

Unterrichtsmaterialien zum Thema

Datenkompression

JAHRGANGSSTUFE 11-13

Material für SchülerInnen

Aufgaben

0. Vorbereitende Hausaufgabe:

Laden Sie sich die App „Columbus Eye“ im Google Play Store. Die App benötigt den Kamera- und Dateizugriff, um zu funktionieren. Es werden keine persönlichen Daten gesammelt. Öffnen Sie die App und laden Sie zusätzlich den Part „Datenkompression“ herunter.



1. Richten Sie die Kamera Ihres Smartphones auf den Erde-Marker und sehen Sie der ISS kurz zu, wie sie die Erde umfliegt.

- Laden Sie das Bilddatenpaket von der ISS auf die Erde herunter. Notieren Sie dabei stichpunktartig Ihre Beobachtungen.
- Tippen Sie anschließend auf den Refresh-Button und laden Sie danach noch einmal das Bilddatenpaket herunter: Sie erhalten nun zwei Kommunikations-Satelliten, EDRS-A & EDRS-C, zur Hilfe bei der Übertragung, die mit zwei Mal Tippen auf den EDRS-Button aktiviert werden können. Beschreiben Sie den Weg der Übertragung und die Veränderungen im Vergleich zu Aufgabenteil a).

Tipp: Sowohl die ISS als auch die Satellitenmodelle und die Bodenstationen können näher betrachtet werden, indem man sie auf dem Bildschirm antippt.

- Es gibt drei Bodenstationen für die Kommunikation mit den EDRS-Satelliten, welche sich alle in Europa befinden. Stellen Sie Vermutungen auf, warum nicht stattdessen beispielsweise eine Station in Europa, eine in Amerika und eine in Asien existiert.
- Das Bilddatenpaket ist 2,5 GB groß. Das EDRS kann bis zu 1,8 Gbit/s übertragen. Berechnen Sie die benötigte Übertragungszeit über das EDRS.
- Diskutieren Sie, wie die Übertragung des Bilddatenpakets noch weiter beschleunigt werden kann.

2. Finden Sie sich in Kleingruppen von 3-4 Personen zusammen und lesen Sie den Text „Satellitenbilddaten“ auf Seite 5.

Erarbeiten Sie in Kleingruppen jeweils **einen** der Kompressionsmodi Farbreduktion (Seite 6), Redundanz-/ Ähnlichkeitssuche (Seite 7) oder Farbunterabtastung (Seite 8).

- Erläutern Sie Ihr Verfahren, indem Sie den entsprechenden Text lesen und die App mit dem Marker 1 auf Seite 5 (Ätna mit Lupe) benutzen. Schalten Sie, sobald das UI mit den Kompressionsverfahren erscheint, auf das entsprechende Verfahren Ihrer Gruppe.
- Beurteilen Sie, inwiefern das Verfahren für das Satellitenbild vom Ätna auf Sizilien (Marker 1, Seite 5) mit seinen Anwendungsbereichen geeignet ist.
- Präsentieren Sie anschließend die Ergebnisse der Klasse.

3. Eine Möglichkeit zur Farbreduktion ist der Median Cut. Implementieren Sie diesen mit Hilfe des Textes auf Seite 9: In der Datei **Median_Cut_SuS.py** ist ein Grundgerüst bereits implementiert: Input, Übertragung der Pixel in die Liste *img_arr* und Übertragung der neuen Pixelwerte sind bereits vorhanden. Vervollständigen Sie die Funktion *split_into_buckets* anhand der folgenden Anleitung:

- a) Schauen Sie sich zunächst die Inputparameter der Funktion „*split_into_buckets*“ an. Geben Sie an, in welchem Datenformat der Parameter „*img_arr*“ übergeben wird.
- b) Die zu entwickelnde Funktion muss zuerst bestimmen, welcher Farbraum (Rot, Grün, Blau) die höchste Spannweite zwischen dem Minimal- und Maximalwert hat. Hierzu kann die externe Bibliothek *numpy* genutzt werden, die bereits importiert ist. Die Funktionen *np.max()* und *np.min()* geben die notwendigen Werte zurück, um die Spannweite berechnen zu können.

Wenn die Farbraum-Spannweiten bestimmt sind, muss der Wert von *space_with_highest_range* entsprechend gesetzt werden: 0 falls Rot, 1 falls Grün, 2 falls Blau die höchste Spannweite aufweist.

Implementieren Sie dies in der Datei **Median_Cut_SuS.py**.

- c) Die Funktion *split_into_buckets* muss nun zwei Mal rekursiv aufgerufen werden: Beim ersten Mal muss eine neue Liste mit den Werten **bis** zum Index und beim zweiten Mal **ab** dem Index an die Funktion übergeben werden. Implementieren Sie diese Rekursion in Ihrem Code.
- d) Ein rekursiver Algorithmus kann sich bis in alle Ewigkeit immer wieder selbst aufrufen, sofern es keine Stoppbedingung gibt. In dieser Funktion muss es sogar zwei geben: Falls *img_arr* leer ist, ist die Funktion zu beenden und falls *depth = 0* ist, muss *median_cut_quantize* mit dem aktuellen *img_arr* aufgerufen und die Funktion dann beendet werden. Implementieren Sie die Stoppbedingung.

- e) Testen Sie Ihr Programm, in dem Sie den Befehl

```
python MedianCutSample_SuS.py -c {Anzahl Farben} -i {Pfad/zur/Input/Bilddatei} -o {Pfad/zur/Output/Bilddatei}
```

in die Konsole eintippen. Sie müssen sich im Verzeichnis der MedianCutSample_SuS Datei befinden. Um das Bild auf z.B. 8 Farben zu reduzieren, müssen sie aber tatsächlich „Anzahl Farben“ = 3 setzen, denn $2^3 = 8$. Für eine Reduktion auf 4 Farben z.B. „Anzahl Farben“ = 2, da $2^2 = 4$. Die Output-Datei wird durch diesen Befehl neu erzeugt.

4. Vergleichen Sie den implementierten Code mit dem Code auf der folgenden Seite 3, indem Sie...

- a) ... den nachfolgenden Code zunächst beschreiben.
- b) ... das grundlegende Prinzip nennen, das die beiden Algorithmen unterscheidet.

5. Nehmen Sie Stellung zu der folgenden Aussage: „Es gibt kein perfektes Kompressionsverfahren.“

Alternative Implementierung

```

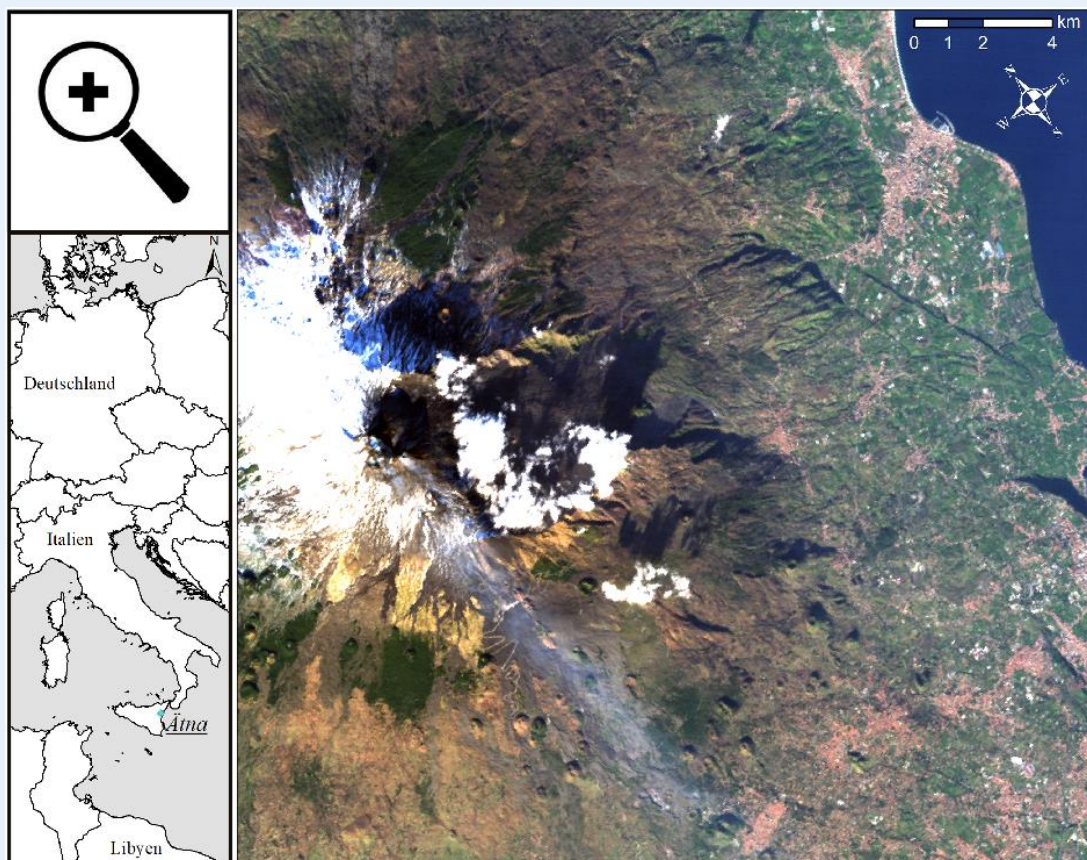
1 def split_into_buckets_alternative(img, img_arr, depth):
2     if depth == 0:
3         print("Das Bild wäre einfarbig")
4         return
5
6     ### 2 Platzhalter werden definiert. Diese werden zu Arrays und halten den aktuellen Stand der "Buckets"
7     oldBuckets = [0]
8     newBuckets = [0]
9     newBuckets[0] = img_arr
10
11     for currentDepth in range(depth):
12         ### Neuberechnung von Kontrollvariablen
13         oldBuckets = newBuckets
14         currentNumberOfBuckets = 2**(currentDepth+1)
15
16         ### Buckets neu sortieren
17         for index, bucket in enumerate(newBuckets):
18             r_range = np.max(bucket[:, 0]) - np.min(bucket[:, 0])
19             g_range = np.max(bucket[:, 1]) - np.min(bucket[:, 1])
20             b_range = np.max(bucket[:, 2]) - np.min(bucket[:, 2])
21
22             space_with_highest_range = 0
23
24             if g_range >= r_range and g_range >= b_range:
25                 space_with_highest_range = 1
26             elif b_range >= r_range and b_range >= g_range:
27                 space_with_highest_range = 2
28             elif r_range >= b_range and r_range >= g_range:
29                 space_with_highest_range = 0
30
31             # Sortiert das img_arr von niedrigen zu hohen Werten im Bezug auf
32             # den zuvor gefundenen Farbraum (R,G oder B) mit der größten Spanne an Werten.
33             # space_with_highest_range = {0,1 oder 2}
34             newBuckets[index] = bucket[bucket[:, space_with_highest_range].argsort()]
35
36         ### Buckets trennen
37         newBuckets = [0] * currentNumberOfBuckets
38         for bucketNumber in range(currentNumberOfBuckets):
39             #Den Trennindex berechnen
40             #Anfang-Mitte und Mitte bis Ende, des jeweiligen Buckets
41             if bucketNumber%2 == 0:
42                 old_index = 0
43                 new_index = int((len(oldBuckets[int(bucketNumber/2)]) + 1) / 2 )
44             else:
45                 old_index = new_index
46                 new_index = len(oldBuckets[int(bucketNumber/2)])
47             #Den Bucket belegen
48             newBuckets[bucketNumber] = oldBuckets[int(bucketNumber/2)][old_index:new_index]
49
50
51     for bucket in newBuckets:
52         median_cut_quantize(img, bucket)
53

```

Satellitenbilddaten

In den 1960ern entwickelte die amerikanische Raumfahrtorganisation NASA maßgeblich die ersten Digitalkameras mit, um für den Flug zum Mond und die Erdbeobachtung mit Satelliten nicht auf Filmrollen angewiesen zu sein. Die „Sensoren“ genannten Kameras nehmen die Erde von ihrem Orbit aus auf und senden die Daten zu Bodenstationen, wo sie weiterverarbeitet und für die verschiedensten Zwecke bereitgestellt werden. Schon damals waren die Bilder hunderte Megabyte groß. Deshalb wurden nur wenige Bilder aufgenommen und auch nur auf Bestellung - Speicherplatz war stark begrenzt. Heute sieht das anders aus: Ein stetiger Strom an Erdbeobachtungsdaten, generiert von dutzenden Satelliten auf verschiedenen Orbits, erreicht die Erde in riesigen Datenzentren, wo die Daten für Nutzer aus Wissenschaft und Forschung, Katastrophenmanagement, Sicherheit und Gewerbe zur Verfügung stehen.

Je nach Art der Bilder erreichen diese eine Dateigröße von mehreren Gigabyte – zum Beispiel bei Hyperspektraldaten, die nicht nur in den Kanälen Rot, Grün und Blau Bilder der Erde aufnehmen, sondern in hunderten Kanälen entlang des elektromagnetischen Spektrums. Der Sensor DESIS, der sich an Bord der ISS befindet, nimmt vom ultravioletten bis infraroten Bereich dieses Spektrums insgesamt 235 Kanäle pro Bild auf. Jedes dieser Bilder ist 1024 Pixel breit und hoch und verfügt über eine Bittiefe von 16 Bit pro Pixel. In einem Download von der ISS zur Bodenstation müssen bis zu 100 Bilder übertragen werden. Mithilfe verschiedener Bildkompressionsverfahren wäre es möglich, den Speicherverbrauch dieser Bilder zu reduzieren, ohne dass die Qualität der Bilder sich stark verschlechtert.

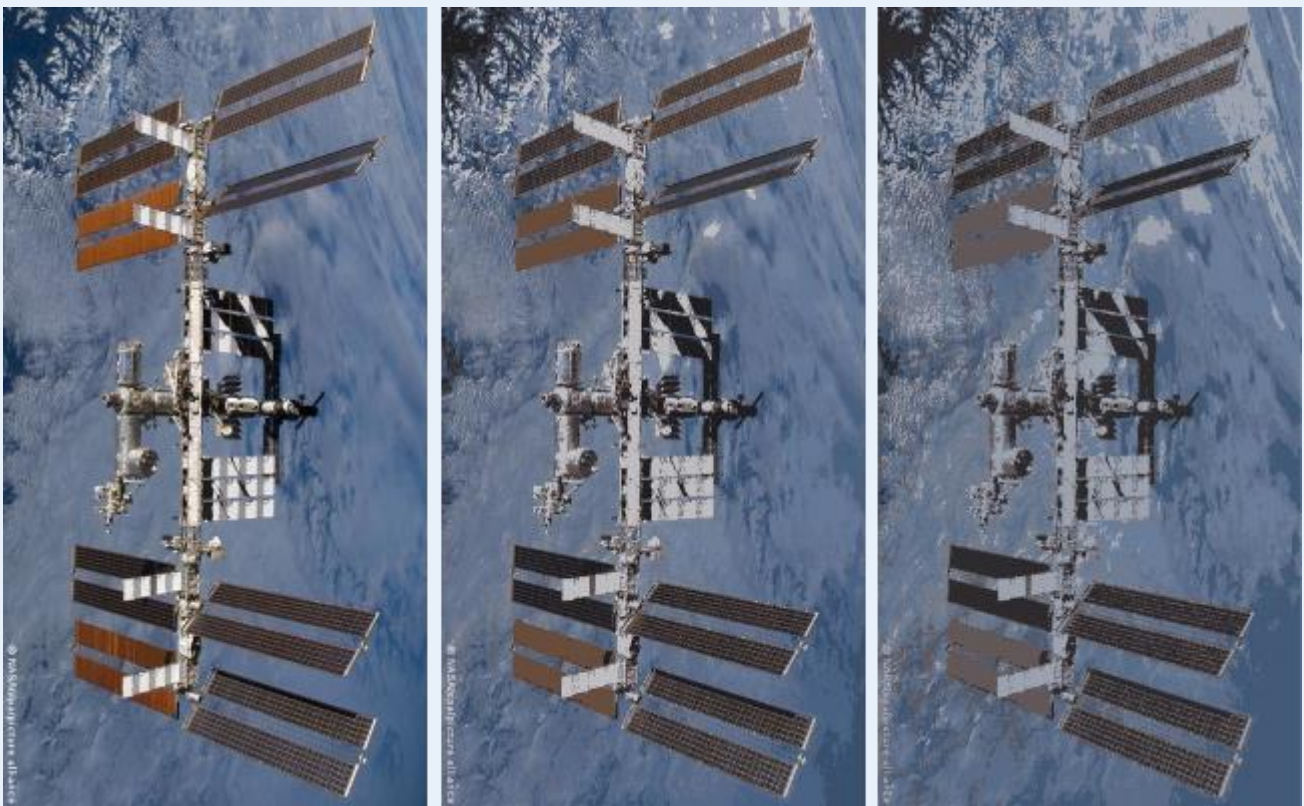


Marker 1: Ätna mit Lupe DESIS-Bild des Ätna vom 15.02.2020 um 15:34 Uhr. Wird das Lupensymbol angetippt, während die Kamera in der „Columbus Eye“-App (Part Datenkompression) auf das Bild gehalten wird, kann eine Lupenfunktion genutzt werden.

Farbreduktion

Eine Form der Bildkompression ist die der Farbreduktion bzw. Farbquantisierung. Hierbei wird der Farbraum, d.h. die Anzahl der möglichen Farben, eingeschränkt. Dies wird realisiert, indem jedem Pixel nur noch eine geringere Anzahl an Bits pro Farbkanal zur Verfügung stehen, z.B. statt 8 Bit pro Kanal nur noch 4 Bit. Bildformate wie das GIF-Format unterstützen einen Farbraum von nur 256 Farben. Der Speicherverbrauch kann hierbei folglich stark reduziert werden. Gleichzeitig wird jedoch auch die Bildqualität stark reduziert.

Die Farbreduktion ist ein verlustbehaftetes Kompressionsverfahren. Hierbei unterscheidet sich das Originalbild von dem komprimierten Bild, es gehen also Informationen verloren. Außerdem kann das Originalbild aus dem komprimierten Bild nicht wiederhergestellt werden.



Marker 2: Die ISS im reduzierten Farbraum Links: Original mit 16,7 Millionen Farben, Mitte: 16 Farben, Rechts: 8 Farben

Redundanzsuche & Ähnlichkeitssuche

Eine Art der Bildkompression ist die Redundanzsuche bzw. Redundanzreduktion. Hierunter werden verlustfreie Kompressionsverfahren¹ verstanden, bei denen der Speicherverbrauch gesenkt wird, indem Redundanzen² entfernt werden. Dadurch wird Speicherplatz gespart, da beispielsweise 30 Pixel mit demselben RGB-Wert durch ein einzelnes Symbol repräsentiert werden können. Außerdem können häufig auftretende Farbwerte durch kurze Symbole ersetzt werden (z.B. "0", "01", etc.). Selten auftretende Farbwerte werden durch lange Symbole repräsentiert (z.B. "00100001"). Dadurch kann die Bitlänge der gesamten Datei reduziert werden.

Neben dieser Redundanzreduktion kann zusätzlich (vorher) noch eine Ähnlichkeitssuche durchgeführt werden. Dabei wird jeder Pixel darauf überprüft, ob ein umliegender Pixel einen ähnlichen Farbwert besitzt. Wann der Farbwert eines Pixels als ähnlich genug gilt und übernommen wird, legt der Grenzwert fest. Dieser ist in der App in Prozent angegeben.

Durch die Ausführung einer Ähnlichkeitssuche kann die Anzahl der genutzten Farben im Bild sinken. Dies ist vorteilhaft in Kombination mit einer Redundanzreduktion, da Pixel mit denselben Farbwerten zusammengefasst werden können. Dadurch ist im Vergleich zur normalen Wörterbuch-Kodierung der Speicherverbrauch niedriger, jedoch auf Kosten der Bildqualität. Das kombinierte Verfahren ist verlustbehaftet. Daher unterscheidet sich das komprimierte Bild visuell vom unkomprimierten Bild.

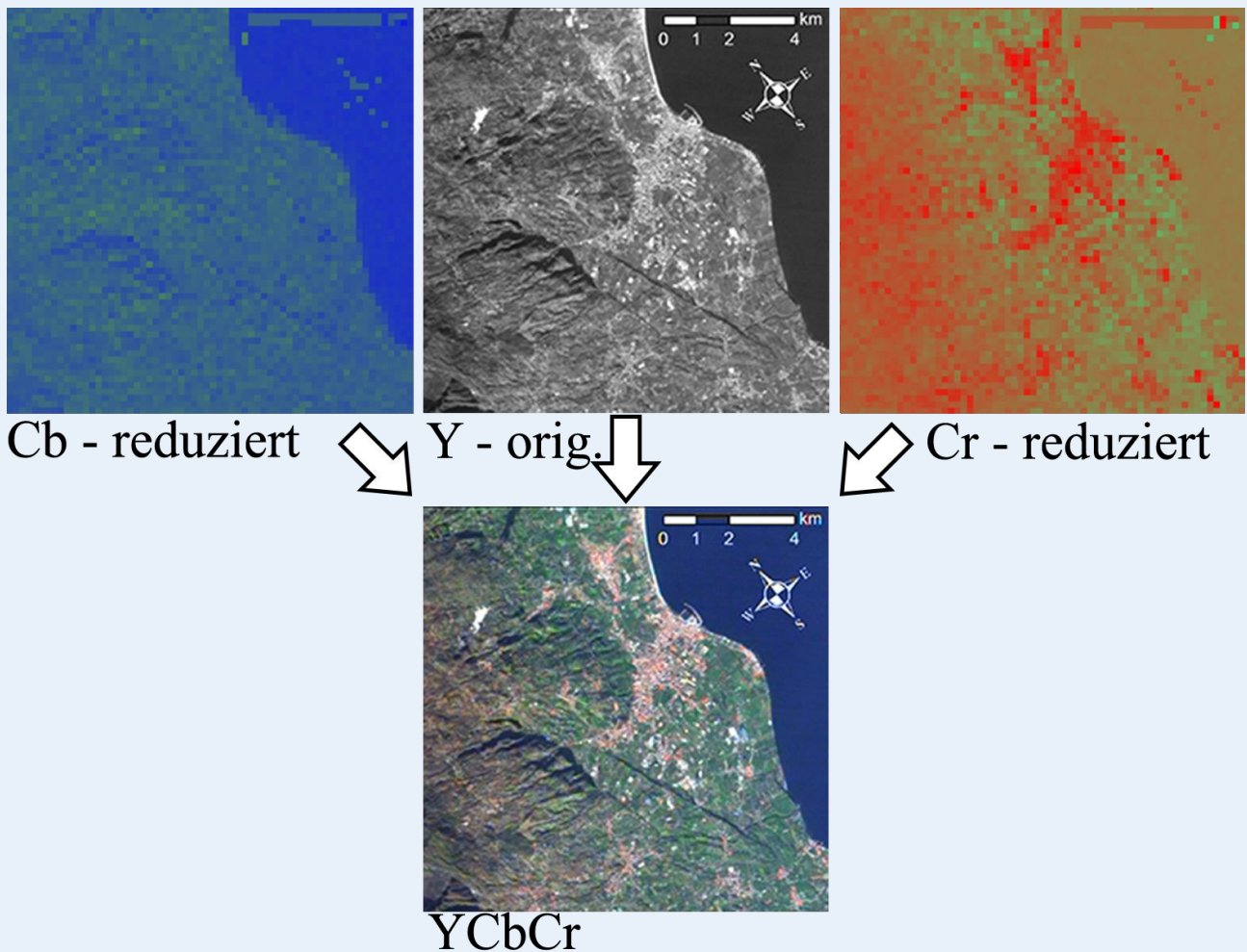
¹ Verlustfreie Kompressionsverfahren sind Verfahren, bei denen keine Informationen verloren gehen. Das Originalbild ist also visuell identisch mit dem unkomprimierten Bild.

² Redundanz beschreibt einen Überfluss an Informationen bzw. deren mehrfaches Vorhandensein.

Farbunterabtastung

Eine Bildkompressionsmethode ist die der Farbunterabtastung (besser bekannt unter dem englischen Namen Chroma Subsampling oder Color Subsampling). Hierbei wird die Tatsache ausgenutzt, dass das menschliche Auge Farbe mit einer geringeren Auflösung wahrnimmt als Helligkeit. Daher erkennen wir Helligkeitsübergänge deutlich besser als Farbübergänge.

Das Bild muss dafür in einen Farbraum konvertiert werden, welcher die Helligkeit separat von der Farbdarstellung speichert, beispielsweise der YCbCr-Farbraum. Hier werden Farben über den Helligkeitskanal Y und die Farbkanäle Cb (Blue-Chrominance) und Cr (Red-Chrominance) dargestellt. In der folgenden Abbildung wird das Originalbild mithilfe des YCbCr-Farbraums in die einzelnen Kanäle Y, Cb und Cr aufgeteilt. Der Y-Kanal ist in Original-Auflösung erhalten (oben Mitte), die beiden Farbkanäle Cb (oben links) und Cr (oben rechts) sind jedoch stark reduziert. Werden sie wieder zusammengesetzt (unten), ergibt sich ein nur leicht verändertes Bild.



Marker 3: Ätna in YCbCr DESIS-Bild des Ätna vom 15.02.2020 um 15:34 Uhr, zerlegt in seine Komponenten Cb: Blue-Chrominance, Y: Helligkeit, Cr: Red-Chrominance, sowie wieder zusammengesetzt. Cb und Cr enthalten nur 1/4 so viele Bildpunkte, wie Y. (Falls in schwarz-weiß gedruckt: Die Abbildung ist in Farbe in der App zu sehen, wenn die Kamera darauf gehalten wird)

Median Cut Algorithmus

Im Median Cut Algorithmus, welcher eine von vielen verschiedenen Farbreduktions-Implementierungen ist, werden zuerst alle einzelnen Pixel des Bildes in einer sehr langen Liste aneinandergereiht (img_arr). Diese Liste (oder „Bucket“ im Englischen) enthält für jeden Pixel die Werte [R , G , B , n , m] mit {n,m} als Koordinatenpunkte des Pixels im Bild.

R	G	B	N	M
120	10	25	0	0
124	12	22	0	1
123	11	28	0	2
...
32	222	231	1920	1079
34	221	229	1920	1080

Um den Farbraum zu verkleinern (von RGB, 24-Bit, 16.7 Millionen Farben auf zum Beispiel 8-Bit, 256 Farben) muss dieses große Bucket in 256 kleine Buckets geteilt werden. Danach wird jedem so entstandenen Bucket eine einzelne Farbe zugeteilt (die Durchschnittsfarbe aller in ihm enthaltenen Pixeln).

Um von einem großen Bucket auf z.B. 256 kleine Buckets zu kommen, werden 8-mal rekursiv alle bereits vorhandenen Buckets in jeweils 2 Buckets aufgeteilt.