

Punkt 0:

Um unsere Datensätze von .PNG zu .JPEG/.JPEG200/.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:

Bei allen Dateiformaten beträgt die Qualität noch 80% des Originals und hat eine Größe von 1024x1024.

JPEG:

* Datengröße nur noch ca. 10% des Originals

JPEG200

* Datengröße noch ca. 90% des Originals

JPRG XR:

* Datengröße noch ca. 15% des Originals

Punkt 1:

Die Gesichtserkennung mit Haar-Kaskaden ist ein auf maschinellem Lernen basierender Ansatz, bei dem eine Kaskadenfunktion mit einem Satz von Eingabedaten trainiert wird. OpenCV enthält bereits viele vorab trainierte Klassifikatoren für Gesicht, Augen, Lächeln usw.[[1]](#footnote-1)

Text

Description automatically generated

* Die Erkennung funktioniert nur bei Graustufenbildern. Daher ist es wichtig, das Farbbild in Graustufen umzuwandeln. (Zeile 13)
* Mit der Funktion detectMultiScale (Zeile 15) werden die Gesichter erkannt. Es werden 3 Argumente benötigt - das Eingabebild, scaleFactor und minNeighbours. scaleFactor gibt an, um wie viel die Bildgröße mit jeder Skalierung reduziert wird. minNeighbours gibt an, wie viele Nachbarn jedes Kandidatenrechteck haben muss, um es beizubehalten.
* „faces“ enthält eine Liste von Koordinaten für die rechteckigen Bereiche, in denen Gesichter gefunden wurden. Wir verwenden diese Koordinaten, um die Rechtecke in unserem Bild zu zeichnen.[[2]](#footnote-2)

A group of people posing for a photo

Description automatically generated

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. [[3]](#footnote-3)

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.

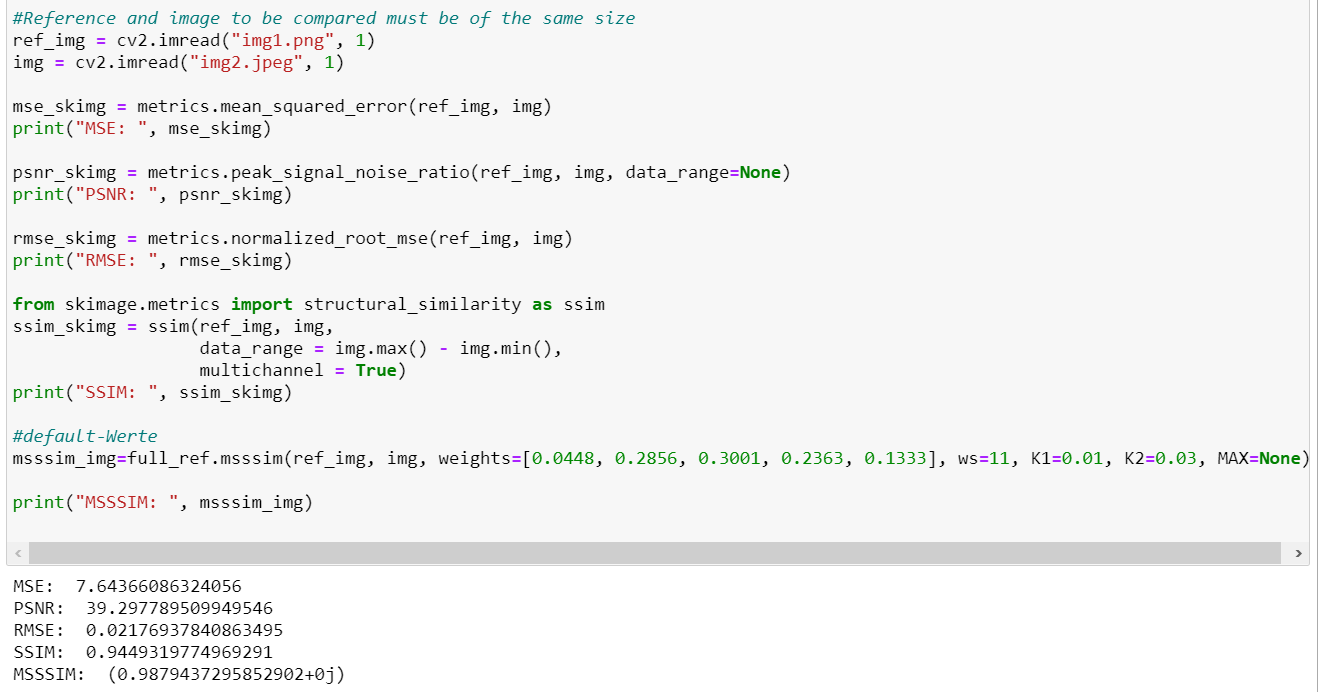
*Typische Werte sind hier zwischen 30 und 50dB und umso höher desto besser.*

* **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 Luminanzinformationen 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.[[4]](#footnote-4)*

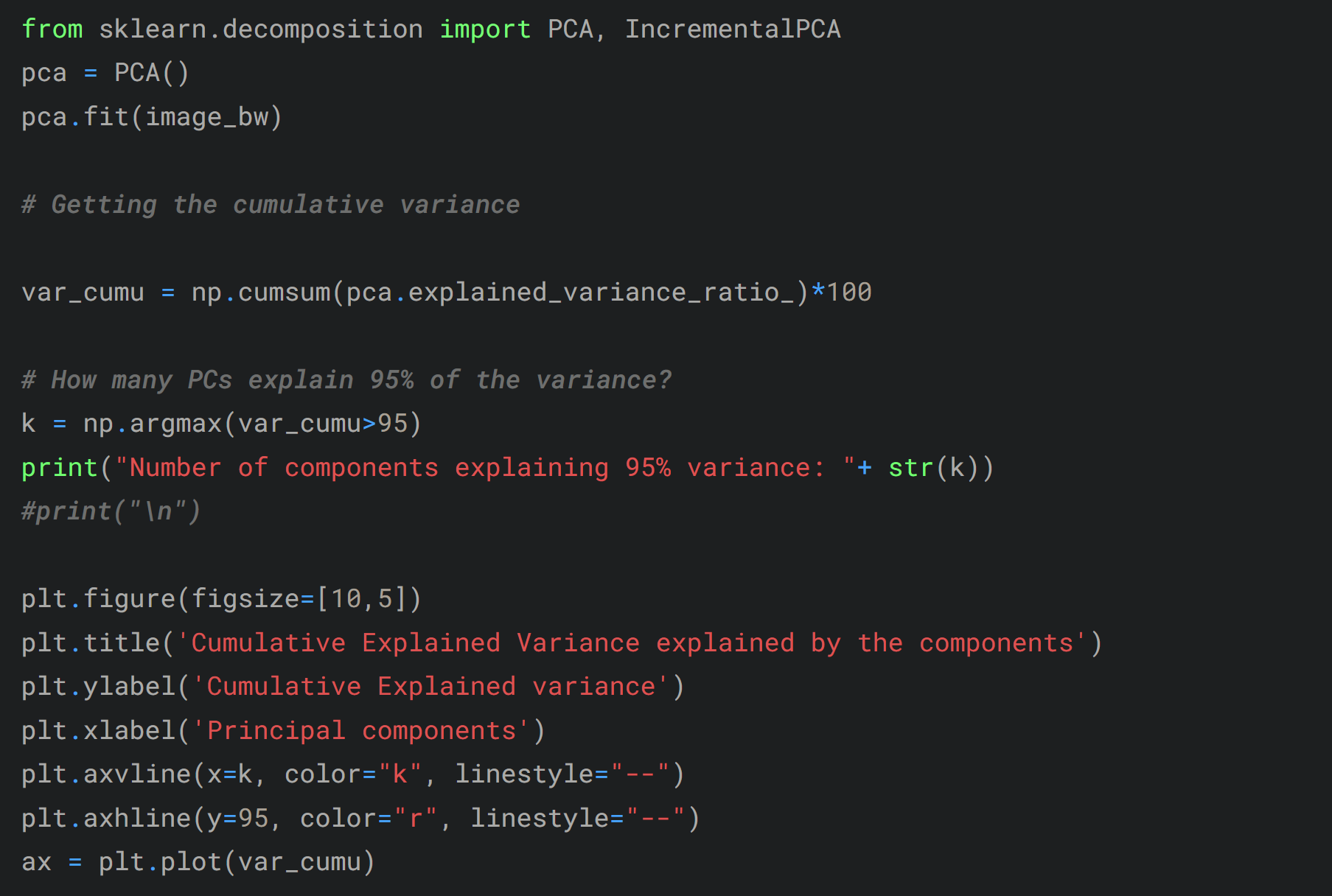


Punkt 3:

Auch hier wurde wieder Python verwendet, um eine Deep-Learning Methode für Lossy-Compression zu implementieren. Dazu haben wir die PCA Methode verwendet.

Im wesentlichen wurden hier folgendes durchgeführt:

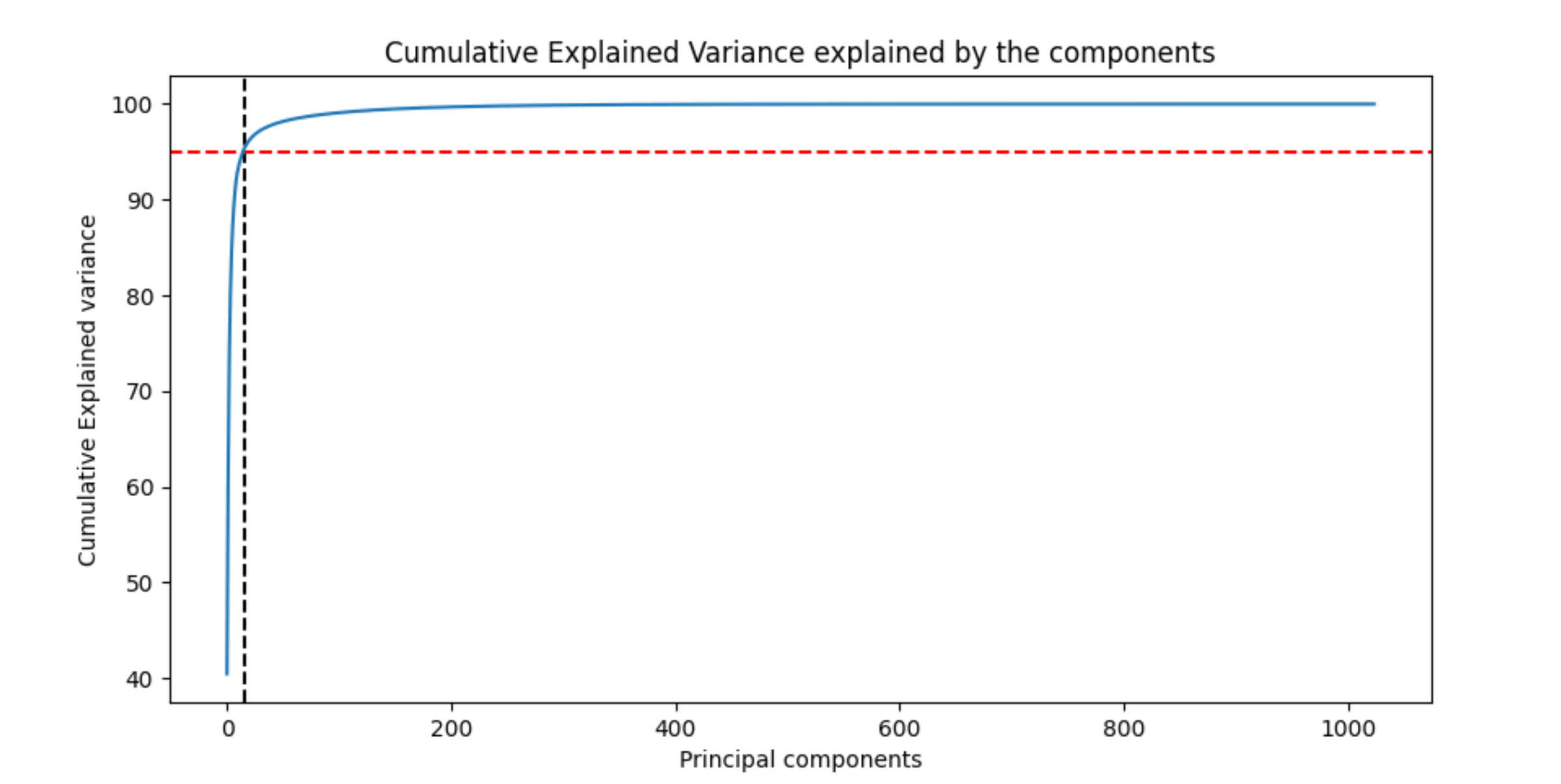
* PCA wurde in Python implementiert.
* Dann das wählen von der Anzahl der Komponenten mithilfe eines Scree Plots.
* Und schließlich das rekonstruieren der ursprünglichen Daten aus den Komponenten um einen Vergleich zum Original haben zu können.[[5]](#footnote-5)



Auszug des Python Programmes.

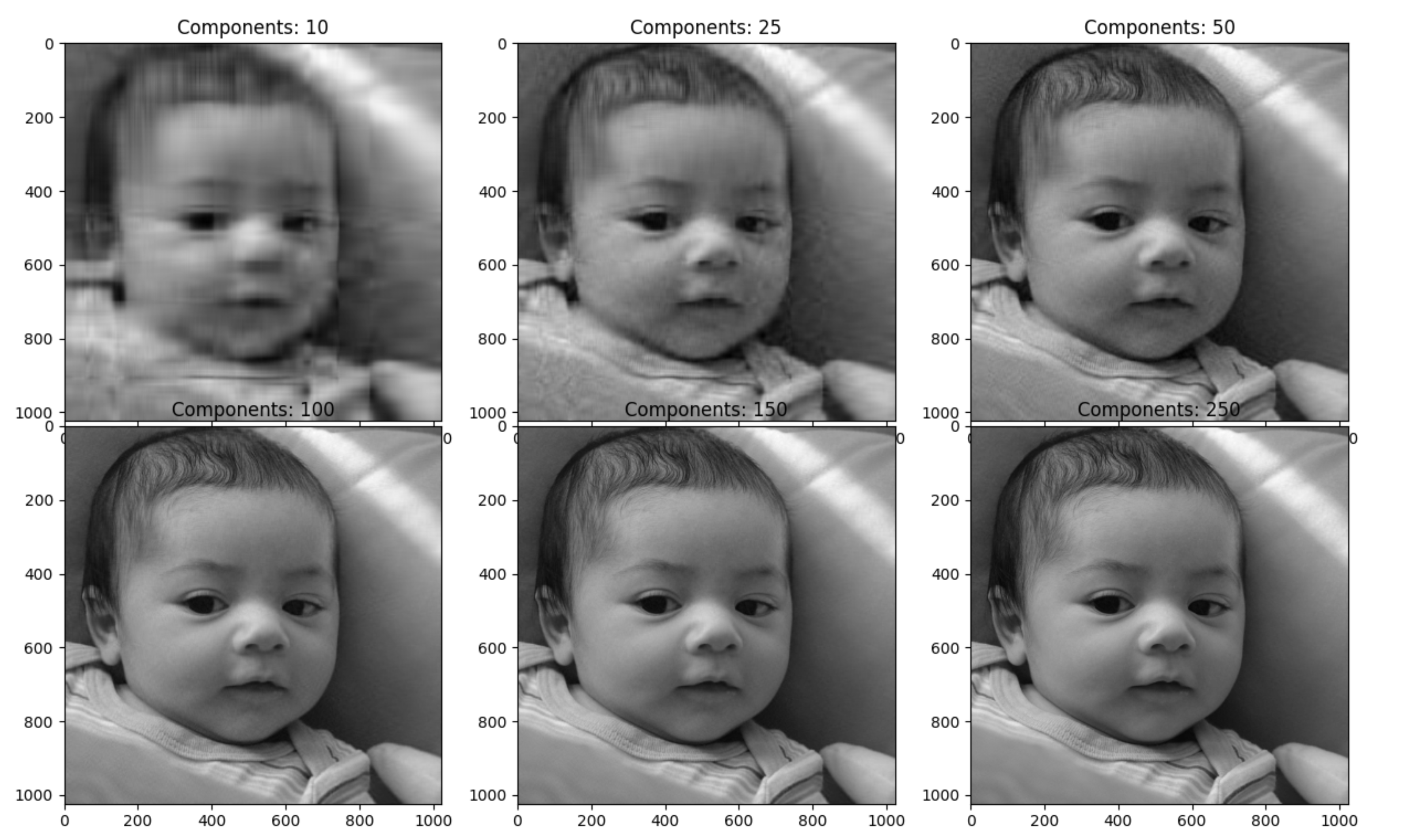
Der Output des Programms ergibt für das „bespielfoto.avif“ mit 1024x1024:

Numbe of components explaining 95% variance: 15



Es sieht so aus, als ob 15 Komponenten, statt 1024 Pixel, 95% der Varianz im Bild erklären können.

Allerdings ist in diesem Fall das rekonstruierte Bild im Detail noch sehr unscharf, weshalb unten unterschiedliche Komponenten ausprobiert wurden:



Bereits ab 50 Components würde man wahrscheinlich bei einer kleinen Auflösung schwer einen Unterschied zum Original feststellen können.

Punkt 4:

**AVIF:**

Um unsere Datensätze auch in diesem Bildformat verarbeiten zu können, haben wir unsere Datensätze wieder mit dem XnConverter in das entsprechende Bildformat umgewandelt und durch minimale Bearbeitung des Codes wieder in Python implementiert und die Funktionen darauf angewendet.

Die Datensätze sind in AVIF Format nur noch 66% der Originalgröße, also ca. um 1/3 verkleinert worden.

1. <https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml> [↑](#footnote-ref-1)
2. https://towardsdatascience.com/face-detection-in-2-minutes-using-opencv-python-90f89d7c0f81 [↑](#footnote-ref-2)
3. <https://sewar.readthedocs.io/en/latest/> [↑](#footnote-ref-3)
4. <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-4)
5. <https://www.kaggle.com/mirzarahim/introduction-to-pca-image-compression-example> [↑](#footnote-ref-5)