

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 oder im Apple App Store herunter. Die App benötigt den Kamera- und Dateizugriff, um zu funktionieren. Es werden keine persönlichen Daten gesammelt. Öffnen Sie die App und lade dir zusätzlich den Part „Datenkompression“ herunter.

In diesem Arbeitsblatt sollen alle als **Marker** bezeichneten Abbildung in der App verwendet werden.



### 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 Farbrunterabtastung (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.java** ist ein Grundgerüst (main Funktion) bereits implementiert: Input, Übertragung der Pixel in die Liste *flattenedImgArray* und Übertragung der neuen Pixelwerte sind bereits vorhanden. Vervollständigen Sie die Funktion *MedianCutAlgorithm* anhand der folgenden Anleitung:

- a) Schauen Sie sich zunächst die Inputparameter der Funktion „MedianCutAlgorithm“ an. Geben Sie an, in welchem Datenformat der Parameter „inputListe“ übergeben wird.
- b) Öffnen Sie anschließend im Internet die Dokumentation zu diesem Datenformat, da Sie damit nachfolgend arbeiten werden.

Für die spätere Berechnung des Durchschnittswertes wird die Summe der R, G und B Farbräume benötigt. Die vorliegende For-Schleife iteriert über jeden Wert innerhalb der inputListe. Speichern Sie in jedem Iterationsschritt die entsprechenden Farbwerte in den Listen „roterKanal“, „grünerKanal“ und „blauerKanal“. Addieren sie diese Werte ebenfalls zu ihren jeweiligen „sumOfRed“, „sumOfGreen“ und „sumOfBlue“ Datenstrukturen.

Die *inputListe* wird nun mit der Hilfe eines selbstgeschriebenen Vergleichsoperators (<, =, >) sortiert. Dazu entscheidet die If-Verzweigung, welcher Farbkanal die höchste Spannweite an Werten hat und sortiert die Datenstruktur anhand dieses Farbkanals.

- c) Trennen Sie nun die *inputListe* in der Mitte und speichern diese in zwei neue temporäre Listen ab, indem Sie die ArrayList Funktionen „subList()“ und „size()“ benutzen.
- d) Rufen Sie die Funktion „MedianCutAlgorithm“ nun zweimal rekursiv auf - jeweils mit einer der beiden gerade erstellten Listen. Speichern sie die Rückgabewerte in der „outputListe“, indem Sie die ArrayList Funktion „addAll()“ benutzen.
- e) Ein rekursiver Algorithmus benötigt immer eine Stoppbedingung, da er sich ansonsten unendlich oft selbst aufrufen würde. In diesem Fall soll die Funktion beendet werden, wenn die übergebenen „iterationSteps“ gleich 0 sind. Implementieren Sie die Stoppbedingung.
- f) Nun muss eine For-Schleife für jeden Pixel der *outputListe*, die R, G und B Werte auf den Durchschnittswert des jeweiligen Farbraums setzen. Orientieren Sie sich an der For-Schleife aus Aufgabenteil b) und nutzen Sie sowohl die von Ihnen selbst berechneten Summen als auch die ArrayList Funktion „size()“. Geben Sie abschließend die *outputListe* als Ergebnis zurück.
- g) Testen Sie ihren Code, indem sie ein beliebiges JPG Bild in den Ordner „images“ ziehen, im Code den Dateipfad und Namen anpassen und die *main* Funktion ausführen lassen. Es wird ein result.jpg Bild entstehen, welches farbreduziert ist.

4. Vergleichen Sie den implementierten Code mit dem Code auf der folgenden Seite 4, 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 public static ArrayList<int[]> MedianCutAlgorithmAlternativ(ArrayList<int[]> inputListe, int iterationSteps){
2     ArrayList<ArrayList<int[]>> oldBuckets = new ArrayList<ArrayList<int[]>>(1);
3     ArrayList<ArrayList<int[]>> newBuckets = new ArrayList<ArrayList<int[]>>(1);
4     oldBuckets.add(inputListe);
5
6     if (iterationSteps == 0){} // Nichts tun!
7     else{
8         for(int step = 1; step <= iterationSteps; step++){
9             for(int bucket = 0; bucket < Math.pow(2, step-1); bucket++){
10                 ArrayList<Integer> rK = new ArrayList<Integer>(oldBuckets.get(bucket).size()); // rK = roter Kanal
11                 ArrayList<Integer> gK = new ArrayList<Integer>(oldBuckets.get(bucket).size()); // gK = grüner Kanal
12                 ArrayList<Integer> bK = new ArrayList<Integer>(oldBuckets.get(bucket).size()); // bK = blauer Kanal
13
14                 for (int[] item : oldBuckets.get(bucket)) {
15                     rK.add(item[0]);
16                     gK.add(item[1]);
17                     bK.add(item[2]);
18                 }
19
20                 // Den selbst geschriebenen Comparator erzeugen und die flattenedImgArray ArrayList
21                 // damit sortieren. Mit .set(1) wird die Liste nach ihrem zweiten Parameter sortiert.
22                 // Das wäre "Grün" im R,G,B,Width,Height
23                 customComparator customComp = new customComparator();
24
25                 if(max(rK) - min(rK) > max(gK) - min(gK)
26                    && max(rK) - min(rK) > max(bK) - min(bK))
27                     customComp.set(0);
28                 else if(max(gK) - min(gK) > max(rK) - min(rK)
29                    && max(gK) - min(gK) > max(bK) - min(bK))
30                     customComp.set(1);
31                 else if(max(bK) - min(bK) > max(gK) - min(gK)
32                    && max(bK) - min(bK) > max(rK) - min(rK))
33                     customComp.set(2);
34
35                 Collections.sort(oldBuckets.get(bucket), customComp);
36
37                 // Arrays in den korrekten Farben speichern
38                 newBuckets.add(new ArrayList<>(oldBuckets.get(bucket).subList
39                     (0, oldBuckets.get(bucket).size()/2)));
40                 newBuckets.add(new ArrayList<>(oldBuckets.get(bucket).subList
41                     (oldBuckets.get(bucket).size()/2, oldBuckets.get(bucket).size())));
42             }
43             oldBuckets = newBuckets;
44             newBuckets = new ArrayList<ArrayList<int[]>>(1);
45         }
46     }
47
48     for (ArrayList<int[]> finishedArrayList : oldBuckets) {
49         int sumOfRed = 0; int sumOfGreen = 0; int sumOfBlue = 0;
50
51         for (int[] item : finishedArrayList) {
52             sumOfRed += item[0];
53             sumOfGreen += item[1];
54             sumOfBlue += item[2];
55         }
56         for (int[] item : finishedArrayList) {
57             item[0] = sumOfRed / finishedArrayList.size();
58             item[1] = sumOfGreen / finishedArrayList.size();
59             item[2] = sumOfBlue / finishedArrayList.size();
60         }
61     }
62
63     // Temporäre Arrays wieder zu einem einzelnen Rückgabearray zusammenführen.
64     ArrayList<int[]> outputListe = new ArrayList<int[]>();
65     for (ArrayList<int[]> tempList : oldBuckets) {
66         outputListe.addAll(tempList);
67     }
68     return outputListe;
69 }

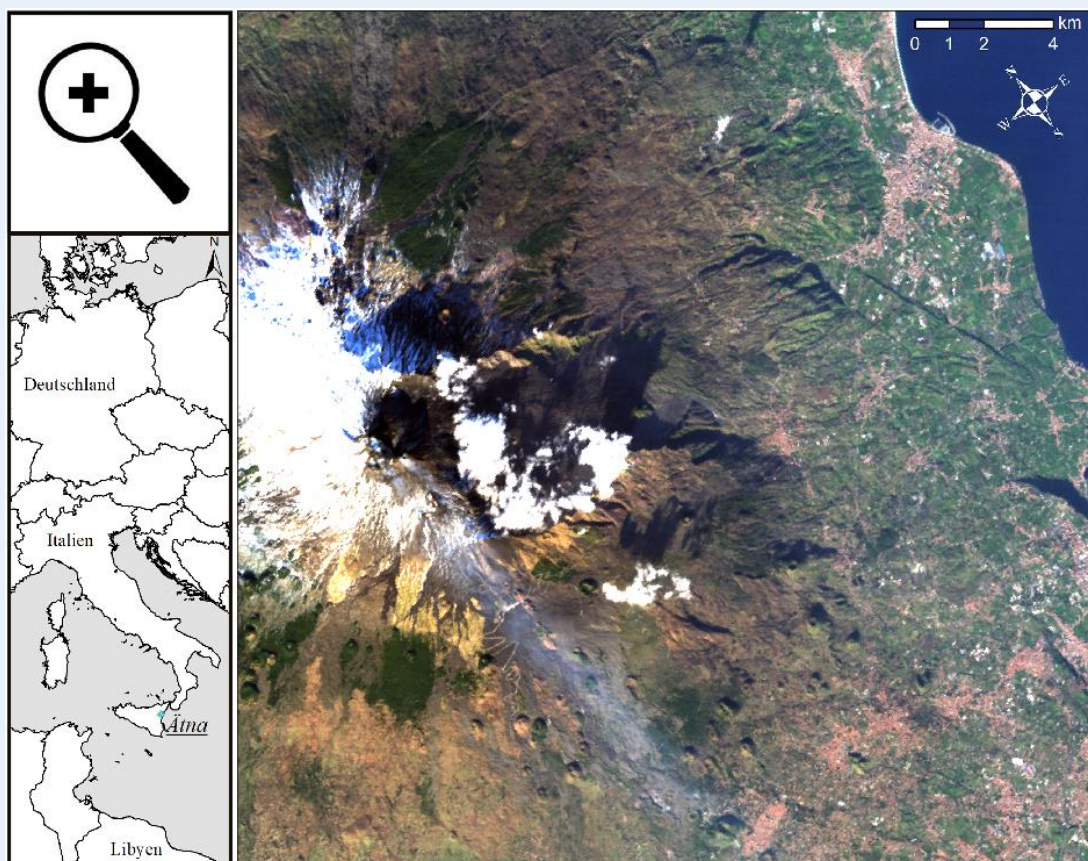
```



## 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.

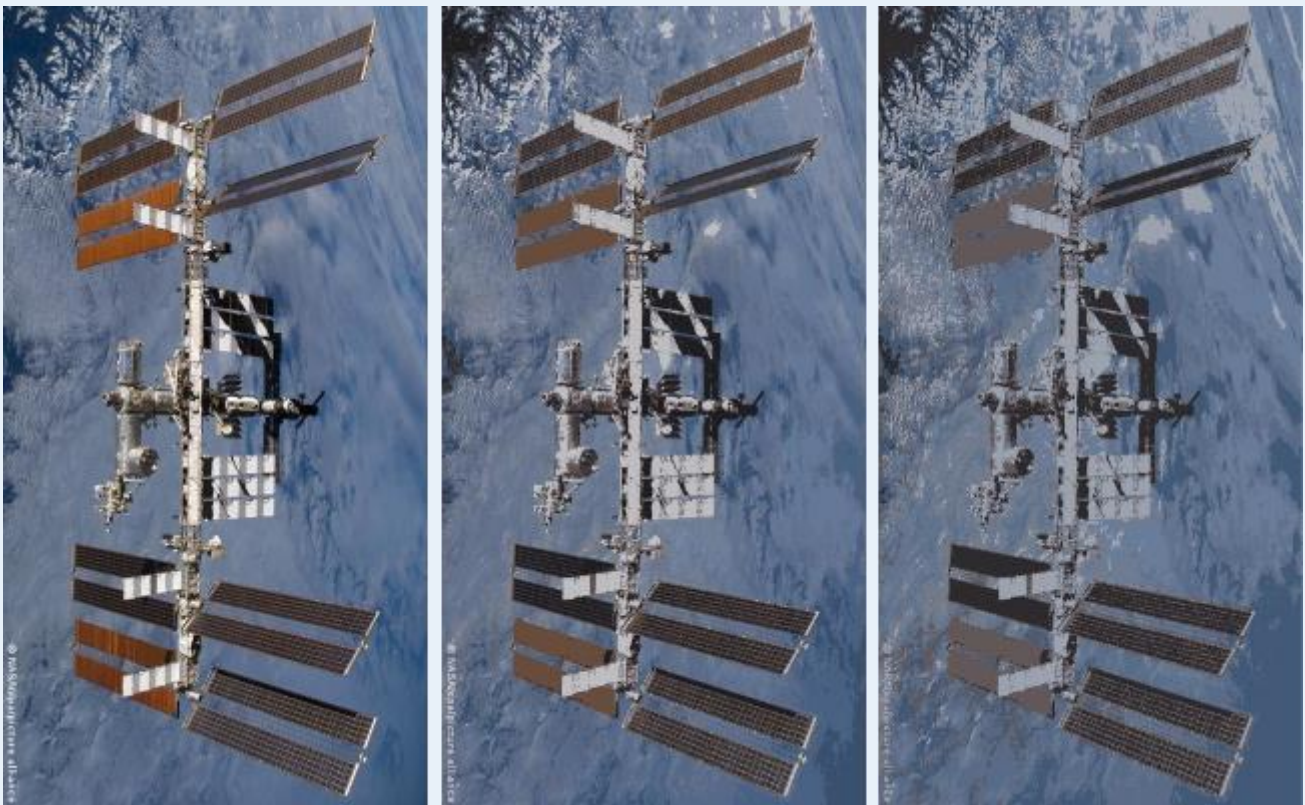


**Marker 1: Ätna mit Lupe** DESIS-Bild des Ätna vom 15.02.2020 um 15:34 Uhr. Wird das Lupensymbol angetippt, während die Kamera in der „Columbus Eye“-App (Part Datenkompression) auf das Bild gehalten wird, kann eine Lupenfunktion genutzt werden.

## 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 (Falls in schwarz-weiß gedruckt: Die Abbildung ist in Farbe in der App zu sehen, wenn die Kamera darauf gehalten wird)

## Redundanzsuche & Ähnlichkeitssuche

Eine Art der Bildkompression ist die Redundanzsuche bzw. Redundanzreduktion. Hierunter werden verlustfreie Kompressionsverfahren<sup>1</sup> verstanden, bei denen der Speicherverbrauch gesenkt wird, indem Redundanzen<sup>2</sup> 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.

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

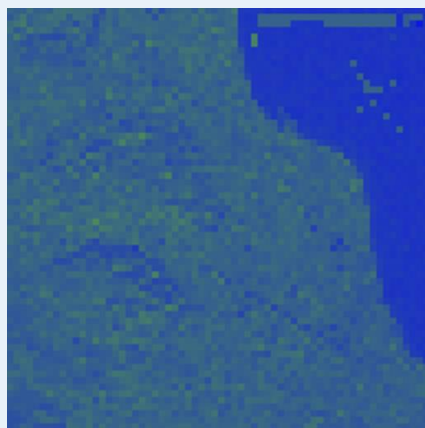
<sup>2</sup> Redundanz beschreibt einen Überfluss an Informationen bzw. deren mehrfaches Vorhandensein.



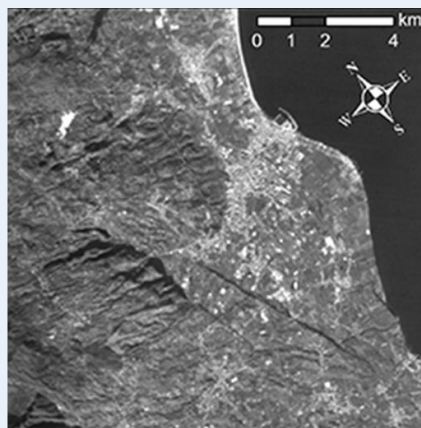
## Farbunterabtastung

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.

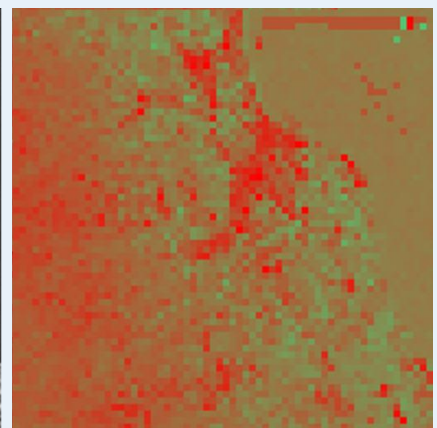
Das Bild muss dafür in einen Farbraum konvertiert werden, welcher die Helligkeit separat von der Farbdarstellung speichert, beispielsweise der YCbCr-Farbraum. Hier werden Farben über den Helligkeitskanal Y und die Farbkanäle Cb (Blue-Chrominance) und Cr (Red-Chrominance) dargestellt. In der folgenden 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.



Cb - reduziert



Y - orig.



Cr - reduziert



YCbCr

**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.