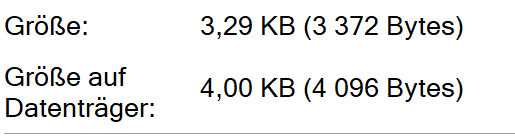


**Punkt 0:**

Um unsere Datensätze von .PNG zu .JPEG/.JPEG2000/.JPRG XR zu komprimieren, haben wir uns für das Programm XnConvert entschieden, da diese alle von uns benötigten Dateitypen unterstützt. Die Einstellung für unsere Gesichtserkennungsdatensätze haben wir wie folgt gewählt:

Die von uns für alle Datenformate gewählte Größe beträgt nunmehr ca. 3,30 KB wie in folgender Abbildung zu sehen ist.



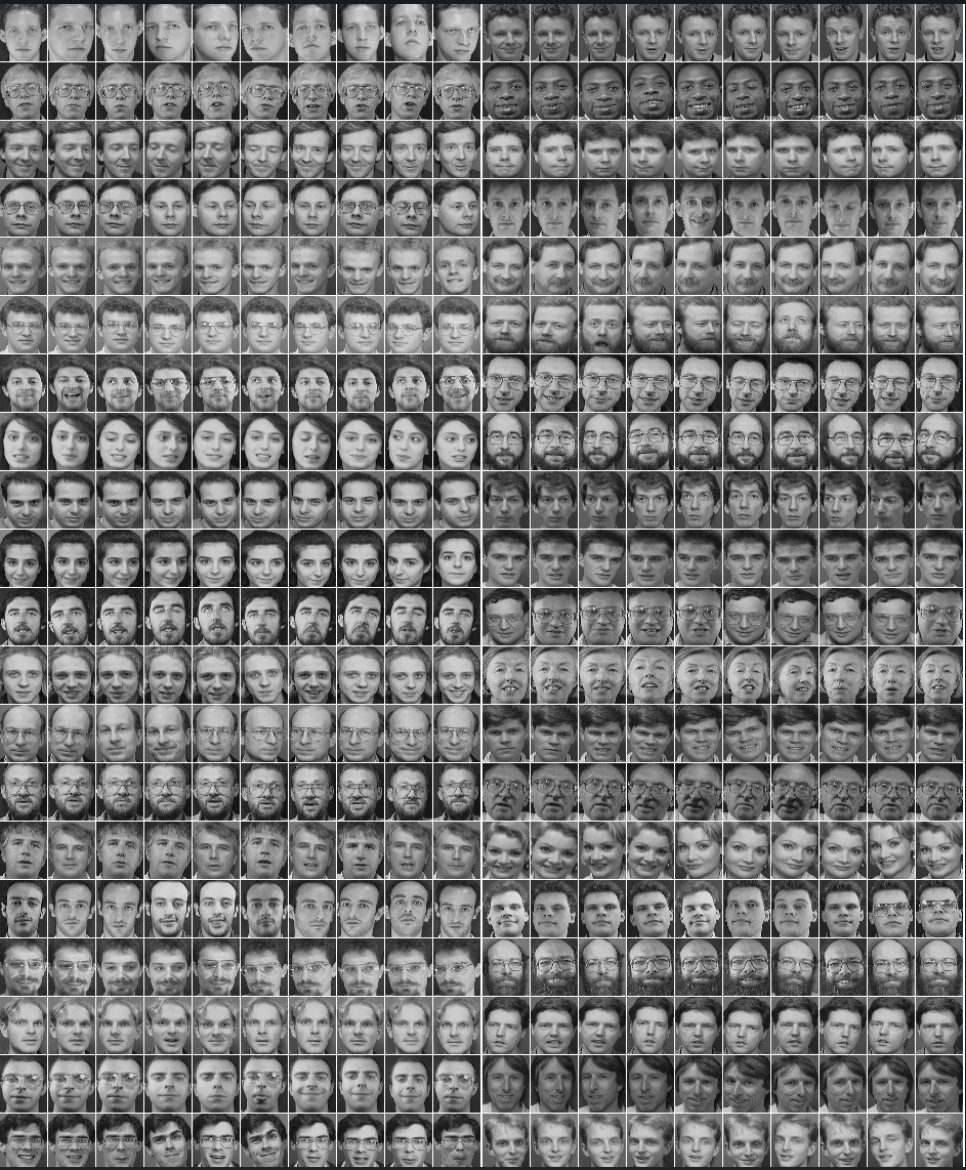
Die originale Datengröße betrug 11 KB, womit sich eine Kompressionsrate von 1:3 ergibt.

XnConvert ermöglicht das Festlegen der oben genannten Kompressionsrate für das Datenformat Jpeg2000. Für sämtliche andere Datenformate wurden die Parameter so eingestellt, dass man auf die gewünschten 3,30 KB kommt.

Bei dem von uns gewählten Datensatz, handelt es sich nunmehr um den AT&T-Gesichtsdatensatz[[1]](#footnote-1), welcher früher 'The ORL Database of Faces' genannt wurde. Dieser enthält einen Satz von Gesichtsbildern, die zwischen April 1992 und April 1994 im Labor aufgenommen wurden.

Das Dataset besteht aus 400 Bildern, die eine Größe von 64x64 aufweisen. Innerhalb dieser 400 Bilder finden sich 40 unterschiedliche Personen. Daraus ergibt sich, dass es für jede Personen 10 unterschiedliche Bilder gibt. Diese unterscheiden sich von Lichtverhältnis, Betrachtungswinkel und Gesichtsausdruck. Einen Überblick über das Dataset verschafft das unten beigefügte Bild auf Seite 3.

Alle Bilder haben eine Abmessung von 92x112 und liegen bereits in schwarz-weiß vor.

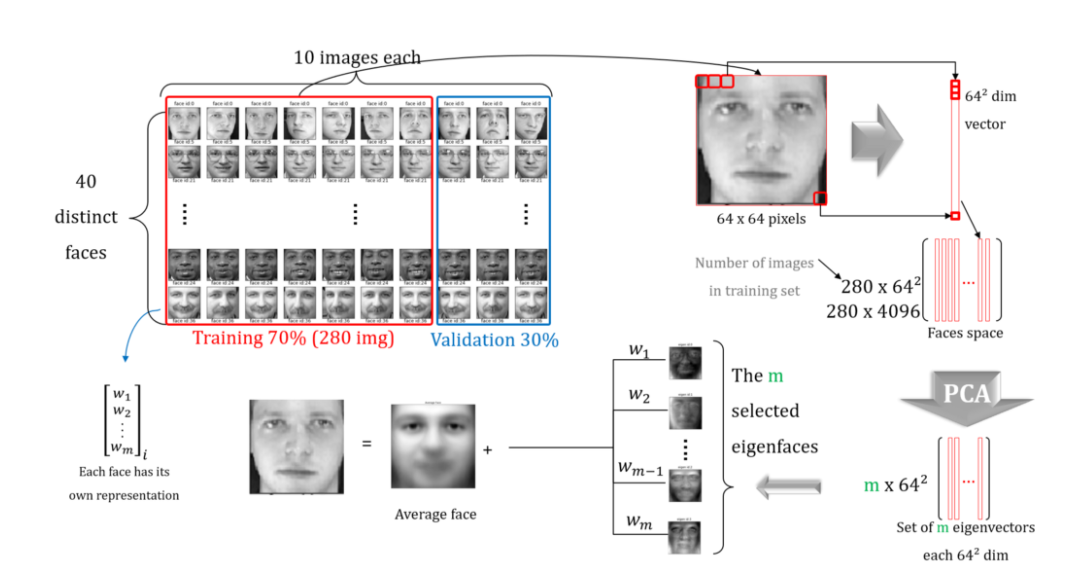


**Punkt 1 & Punkt 5:**

In dieser Studie wurde die Gesichtserkennung mit den Bildern aus dem AT&T-Datensatz durchgeführt. Dies wurde in Python implementiert und brachte folgende Ergebnisse



Es werden für unseren Fall 7 Bilder als Trainingsdatensätze pro Person ausgewählt und 3 als Testdatensätze. Der erste Schritt besteht darin, alle Gesichter des Trainingsdatensatzes zu normalisieren, indem alle gemeinsamen Merkmale zwischen diesen Gesichtern entfernt werden, so dass jedes Gesicht nur noch seine einzigartigen Merkmale aufweist. Dies geschieht, indem das durchschnittliche Gesicht von jedem Gesicht entfernt wird.

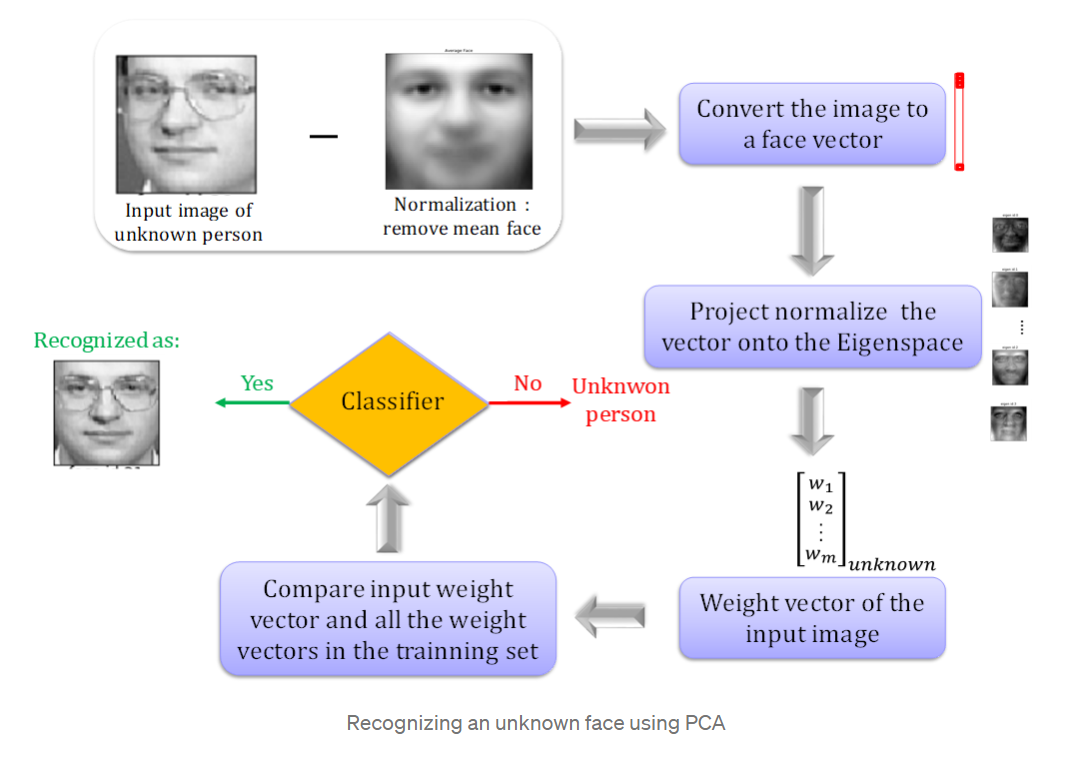


Die Wahl der Anzahl der Komponenten *m* wird entsprechend der besten erhaltenen Genauigkeit bei der Verwendung von Klassifikatoren auf unseren Daten getroffen. Das Verfahren besteht aus einer Schleife über mehrere Komponenten und der Konstruktion eines PCA-Modells für jede angegebene Anzahl von Hauptkomponenten. Dann werden wir einen Klassifikator erstellen und die Genauigkeit aus der Konfusionsmatrix berechnen, um das Diagramm zu erhalten, aus dem die beste Anzahl von Komponenten gewählt werden kann.

Sobald die *m* Eigenflächen ausgewählt sind, können wir jedes Gesicht aus dem Trainingssatz mit diesen Eigenflächen reproduzieren, wie im Bild gezeigt.

Jedes Gesicht aus dem Trainingssatz kann als gewichtete Summe der *m* ausgewählten Eigenflächen plus der mittleren Fläche projiziert werden. Die jeder Eigenfläche zugeordneten Gewichte stellen den Beitrag dieser Eigenfläche zur Reproduktion des Gesichtsoriginals dar.

Sobald alle Gesichter des Trainingssatzes in ihre entsprechenden Gewichtsvektoren umgewandelt sind, können wir die Trainingsgesichter durch ihre Darstellung im Eigenraum reproduzieren.



Um ein unbekanntes Gesicht zu erkennen, führen wir die gleichen Schritte durch, die auch bei den Trainingsbildern angewendet wurden. Wir beginnen mit der Normalisierung dieses Gesichts (Entfernen des Durchschnittsgesichts) und wandeln es in einen Vektor um. Dann projizieren wir den normalisierten Gesichtsvektor auf den zuvor mittels PCA berechneten Eigenraum, was bedeutet, dass wir das unbekannte Gesicht als eine Kombination der *m* Eigenflächen darstellen.

Sobald wir den Gewichtsvektor des unbekannten Gesichts erhalten haben, vergleichen wir ihn im nächsten Schritt mit allen Gewichtsvektoren unseres Trainingssatzes unter Verwendung der euklidischen Distanz als Metrik.

Wenn der Abstand zwischen den beiden Vektoren über einem bestimmten Schwellenwert liegt, sagen wir, dass es sich um ein unbekanntes Gesicht handelt, andernfalls sehen wir die Person, die dem kleinsten Abstand entspricht und identifizieren sie als diese Person. [[2]](#footnote-2)

Beispiel für die Bilder siehe unten, wobei versucht wird für die Bilder 2 und 3 diese zu Bild 1 zu zuordnen.



Und kommen dabei zu folgendem Ergebnis:

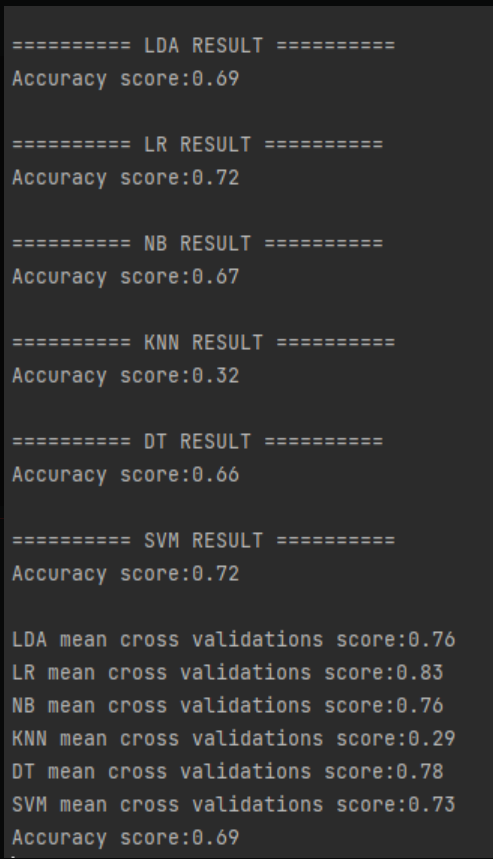
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Hier kann man gut erkennen, dass das System die Person 2 und 3 zu als Person 1 identifiziert, da hier die Distanz am geringsten ist.

Eine robustere Technik besteht darin, einen Klassifikator zu verwenden, anstatt die Eingabe mit dem nächstgelegenen Gesicht aus dem Trainingssatz abzugleichen.

Die Klassifikatoren, die nach der Reduzierung der Bilddimensionen eingeführt werden, sind:

LDA Linear Discriminant Analysis:

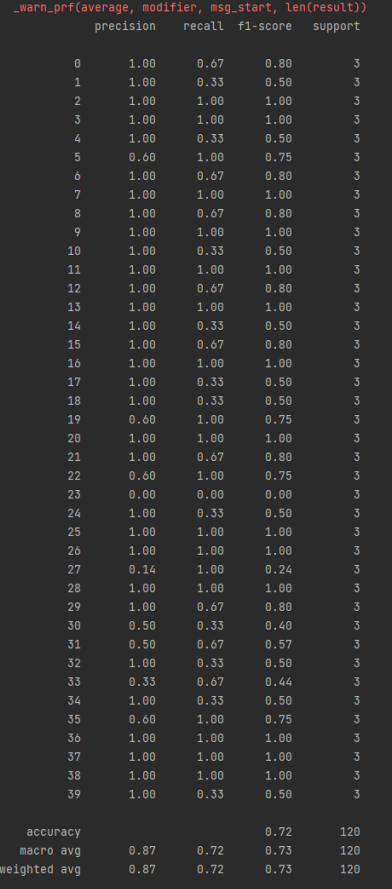
LR Logistic Regression

NB Gaussian NB

KNN Kenighobrs Classifier

DT Decisio Tree Clasifier

Es wurde ein maximaler Accuracy Score von 72% bei JPEG Daten erzielt mit den beiden Klassifikatoren LR und SVM.

****

In der Zeile Precision kann man erkennen wie gut die Gesichtserkennung funktioniert hat. Dabei ist 1 der beste Wert und Richtung 0 wird die Erkennung immer schlechter.[[3]](#footnote-3)

**Punkt 2:**

**full-reference quality metric:**

Versucht die Qualität eines Testbildes anhand eines Referenzbildes zu vergleichen, von dem angenommen wird, dass es eine perfekte Qualität hat. Z.b das Original Bild im Vergleich zu einem JPEG komprimierten Bild

Hier haben wir uns für Python entschieden und die von Python zur Verfügung gestellten Funktionen für die full-reference Metric implementiert.[[4]](#footnote-4)

Verwendete Funktionen:

• **Mean Squared Error (MSE)** misst die durchschnittliche quadratische Different zwischen dem Original und dem komprimierten Bild. Es wird jeder Pixel in der Helligkeitskomponente betrachtet.

*Hier gilt umso niedriger desto besser, MSE = 0 gibt es nur wenn die Bilder ident sind.*

• **Peak Signal-to-Noise Ratio (PSNR)** wird vom MSE abgeleitet und gibt das Verhältnis der maximalen Pixelintensität zur Leistung der Verzerrung an. Hier gilt umso größer der MSE, desto kleiner der PSNR Wert und umso größer der PSNR desto besser.

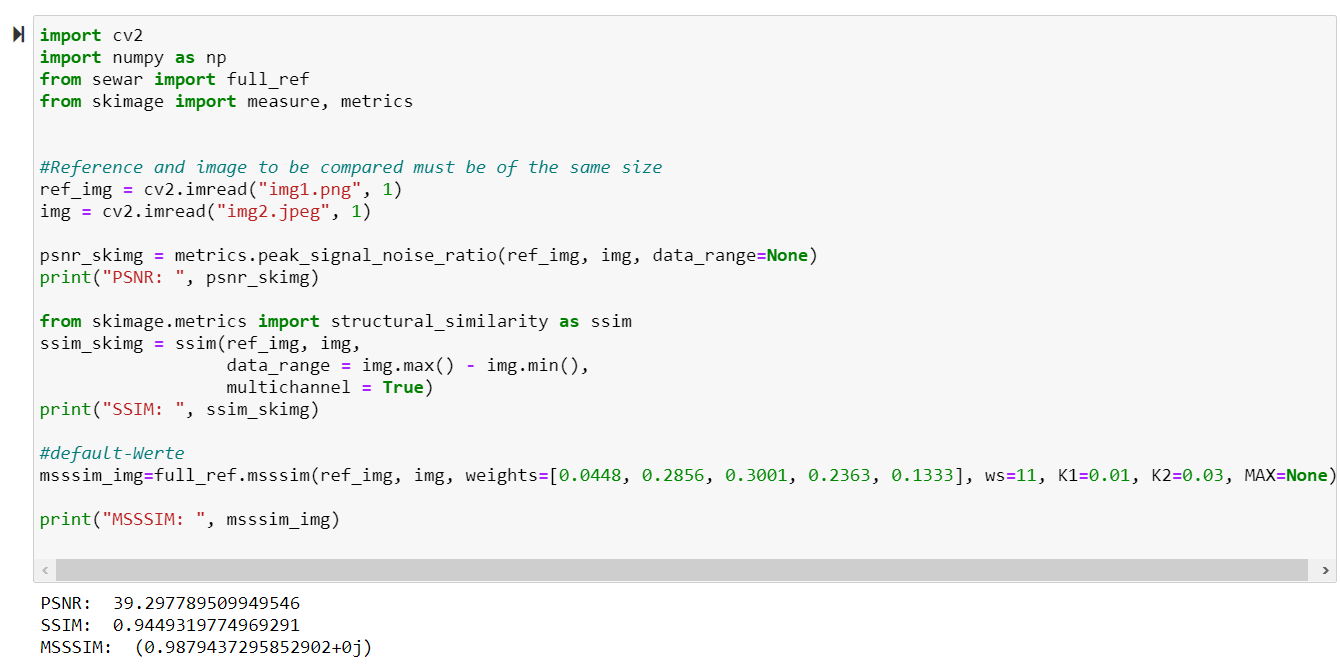
*Typische Werte sind hier zwischen 30 und 50dB und umso höher desto besser.*

• **Structural Similarity Index (SSIM**) Hier wird versucht, die Ähnlichkeit der Bildstruktur zu erfassen. Der Begriff „Struktur“ eines Bildes bezeichnet die Abhängigkeiten zwischen benachbarten Bildpunkten, die unabhängig von der Helligkeit und dem Kontrast in dieser Region des Bildes sind.

*Hier liegt die Maßzahl zwischen 0 (total unterschiedlich) und 1(identisch)*

• **Multi-scale Structural Similarity Index (MS-SSIM)** erweitert den SSIM-Index, indem sie Luminanz Informationen auf der höchsten Auflösungsebene mit Struktur- und Kontrastinformationen in mehreren heruntergerechneten Auflösungen bzw. Skalen kombiniert. Die verschiedenen Skalen berücksichtigen die Variabilität in der Wahrnehmung von Bilddetails, die durch Faktoren wie den Betrachtungsabstand zum Bild, den Abstand der Szene zum Sensor und die Auflösung des Bilderfassungssensors verursacht werden.

*Wenn der Wert näher bei 1 ist, ist die Qualität höher und Richtung 0 sinkt die Qualität. [[5]](#footnote-5)*

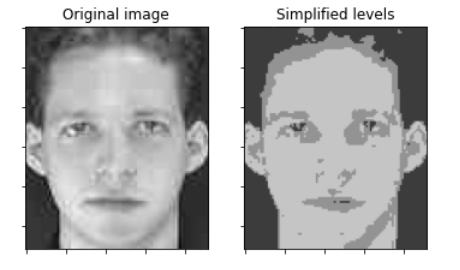


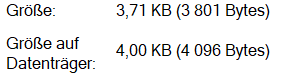
**Punkt 3:**

Bei der Lossy Compression werden die Daten eliminiert, die nicht auffallen. Dadurch kann eine kleinere Größe verwendet werden, da einige Teile des Fotos verworfen werden. Bei dieser Art der Komprimierung wird die Datenqualität beeinträchtigt und die Größe der Daten ändert sich.

Um einen Deep-learning Algorithmus anzuwenden, haben wir uns für die K-Means-Clustering Methode entschieden und diese wieder in Python implementiert und mit unseren Datensatz angewendet. Dieser ist eine zentrumsbasierte Clustering-Technik. Bei dieser Technik wird der Datensatz in k verschiedene Cluster geclustert[[6]](#footnote-6). Jedes Cluster im K-Means-Clustering-Algorithmus wird durch seinen Schwerpunktpunkt dargestellt. Der K-Means-Algorithmus versucht, ähnliche Daten in K-Gruppen oder Clustern zu bündeln. Wir müssen also alle Daten von Personen sammeln, deren Gesichter wir erkennen wollen, und sie in K-Clustern bündeln, so dass wir als Ergebnis K-Cluster haben, von denen jeder eine Person darstellt.[[7]](#footnote-7)

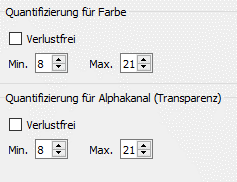
Die Cluster können hier angepasst werden, sodass eine Datengröße von 4KB erreicht werden kann.





**Punkt 4 AVIF:**

Hier wurde unser AT&T Dataset auf das Format AVIF umgewandelt. Für das Format AVIF konnten die 3,30 KB durch das Ausprobieren von den unten gezeigten Parametern erreicht werden:



Dabei bezieht sich der 2. Parameter auf die Transparenz und der 1. auf die Farb-Quantifizierung selbst.

1. <https://git-disl.github.io/GTDLBench/datasets/att_face_dataset/> [↑](#footnote-ref-1)
2. <https://towardsdatascience.com/face-recognition-using-deep-learning-b9be73689a23> [↑](#footnote-ref-2)
3. <https://www.kaggle.com/serkanpeldek/face-recognition-on-olivetti-dataset> [↑](#footnote-ref-3)
4. <https://sewar.readthedocs.io/en/latest/> [↑](#footnote-ref-4)
5. <https://de.mathworks.com/help/images/image-quality-metrics.html#:~:text=Full-Reference%20Quality%20Metrics,-Full-reference%20algorithms&text=Mean-squared%20error%20(MSE),the%20human%20perception%20of%20quality.&text=Peak%20signal-to-noise%20ratio%20(pSNR)>. [↑](#footnote-ref-5)
6. <https://towardsdatascience.com/image-compression-using-k-means-clustering-aa0c91bb0eeb> [↑](#footnote-ref-6)
7. <https://medium.com/analytics-vidhya/face-recognition-using-k-means-clustering-127c462e02f2> [↑](#footnote-ref-7)