    xminor_ticks = np.arange(0, len(Values) + 1, 1)
    ymajor_tics = np.arange(0, 101, 10)
    yminor_ticks = np.arange(0, 101, 2)
    ax.set_xticks(xmajor_tics)
    ax.set_xticks(xminor_ticks, minor=True)
    ax.set_yticks(ymajor_tics)
    ax.set_yticks(yminor_ticks, minor=True)
    plt.xlim([0.5, 6.5])
    plt.ylabel("Cumulative Variance Explained")
```

```
    plt.show()
CumulativeVariancePlot(eigenValue)
```

Appendix 4

PCA for colors approach

```
import numpy as np, pandas as pd, cv2, matplotlib.pyplot as plt
from numpy import float64, linalg as la

# read image
img = cv2.cvtColor(cv2.imread('PCA8X8PixelArt.png'), cv2.COLOR_BGR2RGB)

# split into BGR matrices
blue, green, red = cv2.split(img)
blue_df = pd.DataFrame(data = blue)
green_df = pd.DataFrame(data = green)
red_df = pd.DataFrame(data = red)

# dataframe to numpy array
blue_array = blue_df.to_numpy()
green_array = green_df.to_numpy()
red_array = red_df.to_numpy()

# covariance matrix
b_cov = np.cov(blue_array, rowvar = True, bias = True)
g_cov = np.cov(green_array, rowvar = True, bias = True)
r_cov = np.cov(red_array, rowvar = True, bias = True)

# split into eigenvals, eigenvecs
b_eigenValue, b_eigenVector = la.eigh(b_cov)
g_eigenValue, g_eigenVector = la.eigh(g_cov)
r_eigenValue, r_eigenVector = la.eigh(r_cov)

# sort eigenvalue in decreasing order
b_idx = np.argsort(b_eigenValue)[::-1]
b_eigenValue = b_eigenValue[b_idx]

g_idx = np.argsort(g_eigenValue)[::-1]
g_eigenValue = g_eigenValue[g_idx]

r_idx = np.argsort(r_eigenValue)[::-1]
r_eigenValue = r_eigenValue[r_idx]

# sort eigenvectors according to same index
b_eigenVector = b_eigenVector[:,b_idx]

g_eigenVector = g_eigenVector[:,g_idx]

r_eigenVector = b_eigenVector[:,r_idx]

def img_reduced(b_data, g_data, r_data, b_eivec, g_eivec, r_eivec, BGRpc_components):

    blue, green, red = BGRpc_components

    # pcomponents
    b_feature_vector = b_eivec[:,0:blue]
    g_feature_vector = g_eivec[:,0:green]
    r_feature_vector = r_eivec[:,0:red]
```

```
    # linear transformation
    b_compressed = np.dot (b_feature_vector.T, b_data)
    g_compressed = np.dot (g_feature_vector.T, g_data)
    r_compressed = np.dot (r_feature_vector.T, r_data)


    # reverse PCA
    b_reconstruct = np.dot (b_feature_vector, b_compressed)
    g_reconstruct = np.dot (g_feature_vector, g_compressed)
    r_reconstruct = np.dot (r_feature_vector, r_compressed)

    # spliting into BGR channels
    final_blue_array = b_reconstruct.reshape(8, 8)
    final_green_array = g_reconstruct.reshape(8, 8)
    final_red_array = r_reconstruct.reshape(8, 8)

    # combine BGR matrix
    img_reduced = cv2.merge((final_blue_array, final_green_array, final_red_array))
    # place color values between 0 - 1
    return (img_reduced - np.min(img_reduced)) / np.ptp(img_reduced)

# image comparison
fig = plt.figure(figsize=(8, 6))
fig.suptitle("The colors approach with percent variance explained by each color")
fig.add_subplot(321)
plt.title("Original Image", size = 10)
plt.imshow(img)
fig.add_subplot(322)
plt.title("88.8% B, 85.4% G, 85.9% R", size = 10)
plt.imshow(img_reduced(blue_array, green_array, red_array, b_eigenVector, g_eigenVector,
r_eigenVector, [3, 3, 3]))
fig.add_subplot(323)
plt.title("96.4% B, 93.1% G, 93.7% R", size = 10)
plt.imshow(img_reduced(blue_array, green_array, red_array, b_eigenVector, g_eigenVector,
r_eigenVector, [4, 4, 4]))
fig.add_subplot(324)
plt.title("99.9% B, 98.7% G, 98.7% R", size = 10)
plt.imshow(img_reduced(blue_array, green_array, red_array, b_eigenVector, g_eigenVector,
r_eigenVector, [5, 5, 5]))
fig.add_subplot(325)
plt.title("100% B, 100% G, 100% R", size = 10)
plt.imshow(img_reduced(blue_array, green_array, red_array, b_eigenVector, g_eigenVector,
r_eigenVector, [7, 7, 7]))
fig.add_subplot(326)
plt.title("100% BGR", size = 10)
plt.imshow(img_reduced(blue_array, green_array, red_array, b_eigenVector, g_eigenVector,
r_eigenVector, [8, 8, 8]))
plt.tight_layout()
plt.show()
```

Cumulative graph for showing percent variance explained by PCs

```
# graph
fig = plt.figure(figsize=(8, 6))
fig.suptitle("Variance explained by eigenvectors in BGR")
ax = fig.add_subplot(1, 3, 1)
plt.plot(range(1, 9), (np.cumsum(b_eigenValue) / np.sum(b_eigenValue)) * 100, marker = '.',
markerfacecolor='red')
```

```
plt.title("% Variance of Blue")
plt.xlabel("Principal components")
xmajor_tics = np.arange(0, len(b_eigenValue) + 1, 5)
xminor_ticks = np.arange(0, len(b_eigenValue) + 1, 1)
ymajor_tics = np.arange(0, 101, 10)
yminor_ticks = np.arange(0, 101, 2)
ax.set_xticks(xmajor_tics)
ax.set_xticks(xminor_ticks, minor=True)
ax.set_yticks(ymajor_tics)
ax.set_yticks(yminor_ticks, minor=True)
plt.ylabel("Cumulative Variance Explained")

ax = fig.add_subplot(1, 3, 2)
plt.plot(range(1, 9),(np.cumsum(g_eigenValue) / np.sum(g_eigenValue)) * 100, marker = '.',
markerfacecolor='red')
plt.title("% Variance of Green")
plt.xlabel("Principal components")
xmajor_tics = np.arange(0, len(g_eigenValue) + 1, 5)
xminor_ticks = np.arange(0, len(g_eigenValue) + 1, 1)
ymajor_tics = np.arange(0, 101, 10)
yminor_ticks = np.arange(0, 101, 2)
ax.set_xticks(xmajor_tics)
ax.set_xticks(xminor_ticks, minor=True)
ax.set_yticks(ymajor_tics)
ax.set_yticks(yminor_ticks, minor=True)
plt.ylabel("Cumulative Variance Explained")

ax = fig.add_subplot(1, 3, 3)
plt.plot(range(1, 9),(np.cumsum(r_eigenValue) / np.sum(r_eigenValue)) * 100, marker = '.',
markerfacecolor='red')
plt.title("% Variance of Red")
plt.xlabel("Principal components")
xmajor_tics = np.arange(0, len(r_eigenValue) + 1, 5)
xminor_ticks = np.arange(0, len(r_eigenValue) + 1, 1)
ymajor_tics = np.arange(0, 101, 10)
yminor_ticks = np.arange(0, 101, 2)
ax.set_xticks(xmajor_tics)
ax.set_xticks(xminor_ticks, minor=True)
ax.set_yticks(ymajor_tics)
ax.set_yticks(yminor_ticks, minor=True)
plt.ylabel("Cumulative Variance Explained")
plt.tight_layout()
plt.show()
```

Bibliography

"A geometric interpretation of the covariance matrix." Visiondummy. Accessed January 7. 2022. https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/.

Academy, Khan . "Proof: Matrix determinant gives area of image of unit square under mapping | Matrices | Khan Academy." YouTube. March 10, 2021. Video, 4:18. https://www.youtube.com/watch?v=_OiMiQGKvvc.

Bhandari, Aniruddha. "Feature Scaling for Machine Learning: Understanding the Difference Between Normalization vs. Standardization." April 3, 2020. https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

Boyd, Stephen, and Lieven Vandenberghe. *Introduction to Applied Linear Algebra.* Cambridge: Cambridge University Press, 2018.

Buss, Samuel R. "Some proofs about determinants." Published 2003. https://mathweb.ucsd.edu/~sbuss/CourseWeb/Math20F_2003S/determinants.pdf.

Carrell, James B. *Fundamentals of Linear Algebra*. Self - published, UBC, 2005. https://www.math.ubc.ca/~carrell/NB.pdf, 263-264.

Demers, Mark. "A Derivation of Determinants." Accessed January 6, 2022, faculty.fairfield.edu/mdemers/linearalgebra/documents/2019.03.25.detalt.pdf, 1.

Fox, Pamela. "Lossy compression." Accessed January 7, 2022. https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:digital-information/xcae6f4a7ff015e7d:data-compression/a/lossy-compression.

Hall, Brayton. "The Reasoning Behind Bessel's Correction: n – 1." August 22, 2020. https://towardsdatascience.com/the-reasoning-behind-bessels-correction-n-1-eeea25ec9bc9.

Hefferon, Jim. *LINEAR ALGEBRA*. 4th ed. Vermont: Saint Michael's College Colchester, 2020.

"Introduction to Matrices." Lumen. Accessed January 6, 2022. https://courses.lumenlearning.com/boundless-algebra/chapter/introduction-to-matrices/.

Jollife, I.T. *Principal Component Analysis*. 2nd ed. New York: Springer, 2002.

Lakshmanan, Swetha. "How, When, and Why Should You Normalize / Standardize / Rescale Your Data?." May 16, 2019. https://towardsai.net/p/data-science/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff.

Larson, Ron, and David C. Falvo. *Elementary Linear Algebra*. 6th ed. Boston: Richard Stratton, 2009.

Malouin, Alain. *Introduction To Vectors*. Translated by Claudia de Fulviis. Québec: P.P.I. inc., 2008.

Passed, Virtually. "Dot product and angle between two vectors proof." YouTube. October 11, 2016. Video, 7:52. https://www.youtube.com/watch?v=bbBGgHDhmVg.

Smith, Lindsay I. "A tutorial on Principal Components Analysis." February 26, 2002. http://www.iro.umontreal.ca/~pift6080/H09/documents/papers/pca_tutorial.pdf, 16.

Steinmetz, John. "WHY IS WEB COLOR RGB AND NOT RYB?." April 27, 2016. https://medium.com/@austincoding/why-is-web-color-rgb-and-not-ryb-47f4aea0f557.

Taboga, Marco. "The Laplace expansion, minors, cofactors and adjoints." Published 2021. https://www.statlect.com/matrix-algebra/Laplace-expansion-minors-cofactors-adjoints.