Anwendung von Geoinformatik: Entwicklung von Open Source Tools für die automatisierte Analyse von Geoinformationen

Abschlussprojekt: *Klassifizierung von High Definition Panoramabildern*
Bearbeitung: *Johannes Landmann, Philipp Hochstaffl*

Allgemeiner Überblick

Unser Programm zur Analyse & Klassifizeirung von Webcambildern ist wie folgt aufgebaut. Es besteht aus 3 Dateien (main.py, download.py und classify.py(noch isodata2.py)) wobei das Programm über die Datei main.py gestartet wird. Dabei stellt main.py einerseits die Benutzerschnittstelle zur Verfügung andererseits werden alle weiteren Funktionen und Parameter über main.py aufgerufen und übergeben. Das hier beschriebene Modul classify(isodata2.py) enthält dabei eine Funktion def isodata2.classification(x,y,z) mit drei Übergabe Parametern. Dabei ist x (timelist_format_filename) eine Liste, welche als Inhalt die Namen der Bilddateien (welche der User per Input definiert hat) im für den Download und die Klassifizierung passenden Format enthält. Die Parameter y und z definieren Werte für die Anzahl der Klassen und Iterationen und werden auch als Benutzer-Imput ins main.py eingelesen und entsprechend an download.py und isodata2.py übergeben.

Das Modul download.py

Im Modul download.py werden die 360° High Definition-Aufnahmen aus dem Internet heruntergeladen und präprozessiert. Zunächst spricht das Programm die in der vom Hauptprogramm formatiert übergebenen Zeitliste definierten Dateien auf dem Panomax-Server an und kopiert sie mit Methoden und Funktionen aus der Modulbibliothek „urllib2“ in ein zuvor erstelltes,lokales Verzeichnis. Dabei erfolgt das Laden der Dateien in den Arbeitsspeicher blockweise mit einer beliebig definierbaren Größe (default: 100000 Bytes). Eine programmierte Statusinformation zeigt an, wie viel Prozent der Einzelbilder bereits aus dem Internet heruntergeladen sind.
Die Dateien liegen sodann als acht Abschnitte vor, die jeweils 40° des Vollkreises zu einem Zeitpunkt abdecken.
In einem nächsten Schritt werden die Bilder dann in einer Liste als via „Image“-Methode aus der Python Image Library (PIL) zu einem zusammenhängenden Bild mit den Kanälen RGB (Rot, Grün, Blau) zusammengefügt, in einem dictionary abgespeichert und dann ins Filesystem exportiert. Die gewählte Methode stellt sicher, dass der Zeitraum für das Herunterladen der Bilder flexibel gehalten werden kann. Dennoch sollte er nicht allzu groß gesetzt werden, da dann der Arbeitsspeicher unnötig beansprucht wird.
Die abgespeicherten Bilder dienen in den weiteren Bearbeitungsschritten als ideale Grundlage für Klassifikation beziehungsweise Erstellung einer Maske für Himmelsbereiche.

Das Modul isodata2.py

Nun zum Modul isodata2.py. Für das Einlesen der Bilder wird die Bibliothek „Python Image Library“ (PIL) mit der Klasse „Image“ verwendet. Weiter benötigen wir das Modul Numpy um das Bild in ein Array für die weitere Klassifizierung zu zerlegen. Ein Vorteil bei der Verwendung dieser beiden Bibliotheken besteht darin, dass z.B.: ein mit der PIL eingelesenes JPEG Bild „on the fly“ in ein Numpy-Array konvertiert werden kann. Darüber hinaus bietet die PIL eine umfangreiche aber dennoch einfache und übersichtliche Benutzerschnittstelle, welche auch Online unter http://effbot.org/imagingbook/ (The Python Imaging Library Handbook, 2012) sehr gut Dokumentiert ist. Auch zum Modul Numpy finden sich zu den jeweiligen Klassen zahlreiche Beschreibungen und Tutorials Online http://www.numpy.org/ (NumPy Developers, 2013).

Isodata2.py liest die Einträge der Datei (timelist_format_filename) über eine Schleife von Anfang (erstes Bild) bis Ende (letztes Bild). Dabei wird mit jedem Schleifendurchlauf ein Bild über Image.open(Bild(i)) geöffnet (die Bilder liegen bereits vom Modul download.py heruntergeladenen und zu einem Panorama zusammengefügt im entsprechenden Ordner „temp_panomax“am Desktop vor).
Sind diese beiden Variablen importiert läuft das Skript in eine for-Schleife von welcher sequentiell das Öffnen der Bilder von tstart bis tend mit der von PIL bereitgestellten Methode image = Image.Open(Pfad/Bild) erfolgt. Anschließend erfolgt mit der Methode numpy.asarray(image) die Konvertierung des im PIL Format vorliegenden Bildes in ein Numpy-Array. Das Numpy-Array Format bietet einem Vorteile in der weiteren Verarbeitung. Man ist damit nicht ausschließlich an die Methoden der PIL Image Class gebunden und kann so das Bild einer tiefgreifenden und damit Umfangreicheren Analyse mit anderen Modulen wie z.B.: SciPy etc. unterziehen.

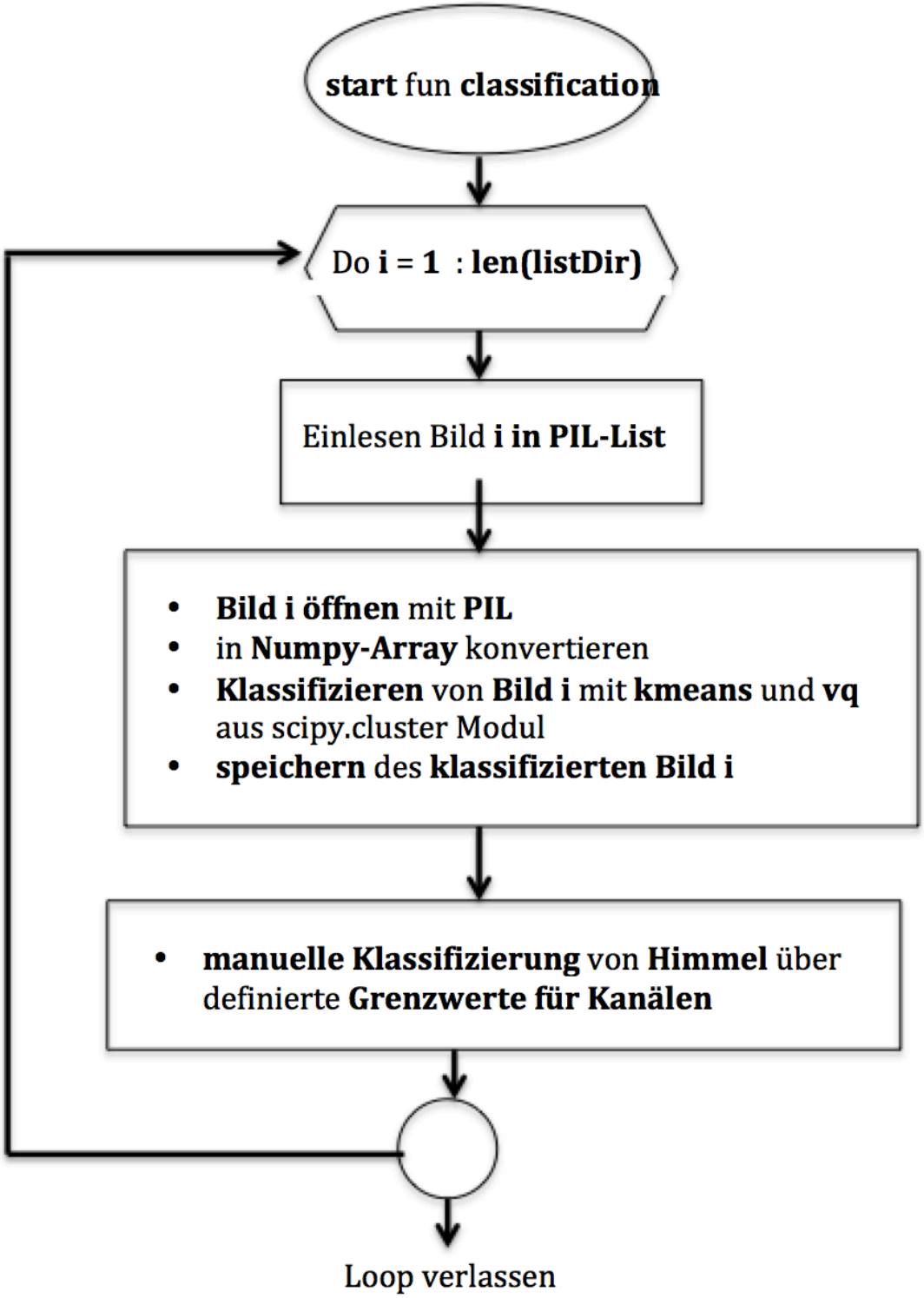
Genau diesen Vorteil nutzt unser Modul isodata2.py und verwendet für eine erste Klassifizierung die Klasse scipy.cluster.vq mit den darin zur Verfügung gestellten Methoden vq, k_means und whiten.

Klassifikation mit k_means

Die Methode k_means erstellt dabei auf Basis eines Eingabe-Vektors (unser Bild) sogenannte „Centroids“, also Punkte im Merkmalsraum, um welche die Daten mit der geringesten quadratischen Abweichung liegen. Diese Punkte stellen also die Basis für die weitere Klassifizierung mit der Methode vq dar.
Bevor nun das erste oben eingelesene Bild anhand der k_means Methode analysiert werden kann, bedarf es noch einer Anpassung bzgl. des Formats, da diese Methode wie gesagt einen Vektor im entsprechenden Format benötigt (Bildzeile*Bildspalten, Bildkanäle), als ein Vektor mit zwei Dimensionen. Die Konvertierung in das entsprechende Format erfolgt anhand des Befehls newvector = np.reshape(Vektorname, newshape). Nun wird der newvector der Methode whiten als Input übergeben whitend = whiten(newvector) um eine Normalisierung der Werte bzgl. deren Standardabweichung zu erhalten. Damit liegt der Vektor whiten nun als passender Input für die Methode k_means vor.
k_means wird nun wie folgt aufgerufen: k_means,_ = kmeans(whitened, #classes, iter = #iterations, thresh=thresholdvalue). Bis auf den Inputparameter thresh handelt sich bei allen Inputwerten um vom Benutzer spezifizierte Größen.
Der Output von k_means kann nun direkt weiter an die Methode vq übergeben werden.

Diese gibt einen Vektor aus der ganz einfach wieder über imgclass = np.reshape(newvektor, Bildzeilen, Bildspalten) in die Dimension des ursprünglichen Bildes gebracht werden kann. Dieses liegt damit klassifiziert dann natürlich nur noch in einem Kanal vor.

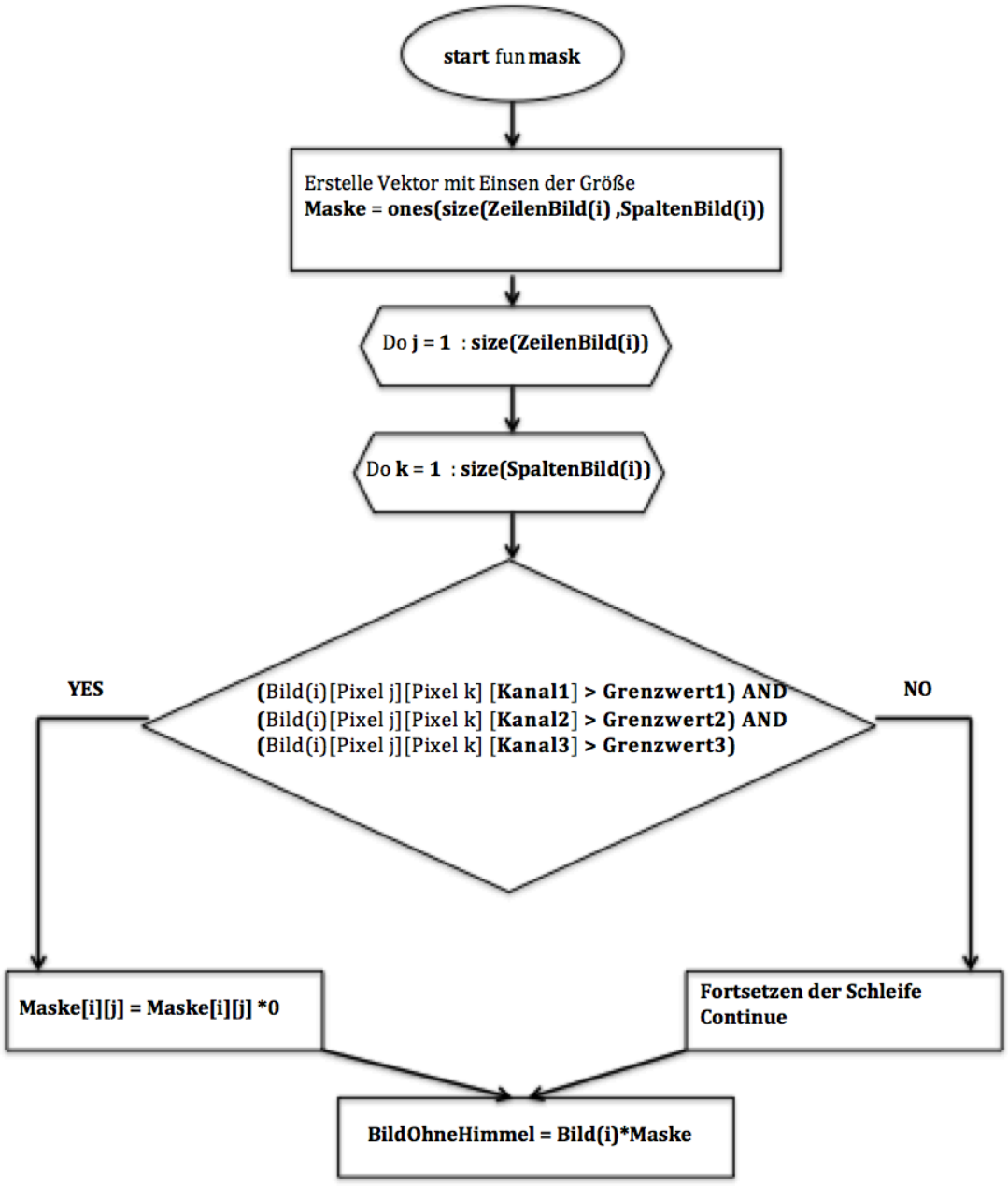
Zum Schluss wird noch mit scipy.misc.imsave(‚Pfad/Dateiname.Format', imgclass) das ganze als (in unserem Fall) JPEG ins Dateisystem des Betriebssystem geschrieben (also auf die Festplatte).



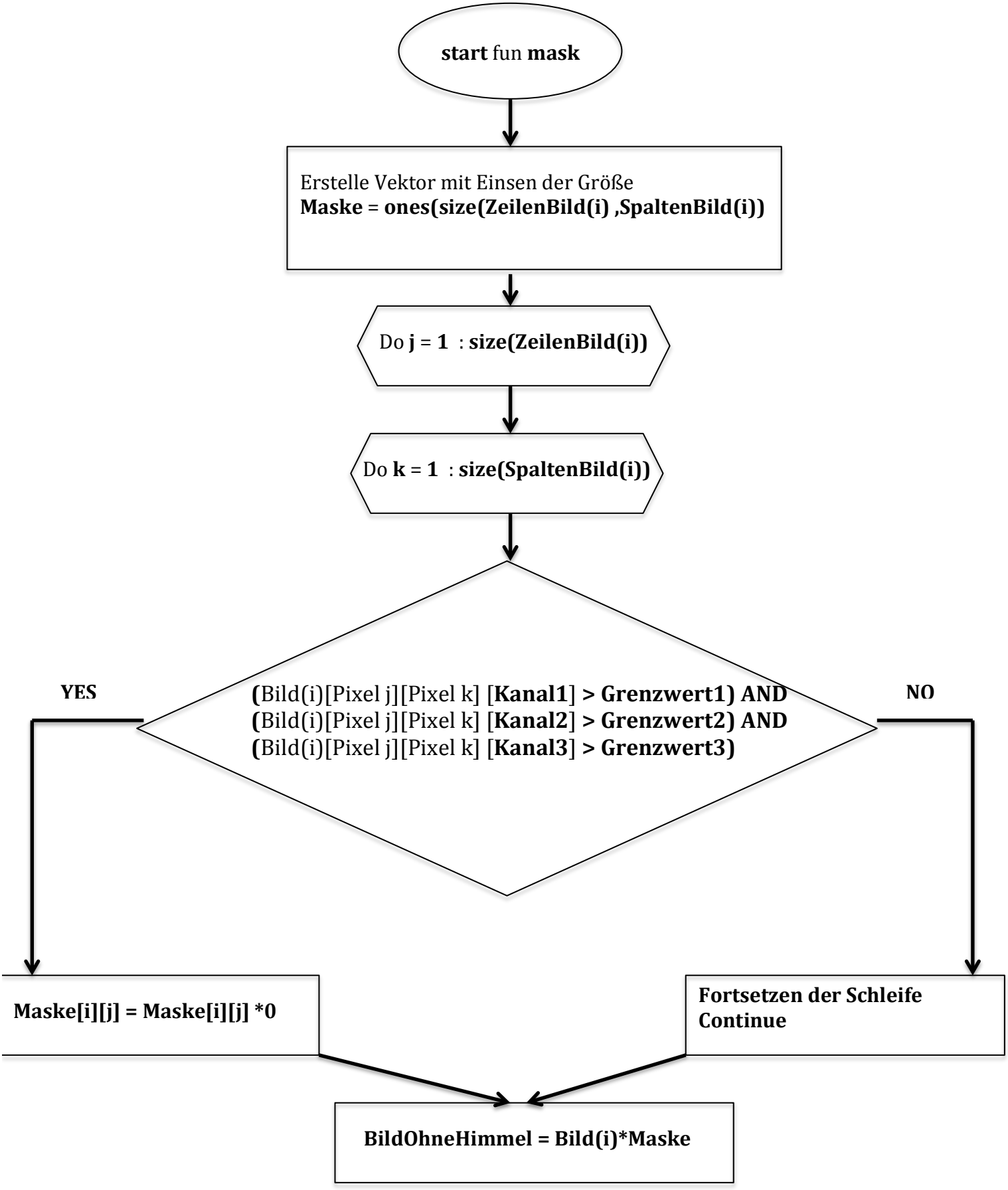
Erstellen und verschneiden einer Maske für den Bereich Himmel:

Im gleichen Schleifendurchlauf wird nun noch versucht anhand von manuell für alle drei Kanäle festgelegten Grenzwerten eine Klassifikation des Himmels vom Rest des Bildes zu erreichen. Dazu wird zu Beginn ein Vektor von der Größe des vorliegenden Bildes erstellt und mit lauter Einsen aufgefüllt. Danach startet eine Schleife über alle Zeilen und Spalten welche jeweils einen Grenzwert pro Kanal für alle drei Kanäle abfragt. Treffen alle drei Bedingungen für ein Pixel zu, so wird davon ausgegangen dass es sich um einen Bereich mit Himmel im Bild handelt und entsprechend wird das Pixel in der zuvor erstellten Einse-Array auf Null gesetzt. Ansonsten wird der Wert Eins beibehalten und das nächste Pixel analysiert.

Das so entstande Array mit Einsen und Nullen dient nun als und kann mit dem Originalbild Pixelweise verrechnet (multipliziert) werden, wodurch sich allen Werte die als Himmel Klassifiziert wurden auf Null gesetzt werden und alle anderen unverändert bleiben.



Am Ende der Schleife wird wieder mit scipy.misc.imsave(‚Pfad/Dateiname.Format', imgclass) das Array mit den Beiden Klassen Himmel/kein Himmel sowie das Bild ohne Himmel (Werte sind auf Null gesetzt) als (in unserem Fall) JPEG ins Dateisystem des Betriebssystem geschrieben (also auf die Festplatte)



Zunächst wird aber mit Methoden der PIL weitergearbeitet. Mit der Methode Kanäle = Bild.split() wird das Bild in seine Kanäle aufgetrennt. Das Ergebnis dieser Operation liefert eine Liste mit 3 Einträgen in einem PIL-Format. Mit der Methode Rot = Kanäle[0].histogram() wird eine Liste mit den Werten von 0-255 und deren jeweiligen Anzahl im Bild erstellt. Diese Liste kann nun wieder in ein Numpy-Array konvertiert werden und anschließend kann ein Plot zur Häufigkeitsverteilung der Pixelwerte im entsprechenden Kanal ausgegeben werden.