An Automatic Thresholding Algorithm for Cloud Masks

Author : Dan Brody

Advisor : Ben Davis

Course : Convex Optimization

Date: 15 May 2022

Table of Contents

Abstract

Clouds in satellite images hide information that could be helpful towards predicting forest fires, hurricanes, and more [1,2,3]. For this reason, cloud mask algorithms have been created to mask out clouds in satellite images. An example of a cloud mask can be found in Fig.1.

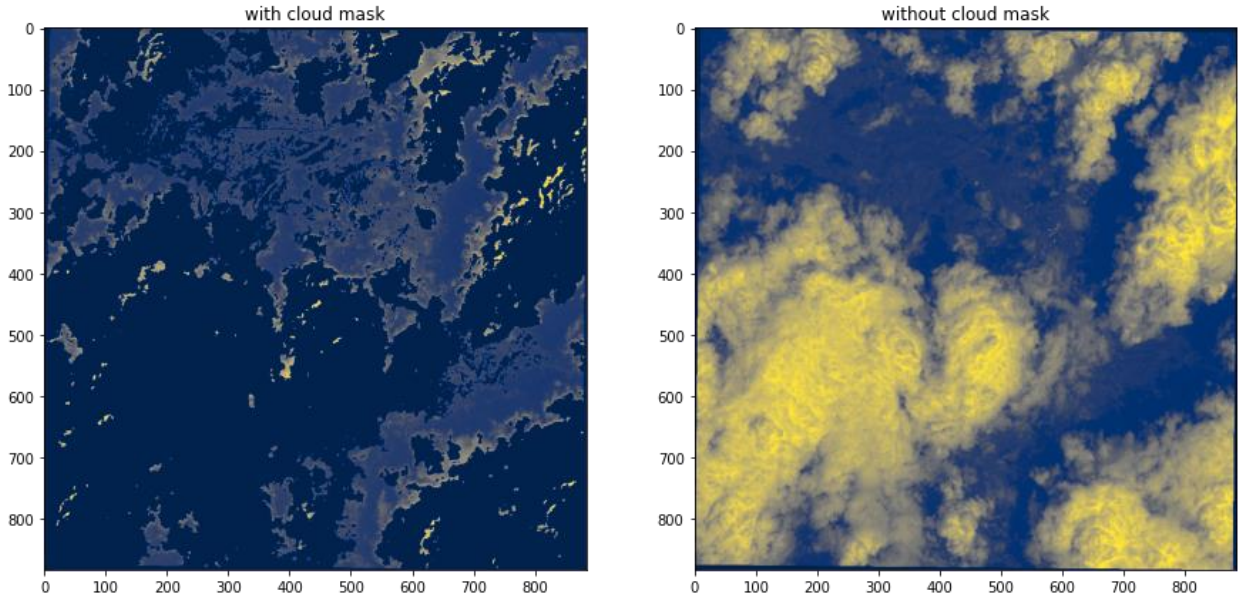

Fig.1. Sentinel-2 L1C Image with and without Cloud Mask [4]

An interesting property of satellite images is that they are multispectral. Every dimension, known as a band, in a satellite image corresponds to the same image but imaged over different wavelength ranges. Different wavelength ranges can expose different properties of the environment (i.e. blue band (450-495 nm) is high for dense cloud so higher values should be masked out). A typical and quick algorithm researchers use for cloud masking is to impose thresholds on different bands. An issue is that thresholds are chosen manually by looking at a histogram of the band across all images and trying to find the separating line that divides typical from anomalous points such as in Fig.2.

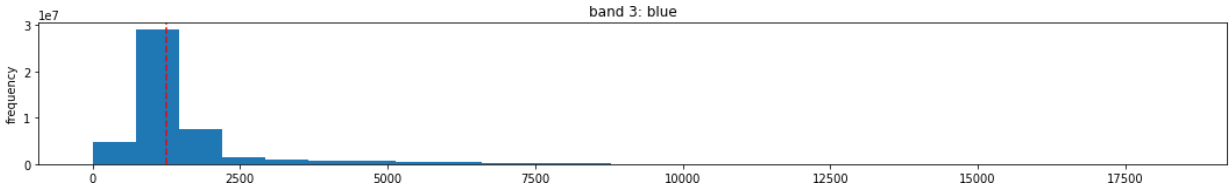

Fig.2. Blue Band Histogram of Sentinel-2 L1C bands from June 28th,2018 to December 30th, 2018 w/ Dashed Line on Threshold [4]

This project sets to improve upon manually setting thresholds for cloud masks by automatically setting thresholds via convex optimization.

Introduction

Satellite images are widely used for secondary use to predict several events including forest fires, hurricanes, and more [1,2,3]. An issue, however, with satellite images is that clouds block the view of the image making the essential parts of the image potentially harder to see. Since possible predictors towards a target in a model are hidden, the cloud cover reduces the performance of any deep or machine learning algorithms on satellite images. In addition, clouds generally introduce noise or unnecessary information into the model, contributing more towards the decreased performance. To solve this problem cloud masks have been created such as in [3].

A very simple and effective cloud mask can be created by utilizing one of the aspects of satellite imaging, bands. Satellite images image at different ranges of wavelengths to detect different objects and each range is represented as band or slice of the image. As an example, the blue band is detected by a wavelength in the range of about 0.45-0.52 micrometers. This band is useful because dense clouds (clouds with a defined shape and thickness) tend to have higher reflectance in the blue band. A typical cloud mask will use thresholds on bands, such as the blue band, to mask out the clouds as seen in Fig.1.

The way in which thresholds are found is by looking at histograms of different bands and making educated guesses as to where the outliers may lie. Let us consider the blue band in Sentinel-2 L1C with images taken from June 28th,2018 to December 30th, 2018 [4]. The histogram is as shown in Fig.1.

In looking at the histogram of Fig.1. we can see that the distribution of the blue band levels off a little after 1250 so we can set a threshold there to eliminate dense clouds in the image. Consider this same methodology for every other threshold being used. When all thresholds have been calculated the final cloud mask is created by masking out all points that match all thresholds.

To illustrate errors in cloud masks [5] shows that the Sentinel-2 L1C cloud mask underestimates cloud coverage. As one of the figures in [5] Fig.3. shows how the Sentinel L1-C cloud mask estimates cloud cover.

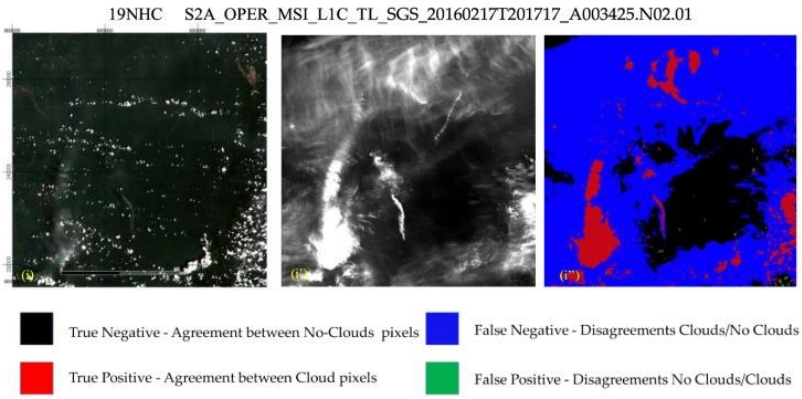

Fig.3. Figure from [5] showing natural color composite (combining B4, B3, and B2) (left), band 10 (middle), and the validation (right)

Much of the cirrus cloud (clouds with no defined shape) for the image are not detected as shown by the large blue area (blue = false negatives) on the right image (validation) where there are wispy cirrus clouds in the middle image (band 10; band used for detecting cirrus). Please note that if we use the same threshold as that in the introduction about the blue band detecting dense clouds (1250) then cirrus clouds would not be detected. This is because cirrus clouds have low, as opposed to high, reflectance in the blue band. Thus, we need to detect cirrus and dense clouds separately.

Dense Cloud Detection – Filtered Bands

For dense cloud detection bands B2, B11, B12 and derived bands Normalized Difference Vegetation Index (NDVI) and Normalized Difference Snow Index (NDSI) can be used for the thresholding.

Let us first discuss the derived band NDVI to detect dense clouds. NDVI is defined with the NIR (Near Infrared) band and red band as:

$$NDVI = \frac{(NIR-Red)}{(NIR+Red)} \quad (1)$$

The measure has a range from -1 to 1 and represents the vegetation in the image. Negative values of NDVI generally correspond to water and dense clouds whereas values closer to zero resemble rock and snow, with larger values representing varying vegetation. As water and dense cloud represent the lower values of NDVI we aim to compute a threshold to mask out values lower than the threshold.

Concerning the blue band, B2, dense clouds have high reflectance in B2, so the aim is to threshold out values larger than the threshold.

Additionally, concerning Short Wave Infrared (SWIR) bands B11 (SWIR1) and B12 (SWIR2) the bands should be used for dense cloud detection since clouds can sometimes be mistaken as snow or ice. Both snow and ice have low reflectance in B11 and B12, so the aim is to threshold out values larger than the threshold.

In the case that SWIR bands B11 and B12 can't be used we can also NDSI. NDSI is defined with the SWIR-1 band and green band as:

$$NDSI = \frac{(Green - SWIR1)}{(Green + SWIR1)} \quad (2)$$

The measure ranges from -1 to 1 where positive values indicate snow is present and negative values indicate snow-free land. The aim here would be to threshold out values less than the threshold such that snow is not masked out with dense cloud.

Cirrus Cloud Detection – Filtered Bands

For cirrus cloud detection bands B2 and B10 can be used for the thresholding.

Concerning the blue band, B2, cirrus clouds have low reflectance in B2, so the aim is to threshold out values smaller than the threshold.

Additionally, concerning SWIR band this is a high atmospheric absorption band where high-altitude clouds are detected. Cirrus in B10 have high reflectance so the aim would be to create a threshold where higher values are masked.

As opposed to dense cloud we cannot use NDSI, B11, or B12 since cirrus clouds can easily be mistaken as snow due to their wispy nature.

Problem Description

Formulation

To create the cloud mask, we can take advantage of the fact that means of pixels across bands are similar if all pixels belong to the same surface. A measure found in [6] can help separate surfaces:

$$\cos(\theta_{rel}) = \frac{\boldsymbol{K(x,x_m)-K(x,y_m)-K(x_m,y_m)+K(y_m,y_m)}}{(\sqrt{\boldsymbol{K(x,x)-2K(x,y_m)+K(y_m,y_m)}})(\sqrt{\boldsymbol{K(x_m,x_m)-2K(x_m,y_m)+K(y_m,y_m)}})} \quad (3)$$

Where **x** is a set of pixels that should belong to a surface (i.e. cloud) that are represented in terms of their features (i.e. blue band in slice 1, NDVI in slice 2, etc.), $\boldsymbol{x_m}$ is the mean of the pixels in the surface that **x** corresponds to where $\boldsymbol{x_m}$ is computed feature wise, $\boldsymbol{y_m}$ is the mean of the pixels in a different surface (i.e. rock and vegetation) computed feature wise, and **K(x,y)** is a kernel function that measures the distance between **x** and **y** in a higher dimensional space.

If all pixels $\boldsymbol{x_i} \in \boldsymbol{x}$ in the surface has the same sign when compared to all other surfaces tested, then all $\boldsymbol{x_i} \in \boldsymbol{x}$ are in the same surface to a level based on the value of $\cos(\theta_{rel})$. Values closer to zero are less likely to be a part of the surface. Taking advantage of the fact define sgn (·) as

$$\text{sgn}(\cdot) = \begin{cases} 1, x > 0 \\ 0, x = 0 \\ -1, x < 0 \end{cases} \quad (4)$$

The larger the amount of values of $\cos(\theta_{rel}))_i, \forall\, i = 1,2,\dots,z$, the larger is $|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|$ so this function is set as the objective function of the problem to maximize or we can take the negative and minimize. It should be noted that 0 is not valid for sgn (·) to evaluate (we only want whether the value is positive or negative) but if $\cos(\theta_{rel}))_i$ is equal to zero then the pixel does not belong to a surface, which should not occur, so we make the assumption that $\cos(\theta_{rel}))_i \neq 0$.

In order to further constrain the solution we can constrain how spread out the values of $\cos(\theta_{rel}))_i, \forall\, i = 1,2,\dots,z$. If we define coefficient of variation (CV) as

$$\text{CV}(\cdot) = \frac{std(\cdot)}{mean(\cdot)} \quad (5)$$

Then we can take $|\text{CV}(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|$ (since mean can be negative if more signs are negative) and impose constraints. CC shows the extent of variability in relation to the mean of the population and the higher the CV, the greater the dispersion. The advantage of CV over other measures such as standard deviation or root mean squared error is that the measure is unitless so can easily be compared across samples of different sizes [7]. An issue with CV, however, is that mean(·) cannot be zero. Based on [8] CV < 1 is considered low variance so we constrain $1 - \varepsilon \geq |\text{CV}(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| \geq 0, \varepsilon > 0$. Since $\mathbf{cos}(\boldsymbol{\theta_{rel}}))$ can consist of both negative and positive values a

mean of 0 can exist however we can assume that if there is low variance then most of the signs of the pixels must be the same so, considering the constraints, a mean of zero should not exist.

The dataset being used to construct the toy dataset is imagery from Sentinel 2 (L1C, Top of Atmosphere) taken over the greater Santa Fe metro area from 2019 to 2020 [4]. Each GeoTIFF file contains seven bands (in corresponding order) [red, green, blue, nir, swir1, swir2, alpha] where the alpha band indicates unnecessary information. The alpha band and cloud mask will be used to mask out unnecessary data in the satellite image. Fig.1. will provide more information on the distribution of the blue band. 4x4 images will be taken by computing a 4x4 window across the cloudiest image in the dataset. All images are of the same shape so z (the number of images from the windowing) will not change if using a different image from the dataset.

For the first implementation we will have the blue band and NDVI used for thresholds and only detect dense clouds. The blue band threshold (blue_t) will be in [0,1) where the threshold will represent the percentile of the band values and higher values are masked out (i.e. blue_t = .5 for array [0,1,2] represents that the pixels in the band with values greater than 1 will be masked out). All values in the band greater than the percentile will be considered dense cloud. To prevent all values to be masked out the threshold will start where the value of the percentile stops being 0. The other surface that will be tested is rock and vegetation where NDVI values greater than or equal to 0 belong to this surface. The threshold is set to 0 because negative values of NDVI generally correspond to water and dense clouds and larger values and values closer to zero correspond to varying rock, vegetation, and snow (RVS). Since values can be undefined, due to the use of a denominator, we convert all undefined values to -1 (i.e. not in the RVS surface since $-1 < 0$) to prevent any misclassified pixels.

The surfaces are cloud vs. RVS so any pixels that belong to cloud should not belong to RVS. The bands used to match surfaces are SWIR1, green, red, NIR, and blue band. The reason for using blue band is since dense clouds have higher reflectance. The reason for using the green and SWIR1 bands are because these bands are used to compute NDSI which represents snow, contributing to the RVS surface. The reason for using the red and NIR bands are because these are used to computed NDVI, which plays a major role in diagnosing whether a pixel is in the RVS surface. The only reason for not using NDVI and NDSI to match surfaces is that, since there exists a denominator, the metric is undefined.

Let us outline the formulation as follows:

1. Let x_i denote a satellite image in a corpus of images where $s_i \in \mathbb{R}^{nxmxp}$ where p is the number of bands and $s_i \in S = \mathbb{R}^{nxmxpxz}$. For the toy dataset n=m=4,p=5, and z = 781456

2. Let any pixel in s_i at position (j,k) in band d be denoted as u_{jkd}. Let us assume blue band in band 1, NDVI in band 2

3. Let blue_t denote the threshold for the blue band for dense clouds and the NDVI threshold (ndvi_t) be a constant 0.

4. Let cloud be denoted as u_{jk1} > blue_t for $s_i \in S$ and rvs be denoted as u_{jk2} >= 0 for $s_i \in S$

$$\min_{blue_t \in [0,1)} -|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|$$

Subject to:

$\mathbf{x} \in$ *cloud (blue band > blue_t),* **refer to (3) for definition of* $\boldsymbol{x}$

$\mathbf{y} \in$ *rvs (ndvi >= 0),* **refer to (3) for definition of* $\boldsymbol{y}$

$1 - \mathcal{E} \geq |CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| \geq 0\,, 1 > \varepsilon > 0, \mathcal{E}$ *can be chosen with precision*

Problem Type

Within convex optimization, there are different types of problems for which solvers are made for. Types of problems include Linear Programming (LP), Nonlinear Programming (NLP), Mixed Integer Nonlinear Programming (MINLP), and more. The formulation at hand is an LP because the numbers can be manipulated to create the standard form of an LP:

$$\min_{x \in \mathbb{R}^n} \ \mathbf{c}^{\mathrm{T}}\mathbf{x}$$
$$\text{Subject to:} \quad \mathbf{Ax} = \mathbf{b} \qquad (6)$$
$$\mathbf{x} \geq \mathbf{0}$$

We can transform the previous formulation by introducing slack in the inequality, $1 - \mathcal{E} \geq |CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| \geq 0$, $\varepsilon > 0$ such that the inequality turns into an equality and by subtracting ε from both sides of $|\text{mean}\ (\mathbf{cos}(\boldsymbol{\theta_{rel}}))| \geq 0 + \varepsilon, \varepsilon > 0$ such to form:

$$\min_{blue_t \in [0,1)} -|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|$$

Subject to:

x ∈ *cloud (blue band > blue_t)*

y∈ $rock\_and\_vegetation$ *(ndvi >= 0)*

$$|CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| + h_1 = 1 - \mathcal{E} \ , \ 1 > \varepsilon > 0$$
$$|CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| \geq 0$$
$$h_1 \geq 0$$

If we let **x** = $[|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|, |CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|, h_1\ ]^T$, **c** =$[1,0,0]^T$, **A** = [0 1 1] , **b** = $1 - \mathcal{E}$, $\varepsilon > 0$ where then the problem has simplified to a standard form LP:

$$\min_{blue_t \in [0,1)} ([-1,0,0]^T)^T\ [|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|, |CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|, h_1\ ]^T$$

Subject to:

x ∈ *cloud (blue band > blue_t)* *(9)*

y∈ $rock\_and\_vegetation$ *(ndvi >= 0)*

$$[\,0 \quad 1 \quad 1\,]\,[|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|, |CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|\,, h_1\,]^T = 1 - \mathcal{E},\ 1 > \varepsilon > 0$$

$$[|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|, |CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|, h_1\,]^T \geq \mathbf{0}, \qquad \mathbf{0} \in \mathbb{R}^3$$

Solvers

Solvers for LP problems include simplex and non-simplex methods.

The simplex method solves for **x** in **Ax** = **b** by first creating an augmented matrix containing $\mathbf{b} \in \mathbb{R}^{1xn}$ and $\mathbf{A} \in \mathbb{R}^{mxn}$:

$$[\mathbf{A},\mathbf{b}] \qquad (10)$$

We can use elementary row operations on the augmented matrix such to convert the matrix **A** into the identity matrix **I** with the corresponding $\widetilde{\boldsymbol{b}}$ to $\boldsymbol{Ix} = \widetilde{\boldsymbol{b}}$ resulting from the elementary operations:

$$[\boldsymbol{I}, \widetilde{\boldsymbol{b}}] \qquad (11)$$

If we represent the sequence of elementary operations as $\boldsymbol{E} = \boldsymbol{E}_t \ldots \boldsymbol{E}_1$ and put them on both sides of **Ax** = **b** then $\boldsymbol{EAx} = \boldsymbol{Ix} = \boldsymbol{Eb} = \widetilde{\boldsymbol{b}}$ so the solution to **x** is $\widetilde{\boldsymbol{b}}$**.**

Note that if **A** is not a square matrix, then an extra unnecessary matrix **D** will be a part of the augmented matrix. The reason for this is that the basis for the matrix will have the dimension of the rank of the matrix so if m < n then the basis will be of dimension m and vice versa. Any linearly dependent columns will prohibit **A** from having an inverse, so these columns are separated into **D**, making the new matrix $\widetilde{\boldsymbol{A}}$ square and invertible.

Non-simplex methods were created since simplex methods have exponential complexity with respect to the size of **x** in the standard form of the LP. This means that with **x** of large dimension solvers would take a very long time. Some of these methods include Khachiyan's method and Kamarkar's method.

Khachiyan's method generates a "decreasing" sequence of ellipsoids in R n that are guaranteed to contain a minimizing point, using the idea that given a subgradient (or quasigradient) at a point, we can find a half-space containing the point that is guaranteed not to contain any minimizers of f, i.e., a cutting-plane [9]. The method was able to prove polynomial time solvability for LPs but in practice ended up becoming slower than the simplex method. Many later algorithms took inspiration including Kamarkar's method, an interior-point method.

Instead of following the boundary of the feasible set the method moves through the interior of the feasible region, improving the approximation of the optimal solution by a definite fraction with every iteration, and converging to an optimal solution with rational data. [10]. Kamarkar's

method works in polynomial time, as Khachiyan's, but is more feasible in practice, outperforming the simplex method

In the case of this formulation the threshold for blue band, blue_t, is the only changing variable. The issue however is that blue_t is not used in a mathematical way, moreso in a logical way. Pixels belonging to the cloud surface and RVS surface need to be specified by specifying blue_t before any calculations take place. Therefore, we cannot use any of the above solvers for the problem. All that can be done is providing discrete values of the threshold, pushing them through the formulation, and finding meaning in the results in the sensitivity analysis.

Existence/Proofs

To prove existence feasibility and solvability must be defined.

For a solution, direction, etc. to be feasible they must satisfy the constraints of the problem. Let us consider that we find a solution to the objective function in the formulation but $CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| > 1$. The solution goes against the constraints of the problem, so the solution is not feasible. In the case of solvability, a problem is solvable if there exist solutions to the problem. If the feasible set considering the constraints is empty, then a problem is not solvable. This leads us to how we can prove existence of an optimum for a formulation.

One way in which to prove that there is an optimum is to use Weierstrass. The Weierstrass Theorem states that if $f : \Omega \rightarrow \mathbb{R}$, $\Omega \subset \mathbb{R}^n$, is continuous on a nonempty feasible set Ω that is closed and bounded (compact) then f(x) has a global minimum in Ω [11].

Necessary conditions for a minimizer can be stated when following FONC (First Order Necessary Conditions), SONC (Second Order Necessary Conditions), and SOSC (Second Order Sufficient Conditions). FONC can be stated as follows in [12]:

Let Ω be a subset of $\mathbb{R}^n$ and $f \in C^1$ a real-valued function on Ω. If **x*** is a local minimizer of f over Ω, then for any feasible direction **d** at **x***, we have

$$\boldsymbol{d}^T \nabla f(\boldsymbol{x}^*) \geq \mathbf{0}$$

A simple case in which FONC can be satisfied is if **x*** is an interior point, meaning that the minimizer is not on the boundary. For this case the set of feasible directions is all $\mathbb{R}^n$ and we have

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0}$$

Another of the necessary conditions for a minimizer is SONC which can be stated as follows as in [12]:

Let Ω be a subset of $\mathbb{R}^n$ and $f \in C^2$ be a function on Ω, **x*** a local minimizer of f over Ω, and **d** a feasible direction at **x***. If $\boldsymbol{d}^T \nabla f(\boldsymbol{x}^*) = 0$, then

$$\boldsymbol{d}^T \mathbf{F}(\boldsymbol{x}^*) \boldsymbol{d} \geq 0$$ (**F** is the Hessian of f.)

In the interior point case, we can prove SOSC to show both FONC and SONC. SOSC states (as in [12]):

Let $f \in C^2$ be defined on a region in which **x*** is an interior point. Suppose that

1. V/(**x***) = 0.

2. **F**(**x***) > 0.

Then, **x*** is a strict local minimizer of f.

Proofs

Weierstrass

Let us use Weierstrass theorem to prove there exists a global minimum in the cloud thresholding formulation.

To prove the feasible set is closed and bounded let us define the constraints on **x** = $[CV(\mathbf{cos}(\boldsymbol{\theta_{rel}})), h_1, h_2\ , \cos(\theta_{rel}))_1\ , \cos(\theta_{rel}))_2\ \ldots\ldots \cos(\theta_{rel}))_{z=781456}]^T$

Based on the formulation we know:

$1 - \mathcal{E}\ \geq |CV(\mathbf{cos}(\boldsymbol{\theta_{rel}}))||\geq, \varepsilon > 0 \qquad , blue_t \in [0,1)$

In summary $\Omega = \{blue_t: 1 - \mathcal{E}\ \geq |CV(\boldsymbol{cos}(\boldsymbol{\theta_{rel}}))| \geq 0,\ 1 >\ \varepsilon > 0\ , blue_t \in [0,1)\} \in \mathbb{R}$

An example of a point in Ω could be where blue_t corresponds to the value of the second highest pixel in the blue band. In this case $|CV(\boldsymbol{cos}(\boldsymbol{\theta_{rel}}))| = 0$ because $(\boldsymbol{cos}(\boldsymbol{\theta_{rel}}))$ will only have one value in the cloud mask so $std((\boldsymbol{cos}(\boldsymbol{\theta_{rel}}))) = 0$.

Based on the above inequalities and examples the nonempty feasible set Ω is closed (since it contains its boundary points) and bounded (since the feasible set can be enclosed in a ball centered at (0,0) with a radius of 1).

To show that the formulation is continuous we can show that there exist no discontinuities and it satisfies the sequence definition of continuity. By the sequence definition of continuity, a function *f* is continuous if:

$\lim f(x_n) = f(\lim x_n)$ [13]

Suppose we have a convergent sequence $\boldsymbol{t_n}$ of thresholds with $\lim \boldsymbol{t_n} = t_0$. Since the assumption is that $\cos(\theta_{rel}))_i\ , \forall\ i = 1,2, \ldots, z =\ 781456$ all belong to a surface $(\mathbf{cos}(\boldsymbol{\theta_{rel}}))$ is never 0 so the removable discontinuity in the sign function is removed. Since the sign and cosine function are continuous for this problem and $\mathbf{cos}(\boldsymbol{\theta_{rel}}))$ is a function $h(\cdot)$ of the threshold:

$\lim f(\boldsymbol{t_n}) = \lim -|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))\ | = \lim -|\sum sgn(\boldsymbol{h}(\boldsymbol{t_n}))\ | = -|\sum sgn(h(t_0))|=$

$-|\sum sgn\big(h(t_0)\big)| = -|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| =\ f(t_0)$

Furthermore, suppose we start with a satellite image that has 50 pixels on the cloud surface and 50 pixels on the RVS surface. If the threshold starts at 0 in the blue band then $-|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))\ |$ = -0. However dense clouds have higher reflectance in the blue band so as the threshold increases the number of pixels in the RVS surface decrease. Therefore the number of pixels continuously decrease until there are none left and no discontinuities exist.

Since the formulation is continuous on a nonempty feasible set Ω that that is closed and bounded then f(x) has a global minimum in Ω.

FONC and SONC

Let us now prove FONC and SONC.

An argument can be made that the minimizer must be an interior point. In looking at the feasible set $\Omega\ CV(\boldsymbol{cos}(\boldsymbol{\theta_{rel}}))| = 0$ when all $\cos(\theta_{rel}))_i = 1$. This case, however, is not entirely feasible as there will be too few pixels in the cloud mask. The other boundary case is when $|CV(\boldsymbol{cos}(\boldsymbol{\theta_{rel}}))| = 1 - \mathcal{E}$. If $\mathcal{E}$ is high then the minimizer may appear at this boundary case but, based on the assumptions, a lower variance should indicate a better result. Therefore if $\mathcal{E}$ is low both boundary cases are unlikely.

If the local minimizer $\boldsymbol{x}^*$ is an interior point, then $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$ satisfying FONC. We must now prove SONC for $\boldsymbol{x}^*$

$$\boldsymbol{F}(\boldsymbol{x}^*) = -\nabla^2|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| = -\nabla(\frac{\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))}{|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|}\ \nabla(\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}})))) =$$
$$-\nabla(\frac{\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))}{|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|}\ \Big(\nabla\big(sgn(\cos(\theta_{rel}))_1\big) + \nabla\big(\ sgn(\cos(\theta_{rel}))_2\big) + \cdots +$$
$$\nabla\big(sgn(\cos(\theta_{rel}))_z\big)\Big) = 0\ (since\ derivative\ of\ sign\ function\ is\ 0)$$

Hence $\boldsymbol{F}(\boldsymbol{x}^*)$ positive semidefinite and $\boldsymbol{d}^T\mathbf{F}(\boldsymbol{x}^*)\boldsymbol{d} \geq 0$ so SONC is satisfied. The necessary conditions for a local minimizer hold.

Results

The program used to solve the problem was Python. In particular I utilized Jupyter Notebook and frameworks such as NumPy[14], gdal[15], and pandas[16]. The way in which the problem was solved is that I created **x** in (3) by filtering the bands described in the formulation (SWIR1, green, blue, red, NIR) to the pixels where the blue band exceeds the threshold blue_t and **y** in (3), similarly, by filtering the same bands (SWIR1, green, blue, red, NIR) to the pixels where NDVI exceeds or is equal to 0, ndvi_t, and then computed $\cos(\theta_{rel})$) for each image to create $\mathbf{cos}(\boldsymbol{\theta_{rel}})$). This was done for multiple discrete thresholds. The constraints and objective function were computed as functions of $\mathbf{cos}(\boldsymbol{\theta_{rel}})$)

In total the number of optimization variables is z = 781456 (the number of 4x4 images) with two constraints. The feasible set can be defined as $\Omega = \{[\cos(\theta_{rel}))_i, \forall\, i = 1,2,\ldots.,z]^T : 1 - \mathcal{E} \geq |CV(\boldsymbol{cos}(\boldsymbol{\theta_{rel}}))| \geq 0,\ |\text{mean}\,(\mathbf{cos}(\boldsymbol{\theta_{rel}}))| \geq 0 + \varepsilon,\ \varepsilon > 0\ \}$. The total time for the solver to run with the toy dataset was 159.3 seconds. If the problem could have been formulated towards a ready-made solver implementing simplex or Kamarkar's method, then the solver may have run faster.

Let us go into the results for a sensitivity analysis between different discrete thresholds for the blue band (blue_t). The thresholds start at an area where there are no zeros in the blue band since most of the pixels have a value of zero (consider Fig.1.) and masking out values greater than zero is pointless since all other values are nonzero and positive, essentially masking out the whole image. As ground truth we use the threshold computed from looking at the histogram of the blue band in [4] at Fig.1., giving a threshold of 1250.

Fig.4. visually shows the results of the experiment in terms of the blue band. When looking at the cloudiest image the threshold on the blue band in which the values stopped being zero was at ≈ .94 so we set blue_t to iterate between .95, .98, and .99. The row surrounded by a green rectangle below indicates the optimal value and solution. The threshold can be found in the first column as 4821 or as the optimum value that resulted in the threshold, blue_t = 0.98. The optimum solution can be found in the last column where the metric in the legend is $|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|$ so the optimum solution is the negation, -249860. The optimal solution being this number means that there were 249,860 pixels that had the same sign so this is the maximal amount

The optimal value represents the optimal threshold on the blue band for cloud masks, specifically dense cloud. The threshold masks out the densest clouds and the least amount of the RVS surface. Looking at the cloudiest image in Fig.4. without a cloud mask (second column) we can see the whole image consists of cloud-like features so we would expect to threshold at a low value. Therefore, originally, the threshold at 2782.0 (or blue_t = .95) seems much more promising since more of the cloud is masked. However, let us look at the same image in the NDSI band in Fig.5. where snow has higher reflectance. Refer to the red rectangles in the first row. Where the snow has high reflectance in the image without a cloud mask the image with an applied cloud mask masks the area out. Hence a noticeable part of the cloud mask contains the RVS surface.

The smallest threshold for the blue band that creates a cloud mask that masks out the most pixels of the same surface is the optimal value, blue_t = .98. While blue_t = .99 does create

a cloud mask where all pixels are of the same surface the threshold misses out on many pixels that have a high reflectance in the blue band (indicative of dense cloud) as shown in comparing column 1 and column 2 of row 3 in Fig.4.

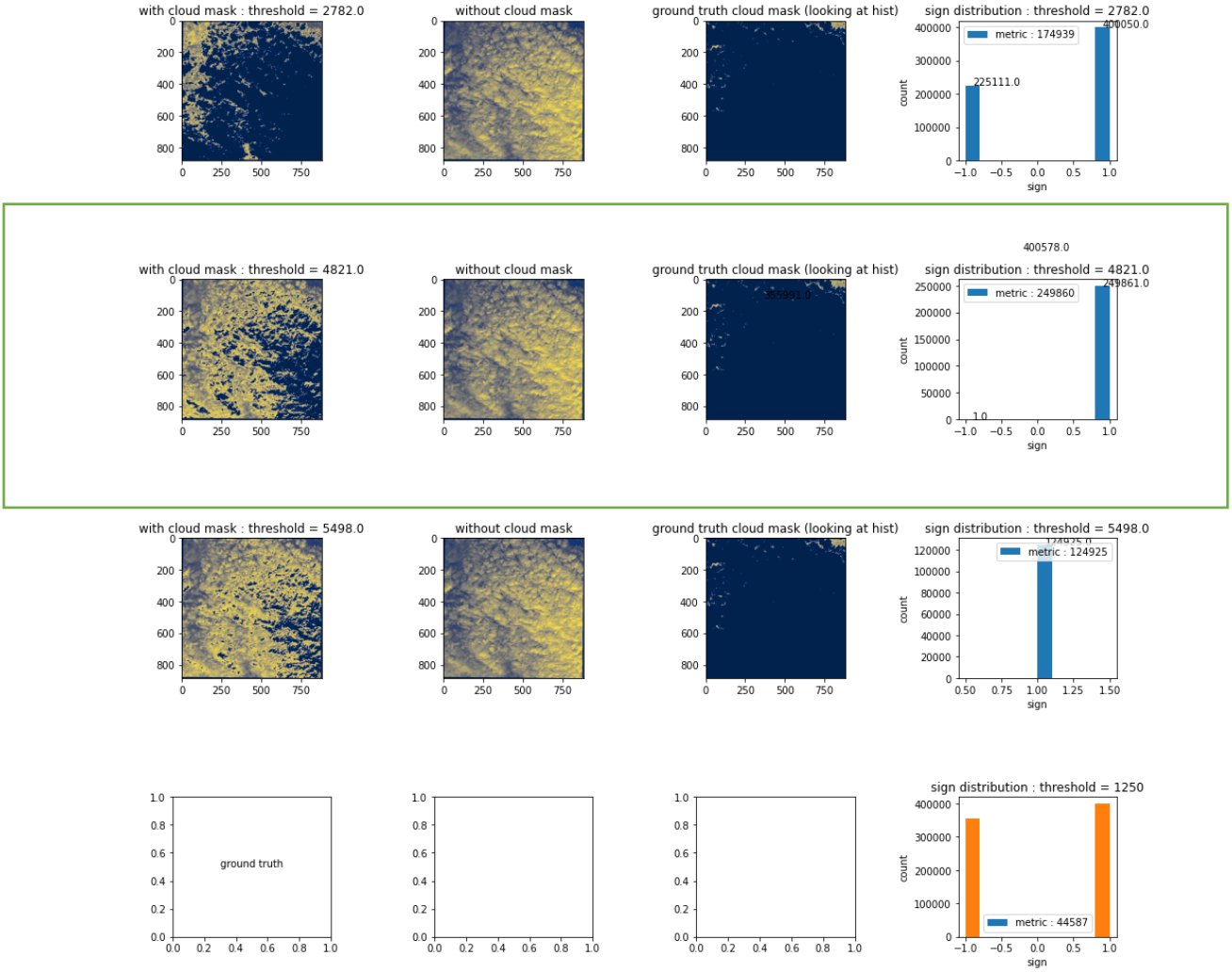

Fig.4. Sensitivity analysis of automatic cloud mask algorithm over blue_t = .95 (row 1) , .98 (row 2), .99 (row 3) for the cloudiest scene with blue band visualization. The green row was the threshold picked by the algorithm. Pictured is the image with applied cloud mask (first column), image without a cloud mask (second column), image with applied cloud mask if looking at the band histogram (considered as ground truth (threshold = 1250)) (third column) and histogram of distribution of signs of $\cos(\theta_{rel})$) (last column). The last row shows the histogram of distribution of signs for ground truth as comparison to the other thresholds.

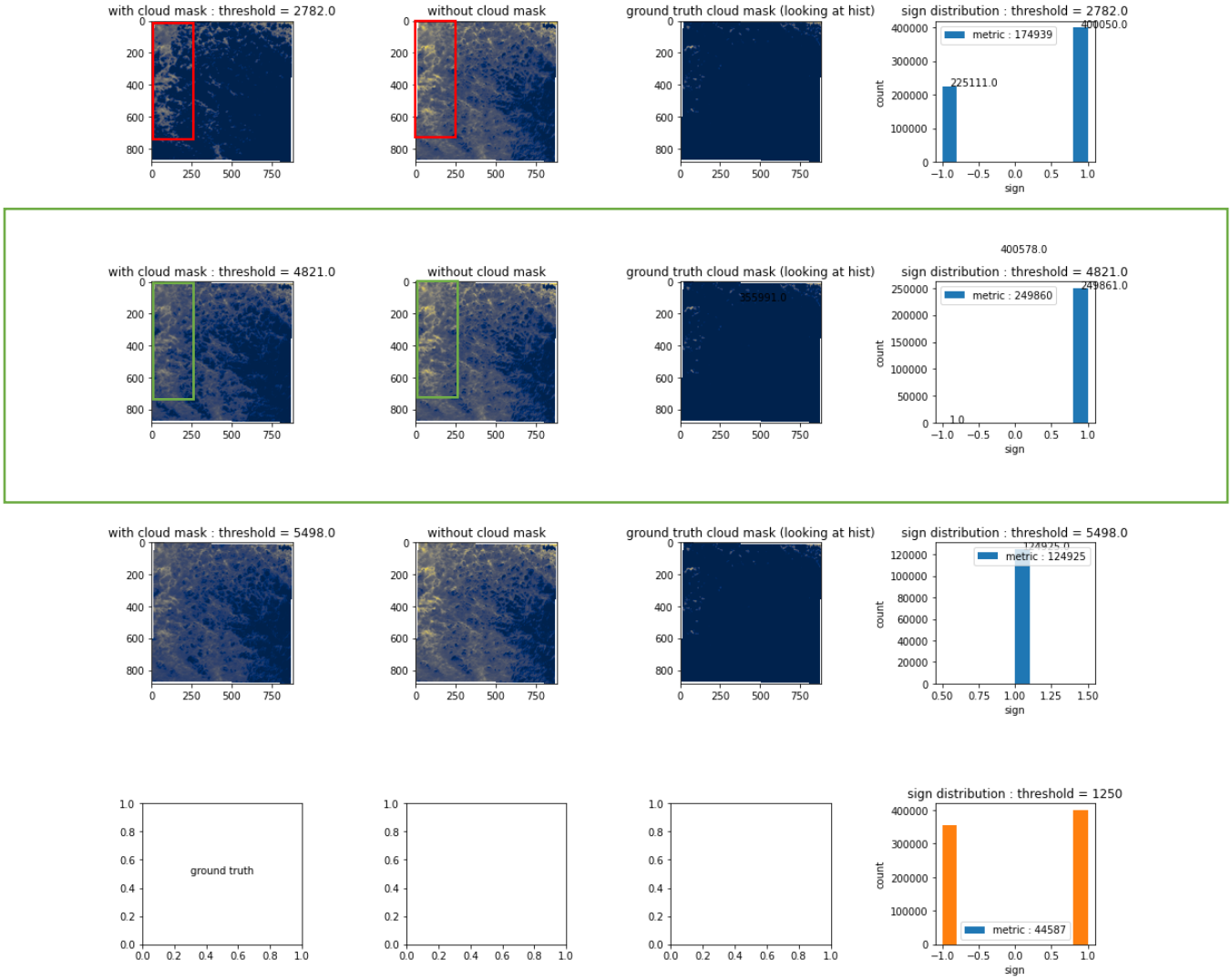

Fig.5. Sensitivity analysis of automatic cloud mask algorithm over blue_t = .95 (row 1) , .98 (row 2), .99 (row 3) for the cloudiest scene with NDSI visualization. The green row was the threshold picked by the algorithm. Pictured is the image with applied cloud mask (first column), image without a cloud mask (second column), image with applied cloud mask if looking at the band histogram (considered as ground truth(threshold = 1250)) (third column) and histogram of distribution of signs of $\cos(\theta_{rel})$) (last column). The last row shows the histogram of distribution of signs for ground truth as comparison to the other thresholds.

We can refer to Fig.6. to look at the distributions of $\cos(\theta_{rel}))_i, \forall\, i = 1,2, \ldots\ldots, z$ in the third column. As shown the first threshold, blue_t = .95, contains too much of the RVS surface. The 25th and 75th percentile of the distribution almost fully encompass the domain of the cosine function ([-1,1]). In addition, CV > 1 so the threshold cannot be used for the feasible set. Moving higher in to blue_t = .98 and blue_t = .99 both thresholds show values clumped closely together on the positive side of the x-axis, have most pixels in the cloud mask of the same surface (by the histogram in the center), and have CV < 1, indicating that the constraint on CV was meaningful

towards the problem. For further analysis on other images such as the greenest and brightest image refer to the Appendix.

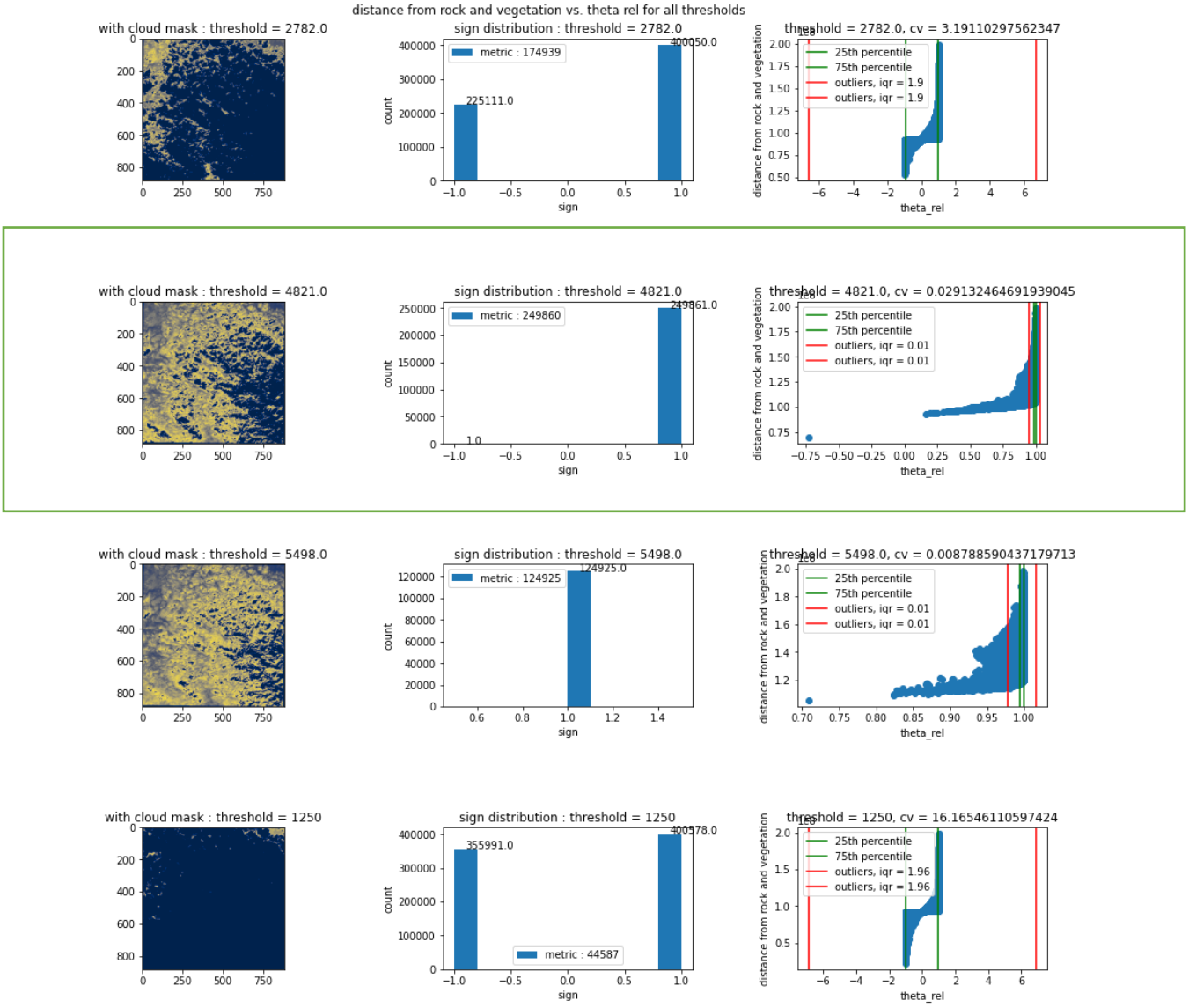

Fig.6. Sensitivity analysis of automatic cloud mask algorithm over blue_t = .95 (row 1) , .98 (row 2), .99 (row 3), a threshold of 1250 (ground truth) (row 4) for the greenest scene with blue band visualization. The green row was the threshold picked by the algorithm. Pictured is the image with applied cloud mask (first column), histogram of distribution of signs of $\cos(\theta_{rel}))$ (second column), and scatter plot of distribution of values of $\cos(\theta_{rel}))$ (last column).

Conclusions/Future Work

In conclusion, the automatic thresholding algorithm has made an improvement on manually finding the thresholds. Comparing the threshold for the blue band identified with the automatic thresholding algorithm to that which was found through looking at the band histogram we can see (in Fig.5.) that the cloud mask obtained from the histogram contains much more of the snow / RVS surface than the automatic algorithm where all but one of the values masked out are of the same surface.

In addition, the algorithm was useful in confirming the formula derived in [6] into a convex formulation. The cloud masks that contained more snow had much more values of different sign than those that contained few or no pixels at all contributing towards that surface (based on comparing columns 1 and 2 in Fig.5). The importance of the optimal solution for this problem is that we can find how many pixels in the image are in the surface tested. Negate the objective function and if all values are of the same sign we get $-(-|\sum sgn(\mathbf{cos}(\boldsymbol{\theta_{rel}}))|) = \begin{cases} |\sum \mathbf{1}|, \mathbf{cos}(\boldsymbol{\theta_{rel}})) > 0 \\ |\sum -\mathbf{1}|, \mathbf{cos}(\boldsymbol{\theta_{rel}})) < 0 \end{cases} = j = \#of\ pixels\ in\ surface.$

While the specific use case does not have all pixels of the same sign for the threshold picked (as shown by the histograms in Fig.4 - 6) this just means that a better threshold does exist which does not contain any pixels of different sign. Unfortunately, blue_t could not be evaluated for all values in [0,1) based on the way in which the formulation was solved so while there may exist a more optimal value and solution for the problem however the one pictured here could be considered the optimal value and solution of the discrete thresholds tested.

Future work could be to apply this algorithm in an image segmentation sense. If the thresholding algorithm could be used to find what pixels belong to a surface, then if we know all the surfaces of an image and their properties the formulation should be able to isolate the pixels belonging to each. Other future work could be extending the algorithm for multiple thresholds of multiple bands for cloud masking. An example could be to find what values of the blue band and SWIR1 band would find the densest clouds since blue band can detect dense cloud and SWIR1 can be used to detect snow, differentiating cloud from snow.

Ultimately, however, I would like to continue experimenting with the thresholding algorithm for just blue_t and its feasibility, working with larger images than 4x4 since this would be more realistic. Using 781456 4x4 images the code took about 159.3 seconds, more than 2 minutes, so there is some concern about the time for the algorithm to complete with larger images. An additional future work could be to find a way to speed up the algorithm to allow for faster times with larger images or more thresholds. This could be done with ideas like multithreading or just changing/transforming the formulation to a form that a polynomial-time solver (like Khachiyan's) can solve.

References

[1] Yang, Suwei, Massimo Lupascu, and Kuldeep S. Meel. "Predicting forest fire using remote sensing data and machine learning." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 35. No. 17. 2021.

[2] Cao, Quoc Dung, and Youngjun Choe. "Post-hurricane damage assessment using satellite imagery and geolocation features." arXiv preprint arXiv:2012.08624 (2020).

[3] Xu, Ke, et al. "DeepMask: An algorithm for cloud and cloud shadow detection in optical satellite remote sensing images using deep residual network." arXiv preprint arXiv:1911.03607 (2019).

[4] "Sentinel-2 L1C." Sentinel Hub, https://docs.sentinel-hub.com/api/latest/data/sentinel-2-l1c/.

[5] Rosa Coluzzi, Vito Imbrenda, Maria Lanfredi, Tiziana Simoniello,A first assessment of the Sentinel-2 Level 1-C cloud mask product to support informed surface analyses. Remote Sensing of Environment. Volume 217. 2018. Pages 426-443. ISSN 0034-4257. https://doi.org/10.1016/j.rse.2018.08.009.

[6] Aruma Ishida, Yu Oishi, Keitaro Morita, Keigo Moriwaki, Takashi Y. Nakajima. Development of a support vector machine based cloud detection method for MODIS with the adjustability to various conditions. *Remote Sensing of Environment*. Volume 205. 2018. Pages 390-407. ISSN 0034-4257. https://doi.org/10.1016/j.rse.2017.11.003.

[7] What is the Coefficient of Variation? UCLA : Statistical Consulting Group. FAQ: What is the coefficient of variation? (ucla.edu)

[8] "Coefficient of Variation." *ReadyRatios*, Audit IT, https://www.readyratios.com/reference/analysis/coefficient_of_variation.html.

[9] Boyd, S. "Ellipsoidal Methods." Stanford, Stanford, 1 Apr. 2018, https://web.stanford.edu/class/ee364b/.

[10] Strang, Gilbert (1 June 1987). "Karmarkar's algorithm and its place in applied mathematics". *The Mathematical Intelligencer*. 9 (2): 4–10. doi:10.1007/BF03025891. ISSN 0343-6993. MR 0883185. S2CID 123541868.

[11] Pin, Chong Edwin Kah, and Żak Stanislaw H. "Concepts from Geometry." *An Introduction to Optimization*, Wiley, Hoboken, 2013.

[12] Pin, Chong Edwin Kah, and Żak Stanislaw H. "Basics of Set-Constrained and Unconstrained Optimization." *An Introduction to Optimization*, Wiley, Hoboken, 2013.

[13] Das, Shagnik. “Continuity of Functions.” *Freie Universitat Berlin*, Freie Universitat Berlin, https://page.mi.fu-berlin.de/shagnik/notes/continuity.pdf.

[14] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2. (Publisher link).

[15] GDAL/OGR contributors (2022). GDAL/OGR Geospatial Data Abstraction software Library. Open Source Geospatial Foundation. URL https://gdal.org DOI: 10.5281/zenodo.5884351

[16] McKinney, Wes, et al. Data structures for statistical computing in python. Proceedings of the 9th Python in Science Conference. Vol. 445. 2010. Pages 51--56

Appendix - Figures

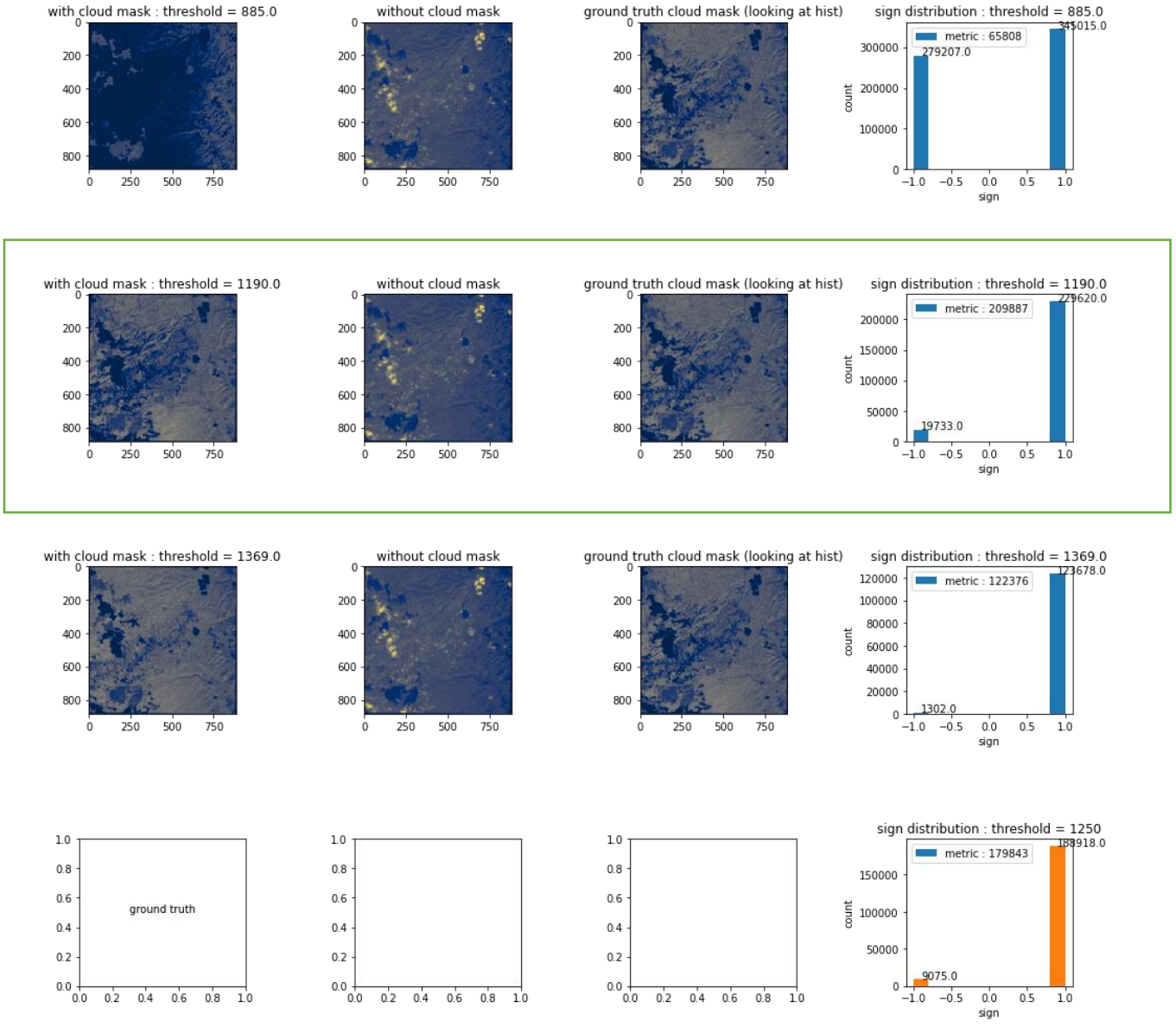

Fig.A1. Sensitivity analysis of automatic cloud mask algorithm over blue_t = .95 (row 1) , .98 (row 2), .99 (row 3) for the greenest scene with blue band visualization. The green row was the threshold picked by the algorithm. Pictured is the image with applied cloud mask (first column), image without a cloud mask (second column), image with applied cloud mask if looking at the band histogram (considered as ground truth(threshold = 1250)) (third column) and histogram of distribution of signs of $\cos(\theta_{rel})$) (last column). The last row shows the histogram of distribution of signs for ground truth as comparison to the other thresholds.

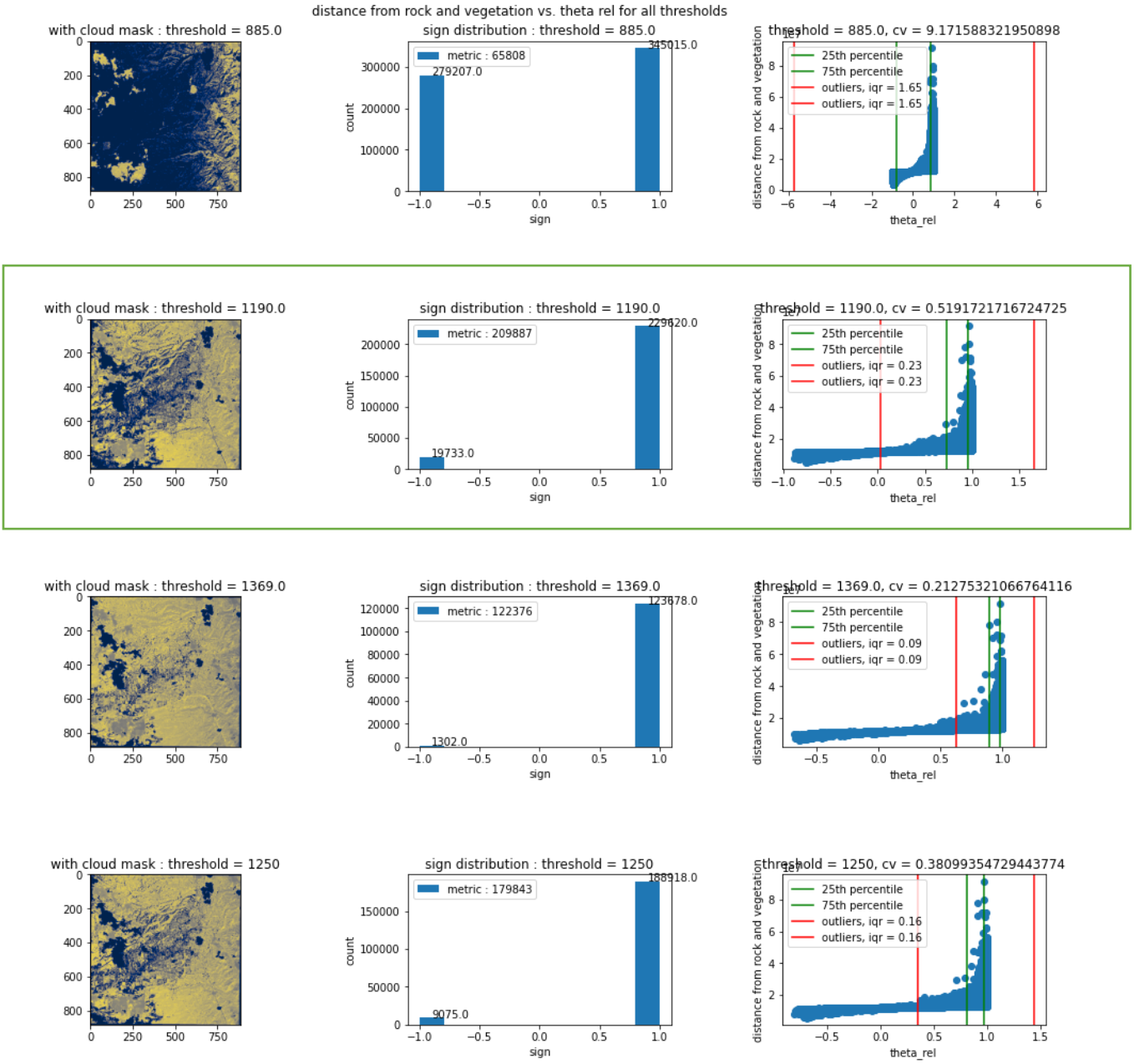

Fig.A2. Sensitivity analysis of automatic cloud mask algorithm over blue_t = .95 (row 1) , .98 (row 2), .99 (row 3), a threshold of 1250 (ground truth) (row 4) for the greenest scene with blue band visualization. The green row was the threshold picked by the algorithm. Pictured is the image with applied cloud mask (first column), histogram of distribution of signs of $\cos(\theta_{rel}))$ (second column), and scatter plot of distribution of values of $\cos(\theta_{rel}))$ (last column).

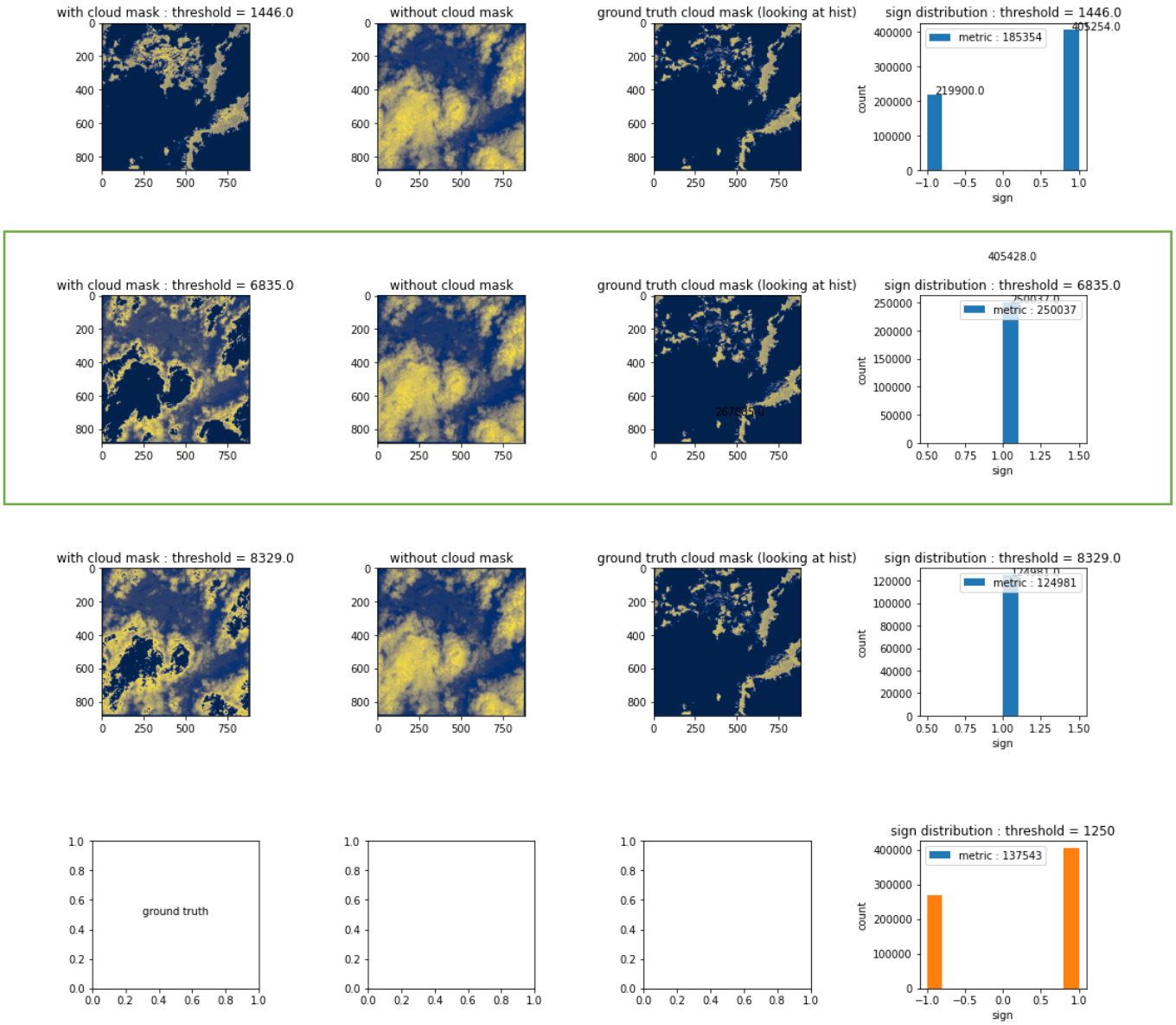

Fig.A3. Sensitivity analysis of automatic cloud mask algorithm over blue_t = .95 (row 1) , .98 (row 2), .99 (row 3) for the brightest scene with blue band visualization. The green row was the threshold picked by the algorithm. Pictured is the image with applied cloud mask (first column), image without a cloud mask (second column), image with applied cloud mask if looking at the band histogram (considered as ground truth(threshold = 1250)) (third column) and histogram of distribution of signs of $\cos(\theta_{rel})$) (last column). The last row shows the histogram of distribution of signs for ground truth as comparison to the other thresholds.

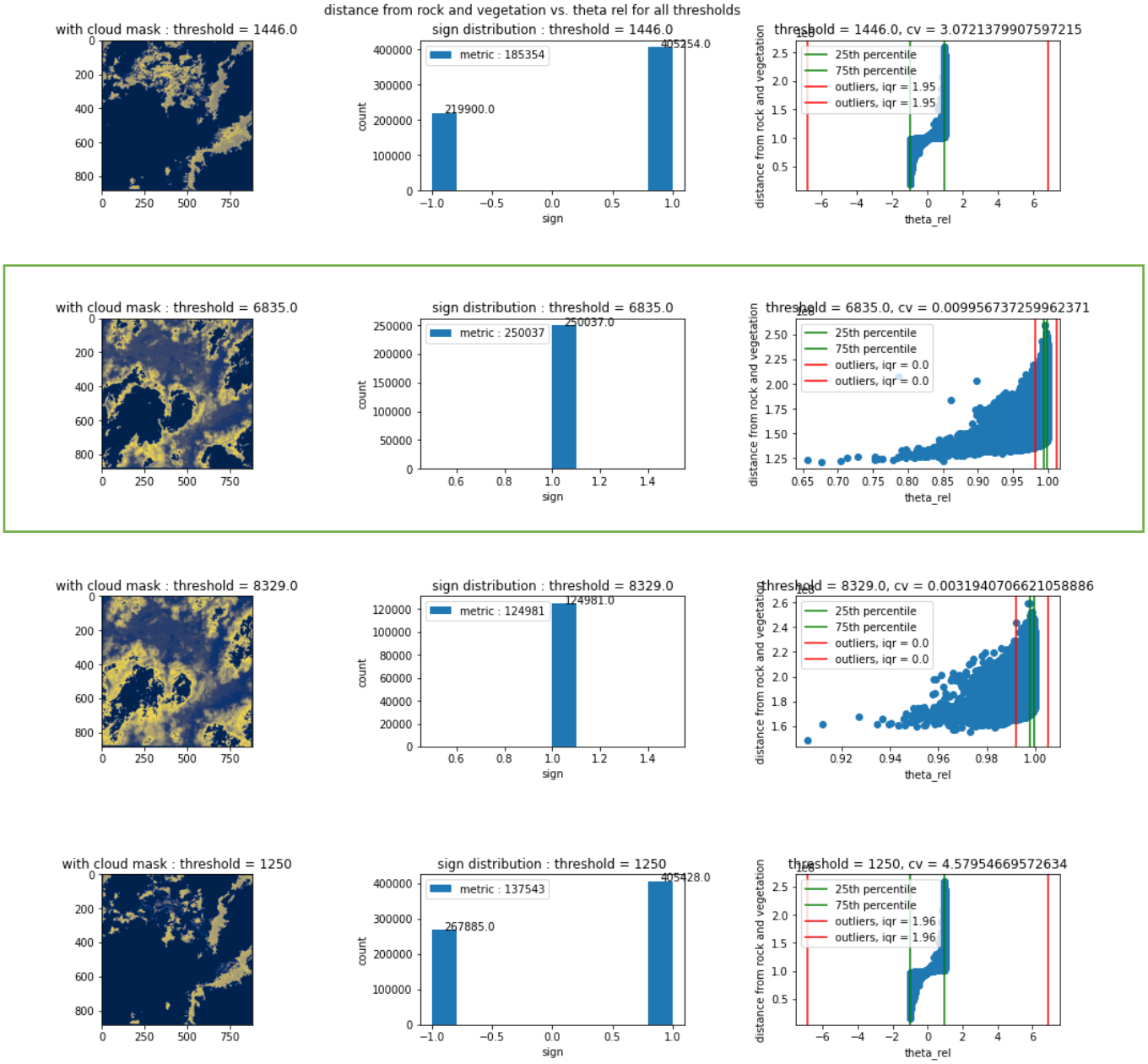

Fig.A4. Sensitivity analysis of automatic cloud mask algorithm over blue_t = .95 (row 1) , .98 (row 2), .99 (row 3), a threshold of 1250 (ground truth) (row 4) for the brightest scene with blue band visualization. The green row was the threshold picked by the algorithm. Pictured is the image with applied cloud mask (first column), histogram of distribution of signs of $\cos(\theta_{rel}))$ (second column), and scatter plot of distribution of values of $\cos(\theta_{rel}))$ (last column).

Appendix – Code *** github can be sent

Convex Toy Dataset Generator

```
 -*- coding: utf-8 -*-
"""toy_dataset_convex_generation.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1j6P12AcWFkxdx-liwHGHD-vH0FhKIbKO

# **Installations and Dependencies**
"""

!apt-get update
!apt-get install libgdal-dev -y
!apt-get install python-gdal -y
!apt-get install python-numpy python-scipy -y
!pip install rasterio
!pip install rioxarray
!pip install geemap
!pip install geojson
!pip install geopandas

"""# **Aligning Dataset**"""

import matplotlib.pyplot as plt
import rioxarray as rxr
import rasterio
from rasterio.crs import CRS
import glob
import subprocess
import urllib, geojson,subprocess,gdal
import geopandas as gpd
import shapefile
import os
import ee
import json
import numpy as np
import pandas as pd
import geemap
from rasterio.plot import show
```

```
import math
import osr

from google.colab import drive

drive.mount('/content/drive')

''' gis_projects is where the original dataset and geojson are stored'''
'''in order to align you have to create a few folders in your drive. create
folders with these names:

- revised_dataset_projected
- revised_dataset_resolution
- new_dataset

'''

geojson_loc = '/content/drive/MyDrive/gis_projects/santafe_crop.geojson'
dataset_loc = '/content/drive/MyDrive/gis_projects'
projected_dataset_loc = '/content/drive/MyDrive/revised_dataset_projected'
resolution_dataset_loc = '/content/drive/MyDrive/revised_dataset_resolution'

"""# Summary

For the first step in aligning the dataset I reprojected files in dataset to
EPSG:32316 because this is a UTM CRS, which was recommended by the professor, but
also because this projection was found in the dataset so it appears that the
coordinate system is native to the images. After that, I changed the resolution
of each image to 20m, since this was the maximum, such that no data will be lost.
e.g. if an image at 20m was changed to a resolution of 10m the data may be lost.
Additionally, I changed the resolution to this value such that less RAM is used
in calulations involnving the dataset. The next step was then to convert the
geojson to a shapefile with the CRS specified (EPSG:32613) and crop to cutline
"""

#for loop to create a copy and paste script for command line to reproject files
to utm (epsg:32613)

i = 0
for name in glob.glob(dataset_loc+'/*.tif'):
    print('gdalwarp -t_srs EPSG:32613' + " " +name+" "
+projected_dataset_loc+'/i'+os.path.basename(name)[:-4]+str(i)+'utm'+'.tif')
    i+=1
```

```
#NOTE: Unfortunately I have been trying for about 10 hours in total to resolve
issues with gdalwarp to no avail.
#The only way that I was able to reproject the dataset was to use the command
line via colab pro.
#My drive is mounted so feel free to use those files or run all other portions of
this code. They should work.
#Another option could be to skip the aligning dataset portion and use the
new_dataset I have stored in my drive.

'''finding crs for the files projected to utm (as a check)'''

for name in glob.glob(projected_dataset_loc+'/*.tif'):
    dem = rxr.open_rasterio(name,masked=True).squeeze()
    print(name, dem.rio.crs)

'''resolution exploration'''

#getting affine transform of tif
#upx is upper pixel x
#xes is pixel width
#xkew is shear in x direction
summation = 0
length = 0
maximum = 0
for name in glob.glob(projected_dataset_loc+'/*.tif'):
    ds = gdal.Open(name)
    upx, xres, xskew, upy, yskew, yres = ds.GetGeoTransform()
    print('the resolution of ' + os.path.basename(name) + " (" + str(xres) +
","+str(yres)+")." )
    if(xres>maximum):
        maximum = xres
print(" ")
print("the max resolution is " + str(maximum))

'''changing resolution of all reprojected images to max resolution (downsampling
will ruin features)'''

i = 0
for name in glob.glob(projected_dataset_loc+'/*.tif'):
    args = ['gdal_translate','-of','GTiff','-
tr',str(maximum),str(maximum),name,resolution_dataset_loc+'/i'+os.path.basename(n
ame)[:-4]+str(i)+'res'+'.tif']
    subprocess.Popen(args)
    i+=1
```

```
'''creating dataset of original names'''

names = []
for name in glob.glob(dataset_loc+'/*.tif'):
    names.append(os.path.basename(name))

'''creating shapefile out of geojson and projecting it to epsg:32613'''

!ogr2ogr /content/drive/MyDrive/gis_projects/output.shp -t_srs
'http://spatialreference.org/ref/epsg/32613/'
/content/drive/MyDrive/gis_projects/santafe_crop.geojson

'''cropping images to shapefile extent and changing name back to original'''

i = 0
for name in glob.glob(resolution_dataset_loc+'/*.tif'):
    print(name)
    args = ['gdalwarp','-
cutline','/content/drive/MyDrive/gis_projects/output.shp','-crop_to_cutline','-
dstalpha',name,'/content/drive/MyDrive/new_dataset/i'+names[i]]
    subprocess.Popen(args)
    i+=1

"""# **Analyzing Dataset**"""

#band 4 is nir
#band 1 is red
#band 7 is alpha

#loading in data

#for each file, the data is opened and the image height, width, bands,
# and alpha band measurement are loaded into a stack.

dataset_array = []

for name in glob.glob("/content/drive/MyDrive/new_dataset/*.tif"):

  data = rasterio.open(name)

  #normalizing alpha band just in case

  alpha_band = np.mean(data.read()[6]/np.max(data.read()[6]))

  dataset_array.append([data.height,data.width,data.read(),alpha_band])
```

```
data_array = np.stack(dataset_array)



''' finding the range of values for each respective band '''

#creating array of dataset bands (in hindsight it may have been easier to use a
column index)

dataset_bands = data_array[:,2]
band_distribution = []

#separating each band

for i in range(len(dataset_bands)):
  band_distribution.append([dataset_bands[i][j] for j in range(7)])
band_distribution = np.array(band_distribution)

'''creating histogram of values of images picked from aligned dataset'''

fig, axs = plt.subplots(7,1)

fig.set_figheight(15)
fig.set_figwidth(15)
fig.tight_layout()
bands = {0:'red',1:'green',2:'blue',3:'nir',4:'swir1',5:'swir2',6:'alpha'}
for i in range(7):
  axs[i].axvline(x = 1250, color = 'r', linestyle = "dashed")
  axs[i].hist(band_distribution[:,i].flatten().squeeze(),bins=25)

  axs[i].set_title("band " + str(i+1) + ": " + bands[i])
  axs[i].set_ylabel('frequency')

'''computing cloud mask'''

def create_cloud_mask(img_array):

  '''creating cloud mask from blue band (band 3) and swir band (band 5).
  blue band identifies dense clouds with high reflectance and
  swir band differentiates snow from cloud because snow can have a high
  reflectance in blue band but low reflectance in swir1 band'''
```

```
  '''
  INPUTS:

  img_array (.tif as array): the .tif you want to apply a cloud mask on

  OUTPUTS:

  cloud_mask_matrix: matrix with zeros (invalid data) at the places where a cloud
is predicted

  '''


  #thresholding based on histogram. It took some testing and educated guesses but
these were the best values
  #using np.where to find the points where over threshold

  blue = np.where(img_array[2] >=1250)
  swir1 = np.where(img_array[4] >=3000)

  #finding where the points intersect between those that surpass the swir1 and
blue

  df_blue = pd.DataFrame({'blue':[(blue[0][i],blue[1][i]) for i in
range(len(blue[0]))]})
  df_swir1 = pd.DataFrame({'swir1':[(swir1[0][i],swir1[1][i]) for i in
range(len(swir1[0]))]})
  df_merged = df_blue.merge(df_swir1,left_on = 'blue',right_on ='swir1')

  #creating cloud mask matrix starting with an array of 1s (valid data), putting
0s where a cloud is predicted

  cloud_mask_matrix = np.ones((np.shape(img_array[2])[0],
np.shape(img_array[2])[1])).astype('uint16')
  print(len(df_merged['blue']))
  for i in df_merged['blue']:
    arr = np.array(i)
    cloud_mask_matrix[arr[0]][arr[1]] = 0
  return cloud_mask_matrix

def create_cloud_arr(image_stack):
  ''' applying create_cloud_mask function'''

  '''
  INPUTS:
```

```
  image_stack: np.stack of .tifs as arrays where each array has a length of n,
  the number of bands

  OUTPUTS:

  alpha_band_arr: updated alpha band with cloud mask added
  thresholded_points: number of points predicted to be part of the cloud from the
cloud mask.

  '''

  alpha_band_arr = []
  num_thresholded_points = []
  for i in range(len(image_stack)):

    #creating cloud mask for an individual image

    cloud_mask = create_cloud_mask(image_stack[i])

    #finding the number of points predicted in the image as cloud

    num_thresholded_points.append(np.shape(cloud_mask)[0]*np.shape(cloud_mask)[1]
- np.sum(cloud_mask))

    #final result

    alpha_band_arr.append(cloud_mask*image_stack[i][6].astype('bool').astype('uin
t16'))
  return alpha_band_arr,num_thresholded_points

#creating cloud masks for each file and storing in an array

cloud_mask_arr, num_thresholded_points= create_cloud_arr(data_array[:,2])

#This function, rescale_img, was specifically taken from Professor Karra from his
lecture on Week 5.
#there is no link to go to but if you are in the Remote Sensing class you can
find the function through Teams and/or Google Drive

def rescale_img(img, min_val=0.0, max_val=1.0, dtype=np.float32, pmin=0.0,
pmax=100.0):
    vmin, vmax = np.nanpercentile(img, pmin), np.nanpercentile(img, pmax)


    img_rescale = ((img - vmin) * (1.0 / (vmax - vmin) * max_val)).astype(dtype)
```

```
    np.clip(img_rescale, min_val, max_val, out=img_rescale)

    return img_rescale

#helper function to shorthand rescale_img with specific parameters

image_rescaled = lambda x: rescale_img(x,pmin = 0.2,pmax = 99.8)


def show_image(picture_num,band):

  '''showing image with a comparison between original and cloud mask'''

  '''

  INPUTS:

  picture_num: (int) index of image stack loaded in (used to isolate individual
images)
  band: (int) band of image to use e.g. band = 0 is the red band

  '''



  alpha_band = (data_array[:,2][picture_num][6].astype('bool')).astype('uint16')
  data_validation = cloud_mask_arr[picture_num]*alpha_band
  cloud_mask_picture =
image_rescaled(data_array[:,2][picture_num][band]*(data_validation))
  orig_image = data_array[:,2][picture_num][band]*(alpha_band)

  fig, axs = plt.subplots(1,2)
  fig.set_figheight(15)
  fig.set_figwidth(15)

  fig.tight_layout()

  #plots image with cloud mask

  axs[0].imshow(cloud_mask_picture,cmap = 'cividis')
  axs[0].set_title('with cloud mask')

  #plots image without cloud mask

  axs[1].imshow(rescale_img(orig_image,pmin=0.2, pmax=99.8),cmap = 'cividis')
  axs[1].set_title('without cloud mask')
```

```
  return orig_image


#testing cloud mask

#show_image(picture_num =59, band = 2)

"""PART 1 FUNCTIONS"""

def apply_cloud_mask(picture_num,band):

  '''
  applying cloud mask to picture num and band

  '''

  '''
  INPUTS:

  picture_num: (int) index of image stack loaded in (used to isolate individual
images)
  band: (int) band of image to use e.g. band = 0 is the red band

  OUTPUTS:

  returns image band with applied cloud mask

  '''

  alpha_band = (data_array[:,2][picture_num][6].astype('bool')).astype('uint16')
  data_validation = cloud_mask_arr[picture_num]*alpha_band
  return data_array[:,2][picture_num][band]*data_validation




def calculate_greenest_scene(cloud_mask_arr,max_or_mean = 'mean'):

  '''calculates the image with the best ndvi based on the max or mean of the
image, default mean.
  The image will have a cloud mask derived from the swir and blue band so that
  invalid data will not be part of the mean. I personally defined greenest scene
as the greatest mean(ndvi)
```

```
  because if we define only on max(NDVI) then there is a possibility that only a
small portion of the image is green.
  I have it so that you can specify max or mean since people have different
interpretations of greenest scene. Note:
  if you are worried about the cloud mask for the max option, it will have no
effect. If a scene truly has an area where there is max NDVI,
  it will not have a high reflectance in the swir and blue band and therefore not
be invalid by the cloud mask.'''

  '''
  INPUTS:

  cloud_mask_arr : (array) array of cloud masks which can be made by using the
create_cloud_arr function
  max_or_mean : (string) if this variable is equivalent to 'mean' it computes the
mean. Elsewise it computes the max.

  OUTPUTS:

  greenest_scene_val: (int) the best NDVI based on whichever option chosen (max
or mean)
  greenest_scene_picture_num: (int) the picture for which there is the best NDVI
ased on whichever option chosen (max or mean)
  '''


  #computing ndvi for each image. There is an initial value of -10000 for NDVI
and -1 for the picture number.
  #the picture number is just used as an index to isolate a specific image in the
stack of images imported.
  #If an ndvi is greater than the last NDVI that was appended to maximum then it
is appended as well as its corresponding picture number.
  #In this way the last entry of maximum includes the max ndvi and picture with
max ndvi

  maximum = []
  maximum.append([-10000,-1])
  for i in range(len(cloud_mask_arr)):

    #getting bands for calculation

    red = data_array[:,2][i][0]
    nir = data_array[:,2][i][3]

    #calculating ndvi
```

```
    aggregated_ndvi = 0
    ndvi = ((nir-red)/(nir+red))
    if(max_or_mean == 'max'):
      aggregated_ndvi = np.nanmax(ndvi)
    else:
      aggregated_ndvi = np.nanmean(ndvi)

    #appending max ndvi if greater than last value of maximum

    if(aggregated_ndvi>maximum[len(maximum)-1][0]):
      maximum.append([aggregated_ndvi,i,ndvi])


  greenest_scene_val = maximum[len(maximum)-1][0]
  greenest_scene_picture_num = maximum[len(maximum)-1][1]
  return greenest_scene_val,greenest_scene_picture_num


#snow

def calculate_snowiest_scene(cloud_mask_arr,max_or_mean = 'mean'):

  '''calculates the image with the best ndsi based on the max or mean of the
image, default mean.
  The image will have a cloud mask derived from the swir and blue band so that
  invalid data will not be part of the mean. I personally defined greenest scene
as the greatest mean(ndsi)
  because if we define only on max(NDSI) then there is a possibility that only a
small portion of the image is green.
  I have it so that you can specify max or mean since people have different
interpretations of greenest scene. Note:
  if you are worried about the cloud mask for the max option, it should have no
effect. If a scene truly has an area where there is max NDSI,
  then it will have a low reflectance in the swir band and therefore not be
invalid by the cloud mask.'''

  '''
  INPUTS:

  cloud_mask_arr : (array) array of cloud masks which can be made by using the
create_cloud_arr function
  max_or_mean : (string) if this variable is equivalent to 'mean' it computes the
mean. Elsewise it computes the max.

  OUTPUTS:
```

```
  snowiest_scene_val: (int) the best NDSI based on whichever option chosen (max
or mean)
  snowiest_scene_picture_num: (int) the picture for which there is the best NDSI
based on whichever option chosen (max or mean)
  '''

  #computing ndsi for each image. There is an initial value of -10000 for NDsI
and -1 for the picture number.
  #the picture number is just used as an index to isolate a specific image in the
stack of images imported.
  #If an ndsi is greater than the last NDsI that was appended to maximum then it
is appended as well as its corresponding picture number.
  #In this way the last entry of maximum includes the max ndsi and picture with
max ndsi

  maximum = []
  maximum.append([-10000,-1])
  for i in range(len(cloud_mask_arr)):

    #getting bands for calculation

    green = apply_cloud_mask(i,1)
    swir1 = apply_cloud_mask(i,4)

    #calculating ndsi

    aggregated_ndsi = 0
    ndsi = ((green-swir1)/(green+swir1))
    if(max_or_mean == 'max'):
      aggregated_ndsi = np.nanmax(ndsi)
    else:
      aggregated_ndsi = np.nanmean(ndsi)

    #appending max ndsi if greater than last value of maximum


    if(aggregated_ndsi>maximum[len(maximum)-1][0]):
      maximum.append([aggregated_ndsi,i,ndsi])


  snowiest_scene_val = maximum[len(maximum)-1][0]
  snowiest_scene_picture_num = maximum[len(maximum)-1][1]
  return snowiest_scene_val,snowiest_scene_picture_num
```

```
def calculate_max_brightness(cloud_mask_arr,max_or_mean = 'mean'):

  '''calculates the image with the best brightness based on the max or mean of
the image, default mean.
   I personally defined greenest scene as the greatest mean(ndsi)
  because if we define only on max(brightness) then there is a possibility that
only a small portion of the image is very bright.
  I have it so that you can specify max or mean since people have different
interpretations of greenest scene.'''

  '''
  INPUTS:

  cloud_mask_arr : (array) array of cloud masks which can be made by using the
create_cloud_arr function
  max_or_mean : (string) if this variable is equivalent to 'mean' it computes the
mean. Elsewise it computes the max.

  OUTPUTS:

  snowiest_scene_val: (int) the best NDSI based on whichever option chosen (max
or mean)
  snowiest_scene_picture_num: (int) the picture for which there is the best NDSI
based on whichever option chosen (max or mean)
  '''

  maximum = []
  maximum.append([-10000,-1])
  for i in range(len(cloud_mask_arr)):

    red = data_array[:,2][i][0]
    green = data_array[:,2][i][1]
    blue = data_array[:,2][i][2]
    brightness = 0
    visible_light = (red+green+blue)/3
    if(max_or_mean == 'max'):
      brightness = np.nanmax(visible_light)
    else:
      brightness = np.nanmean(visible_light)

    if(brightness>maximum[len(maximum)-1][0]):
      maximum.append([brightness,i,visible_light])
  brightness_max = maximum[len(maximum)-1][0]
  max_brightness_picture_num = maximum[len(maximum)-1][1]
  return brightness_max,max_brightness_picture_num
```

```
"""GREENEST_SCENE"""

arr = []
for name in glob.glob('/content/drive/MyDrive/new_dataset/*.tif'):
  arr.append(name)

greenest_scene_val,greenest_scene_picture_num =
calculate_greenest_scene(cloud_mask_arr)
green_image = show_image(greenest_scene_picture_num, band = 1)
print(' ')
print(f'the picture with max ndvi is
{os.path.basename(arr[greenest_scene_picture_num])} with {greenest_scene_val}.')
print(' ')

"""SNOWIEST_SCENE"""

snowiest_scene_val,snowiest_scene_picture_num=
calculate_snowiest_scene(cloud_mask_arr)
snow_image = show_image(snowiest_scene_picture_num, band = 1)
print(' ')
print(f'the picture with max ndsi is
{os.path.basename(arr[snowiest_scene_picture_num])} with {snowiest_scene_val}.')
print(' ')

"""CLOUDIEST_IMAGE"""

cloudiest_image_index = np.where(num_thresholded_points ==
max(num_thresholded_points))[0][0]
cloudiest_image_value = num_thresholded_points[cloudiest_image_index]
cloudy_image = show_image(cloudiest_image_index, band = 4)
print(' ')
print(f'the cloudiest picture is {os.path.basename(arr[cloudiest_image_index])}
with sum of {int(cloudiest_image_value)}.')
print(' ')

"""BRIGHTEST_IMAGE"""

brightness_max,max_brightness_picture_num =
calculate_max_brightness(cloud_mask_arr)
show_image(max_brightness_picture_num, band = 2)
print(' ')
print(f'the picture with max brightness is
{os.path.basename(arr[max_brightness_picture_num])} with {brightness_max}.')
print(' ')
```

```
"""Generating Toy *Dataset*"""

#edit picture num to change image to do a 4x4 window

picture_num_used = cloudiest_image_index

reference = data_array[picture_num_used]

np.save('/content/drive/MyDrive/toy_dataset_reference', reference)

np.shape(reference[2])

def sliding_window(image, stepSize, windowSize):
  '''
  Computes sliding window across image

  INPUTS:

  image : (array) image being windowed
  stepSize : (int) value in which the window moves horizontally after one window
  windowSize : (tuple) size of window

  OUTPUTS:

  set of windows across image
  '''
  for y in range(0, image.shape[0], stepSize):
    for x in range(0, image.shape[1], stepSize):
      yield [x, y, image[y:y + windowSize[1], x:x + windowSize[0]]]

def extractFeatures(window):
  '''
  extract features from window

  INPUTS:

  window: (array) : window containing positions and values

  OUTPUTS:

  features ((x,y),value)
  '''
  return (window[0], window[1]), window[2]
```

```
#computing 4x4 window across image and getting features

feature_lookup = []
orig_image = show_image(picture_num_used, band = 4)
features = np.zeros((len(orig_image)**2,7,4,4))
for band_idx,band in enumerate([0,1,2,3,4,5,6]):
  orig_image = show_image(picture_num_used, band = band)
  windows = sliding_window(orig_image, 4, (4, 4))
  for idx, window in enumerate(windows):
    point, featureVector = extractFeatures(window)
    feature_lookup.append(point)
    features[idx,band_idx] = np.array(featureVector)

''' finding the range of values for each respective band '''

#creating array of dataset bands (in hindsight it may have been easier to use a
column index)

dataset_bands = features[:,2]
band_distribution = []

#separating each band

for i in range(len(dataset_bands)):
  band_distribution.append([dataset_bands[i][j] for j in range(4)])
band_distribution = np.array(band_distribution)

values, counts = np.unique(features, return_counts=True)

'''creating histogram of values of images picked from toy dataset'''

fig, axs = plt.subplots(4,1)

fig.set_figheight(15)
fig.set_figwidth(15)
fig.tight_layout()
bands = {0:'red',1:'green',2:'blue', 3:'alpha'}
for i in range(4):

  axs[i].hist(band_distribution[:,i].flatten().squeeze(),bins=25)

  axs[i].set_title("band " + str(i+1) + ": " + bands[i])
  axs[i].set_ylabel('frequency')

np.save('/content/drive/MyDrive/toy_dataset', features)
```

Convex Toy Dataset Solver

```
# -*- coding: utf-8 -*-
"""convex_toy_dataset_exploration.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/14A2TIoiy26ZHoVIGbW9we8LDlFOkpOWo

<h1><i>An Automatic Thresholding Algorithm for Cloud Masks</i></h1>

<h3><strong>Imports</strong></h3>
"""

!apt-get update
!apt-get install libgdal-dev -y
!apt-get install python-gdal -y
!apt-get install python-numpy python-scipy -y
!pip install rasterio
!pip install rioxarray
!pip install geemap
!pip install geojson
!pip install geopandas

import matplotlib.pyplot as plt
import rioxarray as rxr
import rasterio
from rasterio.crs import CRS
import glob
import subprocess
import urllib, geojson,subprocess,gdal
import geopandas as gpd
import shapefile
import os
import ee
import json
import numpy as np
import pandas as pd
import geemap
```

```
from rasterio.plot import show
import math
import osr
import time

"""<h3><strong>Loading</strong></h3>"""

from google.colab import drive

drive.mount('/content/drive')

t0 = time.time()


toy_dataset = np.load("/content/drive/MyDrive/toy_dataset.npy")
reference = np.load('/content/drive/MyDrive/toy_dataset_reference.npy',
allow_pickle = True)

print(np.shape(toy_dataset))

num_images = np.shape(toy_dataset)[0]

"""<h3><strong>adding ndsi and ndvi as bands to toy dataset</strong></h3>"""

ndsi = np.reshape((toy_dataset[:,2] - toy_dataset[:,4]) /
(toy_dataset[:,2]+toy_dataset[:,4]), (num_images,1,4,4))
ndvi = np.reshape((toy_dataset[:,3] - toy_dataset[:,0]) /
(toy_dataset[:,3]+toy_dataset[:,0]), (num_images,1,4,4))

toy_dataset = np.hstack((toy_dataset,ndvi))
toy_dataset = np.hstack((toy_dataset,ndsi))

"""<h3><strong>adding ndsi and ndvi as bands to reference image for toy
dataset</strong></h3>"""

reference_image_data = reference[2]
reference_img_shape = np.shape(reference_image_data)

#add ndvi and ndsi bands to reference

ndsi = np.reshape((reference_image_data[2] - reference_image_data[4]) /
(reference_image_data[2]+reference_image_data[4]),
(1,reference_img_shape[1],reference_img_shape[2]))
```

```
ndvi = np.reshape((reference_image_data[3] - reference_image_data[0]) /
(reference_image_data[3]+reference_image_data[0]),
(1,reference_img_shape[1],reference_img_shape[2]))

reference_image_data = np.append(reference_image_data, ndvi,0)
reference_image_data = np.append(reference_image_data, ndsi,0)

np.shape(reference_image_data)

"""<h3><strong>Band Histogram</strong></h3>"""

''' finding the range of values for each respective band '''

#creating array of dataset bands (in hindsight it may have been easier to use a
column index)

dataset_bands = toy_dataset[:,2]
band_distribution = []

#separating each band

for i in range(len(dataset_bands)):
  band_distribution.append([toy_dataset[i][j] for j in range(9)])
band_distribution = np.array(band_distribution)



np.shape(band_distribution)[1]-1

'''creating histogram of values of images picked from aligned dataset'''

fig, axs = plt.subplots(np.shape(band_distribution)[1],1)

fig.set_figheight(15)
fig.set_figwidth(15)
fig.tight_layout()
bands = {0:'red',1:'green',2:'blue', 3:'nir', 4:'swir1', 5:'swir2', 6:'alpha',
7:'ndvi', 8:'ndsi'}

for i in range(np.shape(band_distribution)[1]):

  axs[i].hist(band_distribution[:,i].flatten().squeeze(),bins=25)

  axs[i].set_title("band " + str(i+1) + ": " + bands[i])
  axs[i].set_ylabel('frequency')
```

```
def create_cloud_mask(img_array, blue_threshold):

  '''creating cloud mask from blue band (band 3) and swir band (band 5).
  blue band identifies dense clouds with high reflectance and
  swir band differentiates snow from cloud because snow can have a high
  reflectance in blue band but low reflectance in swir1 band'''

  '''
  INPUTS:

  img_array (.tif as array): the .tif you want to apply a cloud mask on

  OUTPUTS:

  cloud_mask_matrix: matrix with zeros (invalid data) at the places where a cloud
is predicted

  '''

  blue = np.where(img_array > blue_threshold)



  #finding where the points intersect between those that surpass the swir1 and
blue

  df_blue = pd.DataFrame({'blue':[(blue[0][i],blue[1][i]) for i in
range(len(blue[0]))]})

  #creating cloud mask matrix starting with an array of 1s (valid data), putting
0s where a cloud is predicted

  cloud_mask_matrix = np.ones((np.shape(img_array)[0],
np.shape(img_array)[1])).astype('uint16')

  for i in df_blue['blue']:
    arr = np.array(i)
    cloud_mask_matrix[arr[0]][arr[1]] = 0
  return cloud_mask_matrix

def rescale_img(img, min_val=0.0, max_val=1.0, dtype=np.float32, pmin=0.0,
pmax=100.0):
    vmin, vmax = np.nanpercentile(img, pmin), np.nanpercentile(img, pmax)
```

```
    img_rescale = ((img - vmin) * (1.0 / (vmax - vmin) * max_val)).astype(dtype)
    np.clip(img_rescale, min_val, max_val, out=img_rescale)

    return img_rescale

#helper function to shorthand rescale_img with specific parameters

image_rescaled = lambda x: rescale_img(x,pmin = 0.2,pmax = 99.8)


def show_image(band, cloud_mask_arr, data_array, threshold,
ground_truth_cloud_mask_arr):

  '''showing image with a comparison between original and cloud mask'''

  '''

  INPUTS:

  picture_num: (int) index of image stack loaded in (used to isolate individual
images)
  band: (int) band of image to use e.g. band = 0 is the red band

  OUTPUTS:

  cloud_mask_picture : picture with cloud mask in specified band and threshold on
blue band
  no_cloud_mask_picture : picture with no cloud mask in specified band
  ground_truth_cloud_mask_picture : picture with cloud mask in specified band and
ground truth threshold on blue band (1250)


  '''



  alpha_band = (data_array[6].astype('bool')).astype('uint16') #edit this. i
believe alpha band is band 7

  data_validation = cloud_mask_arr * alpha_band
  ground_truth_data_validation = ground_truth_cloud_mask_arr * alpha_band


  values, counts = np.unique(data_validation, return_counts = True)
```

```
  cloud_mask_picture = rescale_img(data_array[band]*(data_validation))
  no_cloud_mask_picture = rescale_img(data_array[band]*(alpha_band),pmin=0.2,
pmax=99.8)
  ground_truth_cloud_mask_picture =
rescale_img(data_array[band]*(ground_truth_data_validation))

  return cloud_mask_picture, no_cloud_mask_picture,
ground_truth_cloud_mask_picture

#extracting bands from toy dataset

blue_band = np.reshape(toy_dataset[:,2], (1,-1))
red_band = np.reshape(toy_dataset[:,0], (1,-1))
nir_band = np.reshape(toy_dataset[:,3], (1,-1))
ndvi_band = np.reshape(toy_dataset[:,7], (1,-1))
swir1_band = np.reshape(toy_dataset[:,4], (1,-1))
green_band = np.reshape(toy_dataset[:,1], (1,-1))
ndsi_band = np.reshape(toy_dataset[:,8], (1,-1))
sorted_blue_band = np.sort(blue_band)
blue_band_len = np.shape(blue_band)[1]

blue_band = np.reshape(blue_band,(1,-1))

#specifying threshold in which percentiles stop being 0

zero_threshold = (len(blue_band[blue_band == 0]) / len(blue_band[0])) *100

zero_threshold

"""<h3><strong>Helper Functions <string></h3>"""

def get_cloud_from_band(band,threshold,blue_band):
  '''
  extracts cloud pixels from band

  INPUTS:

  band : (array) array of values of the band
  threshold : (int) threshold on the blue band
  blue_band: (array) array of values of the blue band

  OUTPUTS:

  cloud : (array) pixels and their values belonging to the cloud surface
```

```
  '''
  cloud = band[0][np.where(blue_band[0] > threshold)]
  cloud = np.reshape(cloud, (-1,1))
  return cloud

def get_rvs_from_band(band,ndvi_band):
  '''
  extracts rvs pixels from band

  INPUTS:

  band : (array) array of values of the band
  ndvi_band: (array) array of values of the ndvi band

  OUTPUTS:

  rvs : (array) pixels and their values belonging to the rvs surface

  '''
  rvs = band[0][np.where(ndvi_band[0] >= 0)]
  rvs = np.reshape(rvs, (-1,1))
  return rvs

def create_z_cloud(bands,threshold,blue_band):
  length = len(np.where(blue_band[0] > threshold)[0])
  z_cloud = np.zeros((length,len(bands),1))
  for idx,band in enumerate(bands):
    cloud = get_cloud_from_band(band,threshold,blue_band)

    z_cloud[:,idx] = cloud
  z_cloud = np.nan_to_num(z_cloud, nan = 0)

  cloud_mean = np.nanmean(z_cloud,axis = 0)

  return z_cloud, cloud_mean

def create_z_rvs(bands,ndvi_band):
  length = len(np.where(ndvi_band[0] >= 0)[0])
  z_rvs = np.zeros((length,len(bands),1))
  for idx,band in enumerate(bands):
    rvs = get_rvs_from_band(band,ndvi_band)
    z_rvs[:,idx] = rvs
  z_rvs = np.nan_to_num(z_rvs, nan = 0)
```

```
  rvs_mean = np.nanmean(z_rvs, axis = 0)

  return z_rvs, rvs_mean

"""<h3><strong>Solver</strong></h3>"""

#solver

#define thresholds

vals = [95,98,99]
thresholds = [np.percentile(blue_band,val) for val in vals]
thresholds.append(1250)

#define parameters

rbf_kernel = lambda x,y : np.matmul(x.T,y)

show_band = 0

ndvi_band = np.nan_to_num(ndvi_band, nan = -1) #dont want things to mistakenly be
rvs
x_arr = []
dist_arr = []
bands = [blue_band, red_band, nir_band, green_band, swir1_band]
ground_truth_cloud_mask_arr = create_cloud_mask(reference_image_data[2], 1250)
#blue band is first arg

arr = []
sign_counts = []

ground_truth_cloud_mask_arr = create_cloud_mask(reference_image_data[2], 1250)

#start process

for j in range(len(vals)+1):

  #instantiation
  print(j)
  threshold = thresholds[j]

  print(f'threshold : {threshold}')

  #create cloud mask
```

```
  cloud_mask_arr = create_cloud_mask(reference_image_data[2], threshold) #blue
band is first arg

  x= []
  dist = []

  #cloud

  z_cloud, cloud_mean = create_z_cloud(bands, threshold, blue_band)

  #rock and vegetation

  #https://www.researchgate.net/publication/316471841_Quantification_and_understa
nding_the_observed_changes_in_land_cover_patterns_in_Bangalore/figures?lo=1

  z_rvs, rvs_mean = create_z_rvs(bands, ndvi_band)

  #formula -
https://www.sciencedirect.com/science/article/abs/pii/S0034425717305138


  rbf_numerator = lambda x : rbf_kernel(x, cloud_mean) - rbf_kernel(x, rvs_mean)
- rbf_kernel(cloud_mean, rvs_mean) + rbf_kernel(rvs_mean, rvs_mean)
  rbf_denominator_1 = lambda x : np.sqrt(rbf_kernel(x,x) -
(2*rbf_kernel(x,rvs_mean)) + rbf_kernel(rvs_mean, rvs_mean))
  rbf_denominator_2 = lambda x : np.sqrt(rbf_kernel(cloud_mean,cloud_mean) -
(2*rbf_kernel(cloud_mean,rvs_mean)) + rbf_kernel(rvs_mean, rvs_mean))
  cos_theta = lambda x : rbf_numerator(x) / (rbf_denominator_1(x) *
rbf_denominator_2(x))
  for val in z_cloud:
    x.append(cos_theta(val))
    dist.append(rbf_kernel(val, rvs_mean))

  #evaluation

  values, counts = np.unique(np.sign(x), return_counts = True)
  try:
    sign_counts.append(abs(counts[1] - counts[0]))
  except:
    sign_counts.append(abs(counts[0]))

  cloud_mask_picture, no_cloud_mask_picture, ground_truth_cloud_mask_picture =
show_image(show_band, cloud_mask_arr, reference_image_data, threshold,
ground_truth_cloud_mask_arr)
```

```
  arr.append([cloud_mask_picture, no_cloud_mask_picture,
ground_truth_cloud_mask_picture])

  x_arr.append(x)
  dist_arr.append(dist)

"""<h3><strong>Figures</strong></h3>"""

fig, axs = plt.subplots(len(x_arr),4)
fig.tight_layout()

fig.set_figwidth(15)
fig.set_figheight(15)

for i in range(len(x_arr)-1):



  cloud_mask_picture = arr[i][0]
  no_cloud_mask_picture = arr[i][1]
  ground_truth_cloud_mask_picture = arr[i][2]

  #plots image with cloud mask

  axs[i,0].imshow(cloud_mask_picture, cmap = 'cividis')
  axs[i,0].set_title(f'with cloud mask : threshold = {thresholds[i]}')

  #plots image without cloud mask

  axs[i,1].imshow(no_cloud_mask_picture,cmap = 'cividis')
  axs[i,1].set_title('without cloud mask')

  #plots ground truth image

  axs[i,2].imshow(ground_truth_cloud_mask_picture,cmap = 'cividis')
  axs[i,2].set_title('ground truth cloud mask (looking at hist)')

  #plots histogram of sign dist

  _, _, patches = axs[i,3].hist(np.array(list(map(lambda x : np.sign(x),
x_arr[i])))).flatten(), label = f'metric : {sign_counts[i]}')
  for pp in patches:
    x = (pp._x0 + pp._x1)/2
    y = pp._y1 + 0.05
    if(pp._y1>0):
```

```
      axs[i,3].text(x, y, pp._y1)

  axs[i,3].set_title(f'sign distribution : threshold = {thresholds[i]}')
  axs[i,3].set_xlabel('sign')
  axs[i,3].set_ylabel('count')
  axs[i,3].legend()



axs[len(x_arr)-1,0].text(0.3,0.5,'ground truth')

_, _, patches = axs[len(x_arr)-1,3].hist(np.array(list(map(lambda x : np.sign(x),
x_arr[len(x_arr)-1])))).flatten(), label = f'metric : {sign_counts[len(x_arr)-
1]}')
for pp in patches:
  x = (pp._x0 + pp._x1)/2
  y = pp._y1 + 0.05
  if(pp._y1>0):
    axs[len(x_arr)-1,3].text(x, y, pp._y1)

axs[len(x_arr)-1,3].hist(np.array(list(map(lambda x : np.sign(x),
x_arr[len(x_arr)-1])))).flatten())
axs[len(x_arr)-1,3].set_title(f'sign distribution : threshold =
{thresholds[len(x_arr)-1]}')
axs[len(x_arr)-1,3].set_xlabel('sign')
axs[len(x_arr)-1,3].set_ylabel('count')
axs[len(x_arr)-1,3].legend()

fig, axs = plt.subplots(len(x_arr),1)
fig.tight_layout()

fig.set_figwidth(15)
fig.set_figheight(15)

for i in range(len(x_arr)):
  axs[i].hist(np.array(list(map(lambda x : np.sign(x), x_arr[i])))).flatten())
  axs[i].set_title(f'sign distribution : threshold = {thresholds[i]}')
  axs[i].set_xlabel('sign')
  axs[i].set_ylabel('count')

from scipy.stats import iqr

fig, axs = plt.subplots(len(x_arr),3)
fig.tight_layout()
```

```
fig.set_figwidth(15)
fig.set_figheight(15)

cv = lambda x : np.std(x)/np.mean(x)

for i in range(len(x_arr)):

  cloud_mask_picture = arr[i][0]

  #image with cloud mask

  axs[i,0].imshow(cloud_mask_picture, cmap = 'cividis')
  axs[i,0].set_title(f'with cloud mask : threshold = {thresholds[i]}')

  #plots histogram of sign dist

  _, _, patches = axs[i,1].hist(np.array(list(map(lambda x : np.sign(x),
x_arr[i]))).flatten(), label = f'metric : {sign_counts[i]}')
  for pp in patches:
    x = (pp._x0 + pp._x1)/2
    y = pp._y1 + 0.05
    if(pp._y1>0):
      axs[i,1].text(x, y, pp._y1)

  axs[i,1].set_title(f'sign distribution : threshold = {thresholds[i]}')
  axs[i,1].set_xlabel('sign')
  axs[i,1].set_ylabel('count')
  axs[i,1].legend()

  x_iqr = iqr(x_arr[i])

  #plots scatterplot of distribution

  axs[i,2].scatter(x_arr[i], dist_arr[i])
  pct_25 = np.percentile(x_arr[i],25)
  pct_75 = np.percentile(x_arr[i],75)
  axs[i,2].axvline(x = pct_25, color = 'g', label = '25th percentile')
  axs[i,2].axvline(x = pct_75, color = 'g', label = '75th percentile')
  axs[i,2].axvline(x = pct_75 + (3*x_iqr), color = 'r', label = f'outliers, iqr =
{np.round(x_iqr,2)}')
  axs[i,2].axvline(x = pct_25 - (3*x_iqr), color = 'r', label = f'outliers, iqr =
{np.round(x_iqr,2)}')

  axs[i,2].set_xlabel('theta_rel')
  axs[i,2].set_ylabel('distance from rock and vegetation')
```

```
  axs[i,2].set_title(f'threshold = {thresholds[i]}, cv = {cv(x_arr[i])}')
  axs[i,2].legend()

fig.suptitle('distance from rock and vegetation vs. theta rel for all
thresholds')

#print total time

total_time = time.time() - t0
print(total_time)
```