Friedrich Schiller Universität Jena

Chemisch-Geowissenschaftliche Fakultät

Fachbereich Geoinformatik

GEOG 419 - Modulare Programmierung in der FE: Python Teil I

Abschlussaufgabe

Download, Verarbeitung und Skalierung von Rasterdaten

Erstellt von:

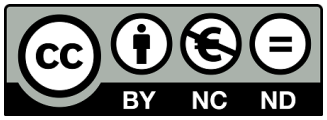
Manuel Rauch 189397

Anastasiia Vynohradova 184412

Betreut durch:

John Truckenbrodt

31. Juli 2021



INHALTSVERZEICHNIS

1. EINLEITUNG

Die in diesem Projekt verarbeiteten Daten entspringen der Copernicus-Mission. Die Sentinel-1A/B Mission ist eine Konstellation von zwei C-Band-Radargeräten mit synthetischer Apertur, die im Rahmen des Copernicus-Programms als gemeinsame Initiative der Europäischen Kommission und der Europäischen Weltraumorganisation (ESA) entwickelt wurde (Fitrzyk et al., 2019). Das Intervall von 6 Tagen zwischen den Wiederholungen der Sentinel-1-Umlaufbahn zusammen mit den kleinen Basislinien der Umlaufbahn, ermöglicht die Durchführung von Cross-SAR-Interferometrie (InSAR). Diese unterstützt geophysikalische Anwendungen wie zum Beispiel die Überwachung der Dynamik der Kryosphäre, z. B. des Gletscherflusses, und die Kartierung der Deformation der Oberfläche, die durch Tektonik, vulkanische Aktivitäten, Erdrutsche oder Bodensenkungen verursacht werden (Geudtner et al., 2021).

Die objektorientierte Programmierung mit Python gewinnt in der allen Bereichen mitunter auch in der Fernerkundung an immer größerer Bedeutung. Durch die Entwicklung verschiedenster Pakete wie gdal (GDAL/OGR contributors, 2020) oder rasterio (Gillies et al., 2013) ist die Verarbeitung von Geodaten immer handlicher und machbarer geworden. Auch die Implementierung von Python in verschiedene GIS Systeme (z.B. ArcGIS, QGIS, GRASS-GIS) macht Python für die Verarbeitung von Geodaten unerlässlich.

Ziel der Abschlussaufgabe war es ein Programm zu entwickeln, dass automatisiert ein Verzeichnis anlegt, eine Zip Datei in dieses Verzeichnis herunterlädt, ebendiese entpackt, das daraus extrahierte GeoTiff (Sentinel-1 Szene) logarithmisch skaliert und dieses letztendlich in eine neue Datei speichert. Zusätzlich sollte zur Bearbeitung der Aufgabe eine Entwicklungsumgebung geschaffen werden, die den Code auf Github organisiert und dokumentiert. Hierbei sollte eine Repository erstellt werden, die letztlich das Skript, die Setup Datei, die Requirements und eine durch das Paket Sphinx erstellte doc Dateien enthält.

2. DATENGRUNDLAGE

Bei der SAR (Synthetic Aperatur Radar) wird die Vorwärtsbewegung der aktuellen Antenne genutzt um eine sehr große Antenne zu simulieren. An jeder Position wird ein Impuls ausgesendet, das zurückgeworfene Signal durchläuft den Empfänger und wird aufgezeichnet. Die Änderung der Frequenz durch den Dopplereffekt für jeden Punkt auf dem Boden ist eine eindeutige Signatur. Die SAR-Verarbeitung umfasst die Anpassung der Dopplerfrequenz und die Demodulation durch Anpassung der Frequenzvariationen in den zurückgeworfenen Signalen von jedem Punkt am Boden. Das Ergebnis dieses angepassten Filters ist ein hochauflösendes Bild (Chan und Koo, 2008).
Die in diesem Projekt verarbeitete Szene ist eine prozessierte Szene des Copernicus SAR Satelliten Sentinel-1B aufgenommen am 28.08.2018. Diese enthält Rückstreuintensität in gamma nought und linearer Skalierung. Die Szene ist VV-polarisiert und umfasst einen Wertebereich von 0,000023 bis 120,484062.

Im Weiteren Verlauf wird schrittweise die Funktionsweise des Skriptes beschrieben und die Verarbeitung der Szene aufgeführt.

3. EINLADEN DER PAKETE

```
# import packages
import os
import requests
import zipfile36 as zipfile
import rasterio
from rasterio.plot import show
import numpy as np
from matplotlib import pyplot as plt
from skimage import exposure
from osgeo import gdal
```

Code-Snippet 1: Pakete einladen in Python

Code-Snippet 1 zeigt wie die benötigten Packete für die Bearbeitung der Aufgabe in die Environment mit dem *import* Befehl eingeladen werden. Diese können für die weitere Anwendungen auch umbenannt werden, indem sie mit einem *as* ... versehen werden. So kann das Packet numpy (Harris et al., 2020) im weiteren Verlauf mit np aufgerufen werden. Einzelne Unterfunktionen eines Packetes können mit dem *from ...import...* Befehl aufgerufen werden, wie zum Beispiel bei dem PIL (Umesh, 2012) Packet. In dem Packet *os* sind Funktionen für den Umgang mit dem Betriebssystem vorhanden. Mit Packet *requests* (Chandra und Varanasi, 2015) können URLs aufgerufen und Daten heruntergeladen werden, während das Packet *matplotlib* (Hunter, 2007b) zur Darstellung von Arrays, Matritzen, Data Frames etc. verwendet werden kann.
Skimage van der Walt et al., 2014 ist für den Kontrast oder die Streckung eines Bildes zuständig, Während *osgeo* bzw. *gdal* (GDAL/OGR contributors, 2020) und *rasterio* (Gillies et al., 2013) generell für die Verarbeitung von Geodaten verwendet werden können. Für die Installation der Packete gdal und rasterio unter Windows müssen wheel Dateien unter dem Link https://www.lfd.uci.edu/~gohlke/pythonlibs für die jeweilige Python Version heruntergeladen werden, da eine Installation via *pip install* nicht möglich ist. Anschließend kann im Terminal oder in der Eingabeaufforderung die *wheel* Datei via *pip install* installiert werden (Code Snippet 2).

```
(base) ....> pip install C:/..path../rasterio-1.2.6-cp38-cp38-win_amd64.whl
```

Code-Snippet 2: Beispiel für Installation von rasterio via wheel Datei

4. SKRIPT FUNKTIONEN

```
def tiff_download():
    # Erstelle Ordner
    # Prüft ob der Ordner bereits existiert
    if os.path.exists('GEO_ex_folder'):
        # Falls der Order existiert wird das mitgeteilt
        print("Folder exists")
    else:
        # Wenn kein Ornder exitiert wird einer angelegt und mitgeteilt
        os.makedirs('GEO_ex_folder')
        print('A new folder has been created')
    # alle zukünftigen Operationen werden in diesem Ordner durchgeführt
    os.chdir('GEO_ex_folder')
```

Code-Snippet 3: Ordner anlegen

Der erste Teil des Codes (Code-Snippet 3) zeigt die die Funktion *tiff_download*. Diese kann mit dem Befehl *from script import tiff_download*, sollte diese seperat benutzt werden, aufgerufen werden. Die Funktion führt *if, else* Bedingungen aus, indem zunächst geprüft wird ob ein bestimmter Ordner schon existiert (*os.path.exists*) und dies im wahren Fall mitteilt. Existiert dieser Ordner noch nicht, wird ein Ordner angelegt (*os.makedirs*) und dies dem der Anwender*in mit dem *print* Befehl angezeigt. Letztendlich wird der Weitere Output des Programms dann auf diesen Ordner gelenkt (*os.chdir*) und dieser als Hauptpfad definiert.

Im nächsten Block (Code-Snippet 4) testet eine if Anweisung (*os.path.file*) ob der Datensatz 'GEO419-Testdatensatz' bereits existiert und dies auch im Falle eines TRUE-Wertes ausgibt (*'Zip file exists'*). Sollte die Zip Datei 'GEO419-Testdatensatz' noch nicht existieren, wird dem User mitgeteilt, dass der Download nun beginnt (*'Download starts'*) und mit dem Befehl *requests.get(zipurl)* ausgeführt, während (*zipurl*) den Downloadlink definiert. Weiter wird der Zip Datei der Name (*'GEO419-Testdatensatz'*) zugewiesen, ei-

```
# Es wird geprüft ob der Datensatz existiert und mitgeteilt falls dieser vorhanden
# ist
if os.path.isfile('GEO419_Testdatensatz'):
    print("Zip file File exists")
# Falls der Datensatz nicht vorhanden ist wird der Download gestartet
elif not os.path.isfile('GEO419_Testdatensatz'):
    print('Download starts')
    zipurl = 'https://upload.uni-jena.de/data/60faad3254c462.96067488/
    'GEO419_Testdatensatz.zip'
    resp = requests.get(zipurl)
# abspeichern des Zip-files unter neuem Namen
    zname = "GEO419_Testdatensatz"
    zfile = open(zname, 'wb')
    zfile.write(resp.content)
    zfile.close()
```

Code-Snippet 4: Download Zip file und speichern in die Zieldatei

ne leere zum Einschreiben berechtigte Datei angelegt (*open(zname, 'wb')*), der Inhalt des Downloads in diese Datei überführt (*zfile.write(resp.content)*) und schließlich wieder geschlossen (*zfile.close()*). Zur besseren farblichen Darstellung wurden die Code-Snippets leicht gegenüber dem originalen Python Skript geändert, welche aber in dieser Form Fehler produzieren würden. So wurden beispielsweise lange Dateinamen getrennt und nochmal in der nächsten Zeile im einem Anführungszeichen versehen.

```
# Prüfen ob ein Zip file vorhanden ist
if os.path.isfile('S1B__IW___A_20180828T171447_VV_NR_Orb_Cal_ML_TF_TC.tif'):
    print("Tif file exists")
# Entpacken der Zip Datei
elif not os.path.isfile('S1B__IW___A_20180828T171447_VV_NR_Orb_Cal_ML_
'TF_TC.tif'):
    print('start unpacking')
    zfile = zipfile.ZipFile('GEO419_Testdatensatz')
    zfile.extractall()
    print('Zip unpacked')
```

Code-Snippet 5: Entpacken der Zip Datei

Code-Snippet 5 testet zunächst, ob die Tiff-Datei, die aus dem Zip File extrahiert werden soll, bereits existiert (*os.path.file*) und teilt dies wiederum den Benutzer*innen mit

(*'Tif file exists'*). Falls die Tiff Datei nicht existieren sollte (*elif not os.path.isfile*) startet das Extrahieren des Zip Files mit dem Befehl *zipfile.ZipFile('GEO419-Testdatensatz')* zum Öffnen des Zipfiles und der anschließenden Extraktion (*zfile.extractall()*). Der Start des Extrahierens und das Ende des Entpackens werden den Benutzer*innen mit den Aufrufen (*print('start unpacking')*) und (*print('Zip unpacked')*) mitgeteilt.

```
def tifcheck():
    # für nach erstmaliger Ausführung des Programms wird nochmal überprüft ob
    # das Enpacken funktioniert hat
    for file in os.listdir():
        if file.endswith(".tif"):
            print(os.path.join("GEO_ex_folder", file),"sucessfully unpacked")
        elif not os.path.isfile('S1B__IW___A_20180828T171447_VV_NR_Orb_Cal_ML_
        'TF_TC.tif'):
            print('tif does not exist, please restart the programm')
```

Code-Snippet 6: Prüfung der Extraktion

In der Funktion *tifcheck* im Code-Snippet 6 wird noch einmal geprüft, ob alle vorherigen Arbeitsschritte funktioniert haben, indem durch den neu angelegten Ordner iteriert (*for file in os.listdir()*) wird, alle Inhalte auf die Endung .tif (*if file.endswith('.tif')*) prüft. Entweder wird der komplette Dateipfad ausgegeben (*print(os.path.join(root, file)*) oder die Benutzer*innen werden dazu auffordert das Programm neu zu starten (*print('tif does not exist, please restart the programm')*), falls keine Tiff-Datei vorhanden ist.

```
def image_read():
    image = gdal.Open("S1B__IW___A_20180828T171447_VV_NR_Orb_Cal_ML_TF_TC.tif",
    gdal.GA_Update)
    return image
```

Code-Snippet 7: Lesen der Tif-Datei

Bei dem nächsten Schritt wird das Bild gelesen. Im Code-Snippet 7 wird das Raster mit der Hilfe der GDAL Bibliothek von osgeo ausgelesen. GDAL ist eine Übersetzungsbibliothek für Raster- und Vektor-Geodatenformate, die von der Open Source Geospatial Foundation unter einer Open-Source-Lizenz im X/MIT-Stil veröffentlicht wird. Als Bibliothek stellt sie der aufrufenden Anwendung ein einzigartiges abstraktes Rasterdaten-

und Vektordatenmodell für alle unterstützten Formate zur Verfügung (GDAL/OGR contributors, 2020).

```
# Logarithmische Skalierung des Bildes
def log_scale(image):
    # lesen das Bild als numpy Array
    image_array = image.GetRasterBand(1).ReadAsArray()
    # logarithmisch skalieren die Werte >0
    image_array[image_array!= 0] = (np.log10(image_array[image_array>0])) * 10
    # 0 Werte als nAn darstellen
    image_array[image_array == 0] = np.nan
    return image_array
```

Code-Snippet 8: Lesen der Datei in ein Numpy Array und logarithmische Skalierung der Rückstreuintensität

Für die weitere Skalierung (Code-Snippet 8) muss das das Band-Objekt aus dem Rasterbild mit *.GetRasterBand(1)* geholt werden, wobei die Nummer *1* den Bandindex beschreibt. In der Zeile werden die Daten gleichzeitig mit *.ReadAsArray()* in ein 2D Numeric Array gelesen. Da die Szene des Copernicus SAR Satelliten Sentinel-1B Rückstreuintensität in gamma nought und linearer Skalierung enthält, muss die Rückstreuintensität logarithmisch skaliert werden. Dafür wurde die folgende Gleichung verwendet:

$$\gamma^0_{dB} = 10 * Log_{10}(\gamma^0_{lin}) \tag{4.1}$$

Es muss aber auch berücksichtigt werden, dass 0 Werte nicht logarithmisch skaliert werden können, weil bx gleich Null ist, wobei x nicht existiert. Darüber hinaus müssen alle 0 Werte hier ausgeschlossen werden *([image != 0])*. Danach werden die 0 Werte durch NaN ersetzt.

Im Code-Snippet 9 wurde die Funktion zur weiteren Skalierung der Daten, bzw. auf die Kontraststreckung definiert. Kontraststreckung (oft auch Normalisierung genannt) ist eine einfache Bildverbesserungstechnik, die versucht, den Kontrast in einem Bild zu verbessern, indem sie den Bereich der darin enthaltenen Intensitätswerte auf einen gewünschten Wertebereich streckt, z. B. auf den vollen Bereich der Pixelwerte, den der betreffende Bildtyp zulässt (Davies, 2005). Das Bild wird noch dadurch verbessert, indem alle Pixelwerte außerhalb der *2%*- und *98%*-Perzentile ignoriert wurden und nur die dazwischen

```
# Definierung einer Funktion zur weiteren Skalierung der Daten,
# bzw. Kontraststreckung
    def rescale_intensity(log_image):
        # berechnen min und max Werte im Array
        min = np.nanmin(log_image)
        max = np.nanmax(log_image)
        # Perzentile für Kontraststreckung definieren
        # und ignorieren nAn Werte
        percentiles = np.nanpercentile(log_image, (2, 98))
        # Strecken der Intensitätsstuffen,
        # die innerhalb des 2. und 98. Perzentils liegen
        scaled = exposure.rescale_intensity(log_image,
                                            in_range=tuple(percentiles),
                                            out_range=(min, max))
        return scaled
```

Code-Snippet 9: Kontraststreckungsalgorithmus der Daten

liegenden Pixel einer Kontrastdehnung unterziehen. Die Ausreißer werden entweder auf 0 oder 255 zugeordnet, je nachdem, auf welcher Seite des Bereiches sie liegen. Für die Implementierung wurde die Scikit-image Bibliothek verwendet. Scikit-image ist eine Sammlung von Algorithmen für die Bildverarbeitung und arbeitet mit Numpy-Arrays. Hier verwendete *skimage.exposure* bietet Funktionen, die die Intensitätswerte über einen größeren Bereich verteilen (van der Walt et al., 2014). Dabei wichtig ist (*np.nanpercentile*) zur Perzentilenberechnung zu verwenden um NaN Werte auszuschließen. Das Strecken der Intensitätsstuffen erfolgt mit *exposure.rescale_intensity.*

```
# Definierung einer Funktion zum Screiben des neu berechneten Array als raster
# Bild
    def image_save():
        # Erstellen der neuen Datei im tiff Format
        driver = gdal.GetDriverByName('Gtiff')
        # Kopieren der Geoinformation vom input Bild in die neue raster Datei
        output_image = driver.CreateCopy("logscaled.tif", image, 1)
        # Kopieren des numpy Arrays in die neue raster Datei
        output_image.GetRasterBand(1).WriteArray(image_int)
        #del output_image
```

Code-Snippet 10: Schreiben des neu berechneten Array als raster Bild

Code-Snippet 10 stellt dar, wie der neu berechnete Array in eine neue Tiff-Datei gespeichert wird. Der Speicherprozess wird mit der gdal Bibliothek durchgeführt (GDAL/OGR contributors, 2020). Als Erstes wird die *CreateCopy()*-Methode des Formatdrivers (*gtiff*) aufgerufen und ein Quelldatensatz(image) übergeben, der kopiert werden soll. Danach wird der neu berechnete Array mit *WriteArray(`image_int`)* in die neue Raster-Datei kopiert.

```
# D    # Öffnen des neu berechneten Bildes mit dem Packet rasterio
    new_image = rasterio.open("..\GEO_ex_folder\logscaled.tif")
    # Erstellen einer Figur und eines Subplots
    fig, ax = plt.subplots(figsize=(7, 4))
    # imshow verwenden um den Colorbar zuordnen zu können
    # das erste Band der Datei kann mit .read(1) gelesen werden
    image_hidden = ax.imshow(new_image.read(1),
                             cmap='gray')
    # Formatierung von y Achselabels um wissenschaftliche Notation zu vermeiden
    ax.get_yaxis().get_major_formatter().set_scientific(False)
    # Plotten auf der gleichen Achse mit rasterio.plot.show
    show(new_image.read(1),
            transform=new_image.transform,
            ax=ax,
            cmap='gray')
    # Colorbar unter Verwendung des jetzt ausgeblendeten Bildes hinzufügen
    cbar = fig.colorbar(image_hidden, ax=ax)
    cbar.set_label('dB')
    # Darstellung des Bildes mit Colorbar in einem Fenster
    plt.show()
```

Code-Snippet 11: Visualisierung des logarithmisch skalierten Bilds

Die Visualisierung des logarithmisch skalierten Bildes und die Legende wird im Code-Snippet 11 dargestellt. Das Lesen des gespeicherten Bildes in der Funktion `image_save` wird hier mit dem Packet *rasterio* (**2019**) durchgeführt. Rasterio liest und schreibt GeoTIFFs , aber auch andere Formate und bietet zusätzlich eine Python-API, die auf Numpy N-dimensionalen Arrays und GeoJSON basiert. Der Befehl *rasterio.open* öffnet *logscaled.tif*. Für die Visualisierung des Bildes wird die *Matplotlib* Bibliothek (Hunter, 2007a) verwendet. Der *pyplot.subplots* Befehl erstellt mit einem einzigen Aufruf eine Abbildung und ein Raster von Subplots und bietet gleichzeitig eine angemessene Kontrolle darüber, wie die einzelnen Plots erstellt werden. *Subplots(figsize=(7, 4))* gibt eine Abbildung mit

einer Größe von 7x4 und zwei Achsen zurück. Als Nächstes zeigt das Modul *matplotlib.pyplot.imshow* die Daten als Bild an, d. h. in einem 2D-Raster. Hier wir das Rasterband *.read(1)* gelesen. Als Parameter wird hier *cmap='gray'* (Colormap) vewendet, der für die Zuordnung der skalierten Daten zu den Farben zuständig ist. Für die y Achse Formatierung wird der Befehl *ax.get_yaxis().get_major_formatter().set_scientific(False)* eingesetzt, um die wissenschaftliche Notation zu vermeiden. Das Plotten eines einzelnen Bandes als zweidimensionale Darstellung kann direkt mit pyplot von *rasterio* durchgeführt werden. Um die Koordinatenbezeichungen zu bekommen, muss bei der Übergabe von Arrays das Argument *Transform* mitgegeben werden. Die Methode *colorbar()* der matplotlib-Bibliothek wird verwendet, um einen Farbbalken zu dem Bild hinzuzufügen. Mit dem Befehl *set_label()* lässt sich eine Legendebeschriftung auf den Colorbar legen. Anschließend wird das Bild mit Achsen und Colorbar in einem Fenster angezeigt.

5. SETUP

Um das Programm schneller reproduzieren zu können, wurde eine *setup.py* Datei erstellt, von dieser alle erforderlichen Packete für die Hauptfunktion installiert werden können. Hierfür wurde zunächst eine *requirements.txt* Datei angelegt in der alle notwendigen Pakete inklusive ihrer verwendeten Version aufgeführt sind. Um *setup.py* ausführen zu können muss zunächst das Packet *setuptools* installiert werden. Dies sollte in der jeweiligen Entwicklungsumgebung passieren, nachdem ein Projekt mit der jeweiligen IDE erstellt wurde. In diese Umgebung müssen nun die Dateien *requirements.txt*, *setup.py*, *script.py* und der Ordner *doc* abgelegt werden, die unter der github repository https://github.com/anastasivynohradova/geo419 zu finden sind.

Code-Snippet 12 zeigt wie die *requirements.txt* zunächst geöffnet und eingelesen wird. der Befehl *setuptools.setup* enthält zunächst alle Metadaten zu dem Projekt, wie den Namen, die Version, die Lizenz unter der das Projekt veröffentlicht wird, die Autor*innen, deren Emails und das Github-Repository auf dem das Projekt zu finden ist. Die Befehle *setuptools.find_packages()* und *[req for req in requirements if req[:2] != "# "]* installieren die Pakete aus der Requirements-Datei. Die Installation der Pakete in dem Projekt kann durch einen Aufruf im Terminal/Prompt bewerkstelligt werden (Code Snippet 13).

```
import setuptools
# Öffnen der Requirements Datei
with open('requirements.txt', "r") as f:
    requirements = f.readlines()
#Installieren der Packete aus dem requirements file und Metadaten des Projektes
setuptools.setup(
    name='script',
    version='1.0.0',
    license='GPL3',
    author='Manuel Rauch, Anastasiia Vynohradova',
    author_email='manuel.rauch@uni-jena.de, anastasiia.vynohradova@uni-jena.de',
    url="https://github.com/anastasivynohradova/geo419/",
    packages=setuptools.find_packages(),
    install_requires=[req for req in requirements if req[:2] != "# "],
)
```

Code-Snippet 12: Installation der notwendigen Packete

```
python pip install .
```

Code-Snippet 13: Installation der Pakete im Terminal/ Prompt

Der *doc* Ordner enthält verschiedene Dateien, die unerlässlich für die Erstellung der Dokumentation des Skriptes durch HTML Dateien sind. Grundsätzlich können alle Dateien durch den Befehl *sphinx-quickstart* gerneriert und in die Entwicklungsumgebung gespeichert werden. Dies liefert aber nur die Basis Einstellungen, weswegen zur besseren Darstellung die Konfigurationsdatei *conf.py* aus der Repository des *tomography_tutorials* (Salepci et al., 2018 kopiert wurde. In der *index.rst* wurde eine Zeile eingefügt, um auf den Code zu verweisen. Die Datei *code.rst* verweist letztendlich auf die *script.py* Datei und überführt die Docstrings und Codezeilen in die HTML Dateien.
Wird der Befehl *build_sphinx* (Code Snippet 14) ausgeführt, wird in der Entwicklungsumgebung der Ordner *bulid* erstellt, der im Unterordner Sphinx die Datei *index.html* enthält, in der das gesamte Skript dokumentiert ist. In dieser Datei kann durch die verschiedenen Funktionen des Skriptes navigiert werden und auch der dazugehörige Code visualisiert werden.

```
python setup.py build_sphinx
```

Code-Snippet 14: Erstellen der Dokumentation im Terminal/ Prompt

6. ERGEBNIS

Das logarithmisch skalierte Bild wird mit den vorher beschriebenen Funktionen erzeugt. In dem Unterordner wird das skalierte Bild als Geotiff Szene gespeichert. Sollte der/die Benutzer*in das Bild mit der erzeugten Legende und den Achsen abspeichern wollen, kann dies über die Benutzeroberfläche des Programm Outputs vorgenommen werden. Hierbei kann die Szene aber nicht als Geotiff gespeichert, sondern muss in einem üblichen Bidlformat Format wie z.B PNG abgelegt werden.

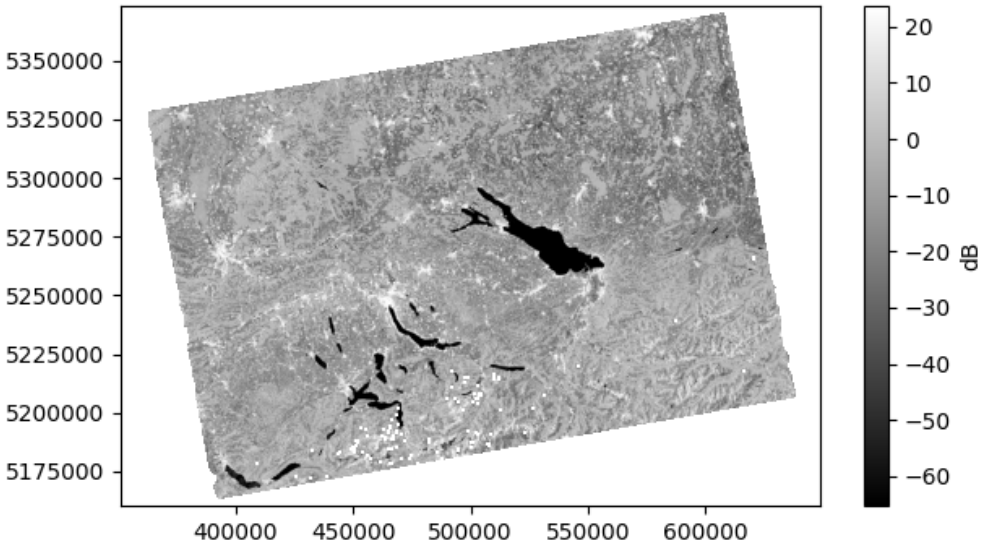


Abbildung 1: skalierten Szene

LITERATUR

Chan, Y. K. & Koo, V. (2008). An introduction to synthetic aperture radar (SAR). *Progress In Electromagnetics Research B*, *2*, 27–60.

Chandra, R. V. & Varanasi, B. S. (2015). *Python requests essentials*. Packt Publishing Ltd.

Davies, E. R. (2005). *Machine vision: Theory, algorithms, practicalities* (3rd ed.). Elsevier. http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10169929

Fitrzyk, M., Engdahl, M. & Fernandez, D. (2019). ESA Copernicus Sentinel-1 Exploitation Activities. *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*, 5389–5392.

GDAL/OGR contributors. (2020). *GDAL/OGR Geospatial Data Abstraction software Library*. Open Source Geospatial Foundation. https://gdal.org

Geudtner, D., Gebert, N., Tossaint, M., Davidson, M., Heliere, F., Traver, I. N., Furnell, R. & Torres, R. (2021). Copernicus and ESA SAR Missions. *2021 IEEE Radar Conference (RadarConf21)*, 1–6.

Gillies, S. et al. (2013). Rasterio: geospatial raster I/O for Python programmers. *Mapbox*. https://github.com/mapbox/rasterio

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007a). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Hunter, J. D. (2007b). Matplotlib: A 2D graphics environment. *Computing in science & engineering*, *9*(3), 90–95.

Salepci, N., Truckenbrodt, J. & Eckhardt, R. (2018). Tomography Tutorial. *Friedrich-Schiller-Universität*. https://github.com/EO-College/tomography_tutorial/blob/master/docs/conf.py

Umesh, P. (2012). Image Processing in Python. *CSI Communications*, *23*.

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E. & Yu, T. (2014). scikit-image: image processing in Python. *PeerJ*, *2*, e453. https://doi.org/10.7717/peerj.453