**Jeremias Penttilä**

# A method for anomaly detection in hyperspectral images, using deep convolutional autoencoders

Master's Thesis in Information Technology

November 14, 2017

University of Jyväskylä

Faculty of Information Technology

**Author:** Jeremias Penttilä

**Contact information:** jeremias.f.penttila@student.jyu.fi

**Supervisor:** Ilkka Pölönen, Timo Hämäläinen

**Title:** A method for anomaly detection in hyperspectral images, using deep convolutional autoencoders

**Työn nimi:** Menetelmä poikkeavuuksien havaitsemiseen hyperspektrikuvista käyttäen syviä konvolutiivisia autoenkoodereita

**Project:** Master's Thesis

**Study line:** Computational Science

**Page count:** 70+5

**Abstract:** Detecting anomalies from any image data, especially hyperspectral ones, is not a trivial task. When combined with the lack of apriori labels or detection targets, it grows even more complex. Detecting spectral anomalies can be done with numerous methods, but the detection of spatial ones is vastly more complicated affair. In this thesis a new way to detect both spatial and spectral anomalies at the same time is proposed. The method has been designed with hyperspectral data in mind, but should work for conventional images also. This is achieved works by using 3-d convolutional autoencoders to learn commonly occurring features both spatial and spectral, across the the test data. By running the test data through this network, the data is transformed to a feature-space. In this space, the images can be analyzed for the presence of anomalies by the means of standard anomaly detection algorithms. A simple real-world use case with unmodified images is presented. Second run for validation purposes is done with data containing synthetic anomalies.

**Keywords:** machine learning, anomaly detection, hyperspectral, hdbscan, convolutional neural network, autoencoder, convolutional autoencoder, CAE, SCAE, deep learning

**Suomenkielinen tiivistelmä:** Poikkeavuuksien havaitseminen kuvista, erityisesti hyperspektraalisista kuvista, on hankalaa. Kun ongelmaan yhdistetään ennalta tuntematon data ja

poikkeavuudet, muodostuu ongelma vielä laajemmaksi. Spektraalisten poikkeavuuksien havaitsemiseen on kehitetty useita eri menetelmiä, mutta spatiaalisten poikkeavuuksien havaitseminen on huomattavasti hankalempaa. Tässä työssä esitellään uudenkaltainen menetelmä sekä spatiaalisten että spektraalisten poikkeavuuksien samanaikaiseen havaitsemiseen. Menetelmä on suunniteltu erityisesti spektraaliselle datalle, mutta soveltuu myös perinteisille kuville. Menetelmässä kolmiulotteisilla konvolutionaalisilla autoenkoodereilla löydetään koulutusdatassa esiintyviä normaaleja piirteitä. Tätä verkkoa käyttämällä voidaan testidata projisoida piirre-avaruuteen. Tästä projisoidusta datasta voidaan etsiä poikkeavuuksia käyttäen perinteisiä algoritmeja. Työssä esitetään kahdet erilliset tulokset. Ensimmäisissä on esitetty menetelmän toimivuus todellisuutta vastaavassa tilanteessa, jossa tietoa poikkeavuuksista ei ole etukäteen. Näiden tulosten lisäksi toinen ajo datalla, johon on lisätty synteettisiä tunnettuja poikkeavuuksia suoritetaan. Tämän toisen ajon tulokset voidaan validoida, koska anomaliat ovat nyt tunnettuja.

**Avainsanat:** koneoppiminen, poikkeavuus, neuroverkko, hyperspektri, hdbscan, konvolutio, autoenkooderi

# Glossary

**AE** Autoencoder.

**AUC** Area Under ROC-Curve.

**BP** Back-Propagation.

**CAE** Convolutional autoencoder.

**CNN** Convolutional Neural Network.

**DBSCAN** Density-Based Spatial Clustering of Applications with Noise.

**EM** Electromagnetic.

**ENVI** Raster file format used to store hyperspectral images. Uses data file and accompanying header file.

**FNR** False Negative Rate. $FNR = \frac{false\ negative}{true\ positive + false\ negative}$.

**FPR** False Positive Rate. $FPR = \frac{false\ positive}{false\ positive + true\ negative}$.

**GLOSH** Global-Local Outlier Score from Hierarchies.

**HDBSCAN** Hierarchial DBSCAN.

**HSI** Hyperspectral imaging.

**MLP** Multilayer Perceptron Network.

**MSE** Mean Squared Error.

**MST** Minimum Spanning Tree.

**RMSE** Regulized Mean Squared Error.

**ROC** Receiver Operating Characteristics.

**RX** Reed-Xiaoli.

**SCAE** Stacked Convolutional Autoencoder.

**SNAP** Sentinels Application Platform, an ESA provided program for handling Sentinel mission data.

**TPR** True Positive Rate, also: recall. $TPR = \frac{true\ positive}{true\ positive + false\ negative}$.

# List of Figures



# List of Tables

# Contents

# 1 Introduction

> One of these things is not like the others, one of these things does not belong.
>
> *Sesame Street*

As Sesame Street teaches us, one can give even children an image and ask them to find something anomalous from it and they would prevail. A deceptively simple process from a human point of view, but an infinitely complex for a computer. Ever since the field of computer vision was created, people have been creating systems and algorithms, methods and techniques to combat this problem. Some of these works to a point under certain conditions, but most are still quite limited. Also one needs to ask what are anomalies in the domain of images? They might be anomalous shapes, or colors. But anomalous to what? To the picture in question or in general? As one can see the detection of anomalies in a formal way can get quite an involved effort.

Trying to detect minute differences between two materials solely on three values (standard image: red green and blue) is extremely difficult, and in some cases completely impossible. The use of Hyperspectral imaging (HSI) for image acquisition gives one access to lot more data, or more accurately a more detailed data about the materials in the image. The more detailed data one has, the easier it is to detect anomalies from the image. Though one has to bear in mind the curse of dimensionality. This ability to get fairly accurate spectroscopic readings from a distance has made HSI something of a trend in recent years. Whether this is the cause of increased computing power or better imaging technology one cannot say. Most likely both of these have carried their weigh into the emergence of these technologies. Still image analysis in general was, for a long time, in a rut. This was changed by the increase in computing power and the insight of Geoffrey Hinton in 2006 (Hinton, Osindero, and Teh 2006), giving rise to a new technique: deep learning. This opened a lot of previously closed roads for image analysis. As an idea deep learning had been conceived around the turn of the millennial, but it was thought that training such a network was too difficult. Hinton, Osindero, and Teh 2006 gave an alternative to traditional training methods, and as a side effect created a new powerful tool for image analysis, and other data scientist alike. At the same time, advances in manufacturing technology has given us more accurate and

smaller hyperspectral sensors, and the rise of UAVs[1] gave more widespread access to aerial hyperspectral imaging, a technique previously available to chosen few.

HSI gives access to detailed data, and can be used to identify different materials from the images. The accuracy of this identification is of course dependent on a variety of parameters ranging from the used sensor to the properties of the materials, but as a general rule HSI can identify or at least differentiate them. This gives the ability to detect materials from the image, but anomaly detection still requires more information. To detect anomalies one has to know what is normal. Historically this has been achieved for example by using statistics. The goal of this thesis is to present a new way to detect anomalies in hyperspectral images in an unsupervised manner.

## 1.1 Background

Computer vision is one of the most prominent areas of research in computer science at the moment, and recent years have seen a rise in techniques using HSI data. On top of the standard use-cases of machine vision (segmentation, classification etc.), HSI technology gives access to far wider spectra and therefore enhances the abilities of computer vision. This has caused HSI to be suggested as a magic bullet to anything from quality control to crop maintenance. The re-emergence of neural networks/deep learning also gave some new wind in this field, and catapulted it to an era of neural network based techniques.

Anomaly detection itself is closely related to classification; one cannot detect anomalies if no normal model exist. To create this model one needs to classify data to normal classes. During my studies I was heavily interested in anomaly detection, and later I started to work with hyperspectral images, specifically detection certain spectral signatures from the image. It was natural for me to combine these two, and first begun by studying the methods of anomaly detection for images in general, and later for hyperspectral images specifically. For normal visible spectrum RGB-images the techniques are mainly shape based, since the spectrum is rather limited and the detection of spectral anomalies is limited to characterize different media. For hyperspectral images the techniques can detect both spectral and spatial anoma-

1. Unmanned Aerial Vehicle

lies. The different techniques themselves are not restricted to any image type. In principle standard and hyperspectral images don't differ. The latter has more spectral information, but the structurally they are same, and therefore same anomaly detection techniques can work on both settings.

Recent technological advances have been a boon for deep learning. Still after my introductory studies in image-based anomaly detection, specifically in hyperspectral images, I saw that these powerful techniques are underrepresented in hyperspectral anomaly detection. The golden standard for hyperspectral anomaly detection has been the Reed-Xiaoli (RX) algorithm. While functional, it's quite limited and can detect only spectral anomalies. Some other techniques have also been introduced in recent years, but most of these still work by statistical and/or probabilistic ways (most are derivatives of the original RX algorithm). Thus, I decided to focus my studies to leverage the recent advances in deep learning.

### 1.1.1 Problem setting

The idea for this thesis started with the following problem: imagine a vast dataset of hyperspectral images covering a large geographical area. The natural choice would be satellite images. Now the question is: what does not belong? This question is way too general to be answerable by the existing methods of hyperspectral anomaly detection. A single technique, like RX, can be used to detect spectral anomalies, but those cover only part of the answers to the original question. And the detection of spatial anomalies is in itself quite complicated. The definition of anomalies also needs to be specified. If the dataset is of a forested area with a single town, this town in its entirety can be an anomaly, while locally (i.e a single image in the dataset containing the said town) it's not. Detecting global anomalies from local ones is quite different. While the same techniques can be used, the training phase is different for these two. As a start point I decided to focus mainly on global anomalies, but with some modification also local ones can be detected. The original idea was to mainly focus on spectral anomalies, but the technique proposed can also find spatial ones.

## 1.2 Research problem

The problem distills now to a single question: how can one detect any anomalies, spectral or spatial, from any kind of hyperspectral dataset without any prior knowledge of the said dataset? Not an easy question by any means, especially when I wanted to create a method for finding both types of anomalies at the same time. To answer this main question, based on what I learned during the preliminary research process, I decided to go with neural networks.

Since the raise of deep learning, some interesting advantages have been made on the field of image analysis by leveraging convolution instead of raw image processing. This will be explored further in theory chapter, but simply but convolution gives a more natural, *fuzzy* way of analysis images. It also more closely resembles the way human vision works. So choosing to use convolution was a natural choice when trying to detect anomalies from any kind of images. From the different convolutional neural networks to choose from, I decided on *convolutional autoencoders*, or their deep learning variants *stacked convolutional autoencoders* for the reasons explained in chapter 2. With this choice in mind, the research problem can be now refined to a more suitable one: Can *convolutional autoencoders* or *stacked convolutional autoencoders* be used to detect anomalies from hyperspectral data?

This thesis does not aim to provide a conclusive answers for these questions, but to conduct an exploratory study of the feasibility of one possible method.

## 1.3 Structure

The thesis is structured as follows: this chapter provided some background on the topic, and the origins of the idea to use Convolutional autoencoders (CAEs) for detection. Chapter 2 will go through some basics of hyperspectral imagery and anomaly detection. This chapter will also present some neural-network models in more detail. After the introduction to the required theory, chapter 3 will present first preset the data used in this thesis, and secondly demonstrate an implementation of the proposed method. Results obtained from the proposed implementation are presented in chapter 4. Chapter 5 will present some discussion on top of these results, and the thesis in general. Finally in chapter 6 conclusions are drawn based on all the presented work.

# 2 Theory

This chapter will present the theory behind the proposed method, and will start with a short introduction to HSI in section 2.1. Note that HSI will not be explored in detail. No imaging techniques etc. will be introduced as these are not relevant to the research questions mentioned. A short primer will be also given to spectroscopy, since it is the field of study that enables this kind of detection methods. Next will be a very general section on anomaly detection, and in a little more detail, section 2.2.1 will formally present the used *HDBSCAN* and *GLOSH* methods. In section 2.3 the theory starts to delve into neural networks. The previous sections are fairly general in form, but from this point onward a more formal mathematical approach was taken. The basic building blocks for the proposed method are introduced. In section 2.3.1 the basics of *autoencoders* and in section 2.3.2 the *convolutional neural networks* are introduced. Lastly, in section 2.3.3 these two are combined to form the *convolutional autoencoder*.

At the end of this chapter the reader should have an idea about the possible problems present in anomaly detection from hyperspectral images, and should be familiar with the basic building blocks of the method that will be proposed in chapter 3.

## 2.1 Hyperspectral imaging

In the grand scale of things, hyperspectral imaging (also known as imaging spectroscopy) is a relatively new technique. While the science behind it has been known from 19th century, the technology to actually build an *imaging spectrometer* was not really developed until the 1970s-80s ( Goetz 2009). Now when we think of a standard image, specifically a color photograph, it consists of red, green and blue dots. There exists a lot of other color spaces, but the RGB space is somewhat intuitive to use, since it loosely corresponds to how human brain interprets colors (fig. 2). Now unlike in RGB-images, the Electromagnetic (EM) spectrum in reality is continuous (fig. 3). So, to store an image of a scene, all this continuous data needs to be sampled or otherwise one image would be infinitely large. Different regular cameras may differ in how they sample the EM spectrum, but in general they work by filtering the image

to three different detectors (*Bayer Filter*) and using math to generate the final image (Adams, Parulski, and Spaulding 1998). Now from figures 2 and 3, one can easily see that with only three values saved for each pixel, a lot of data about the spectrum is omitted. Depending on the construction of the sensor and on the math used, some of this data may be included in the measurements but it cannot be extracted. Say if the sensors blue detector is sensitive to 400-500nm range, then all the data from this spectrum is included in the measurement, but because the *spectral resolution* is low (100 to be exact in this case), detailed data from this area is not recoverable.

Hyperspectral cameras (aka. imaging spectroscopes) are logically like any other camera, but with a much higher spectral resolution and continuous range. There are no exact definition on how high the spectral resolution, or how large the spectral range should be for a camera to be considered hyperspectral, but one crucial aspect is the continuity of measurements on the spectrum (Goetz 2009). If the captured spectrum is not continuous, then the camera is not hyperspectral but multispectral (Goetz 2009). This makes standard cameras multispectral ones, though quite limited ones that save information across only three *bands*: red, green and blue generating a collection of three 2-dimensional images. By contrast images created by hyperspectral cameras contain a lot more of these images, and are often combined to as single unit, referred as a hyperspectral cube (figure 1). As to the spectral range of hyperspectral cameras, these cover a wide range of possibilities from UV to IR, and there are no industry standards on what the range should be. Technically to be called hyperspectral, the camera would need to save more than one band. While the hyperspectral cameras do work on the same principle as any other camera, they are subject to quite complicated details, like atmospheric calibration (Stein et al. 2002). This makes the use of such a camera fairly complicated technical operation.

There are two main reasons to use HSI. Firstly the spectral range of the sensor often exceeds that of a standard camera, and therefore see things otherwise invisible. The second reason is the continuity of spectral information, and specifically the identification of materials by means of spectral analysis. Spectral analysis is a technique that was born in 1835, when Sir Charles Wheatstone proposed, that different metals could be identified by studying the light they reflect. The basis of this technique is the fact that all material absorbs some EM

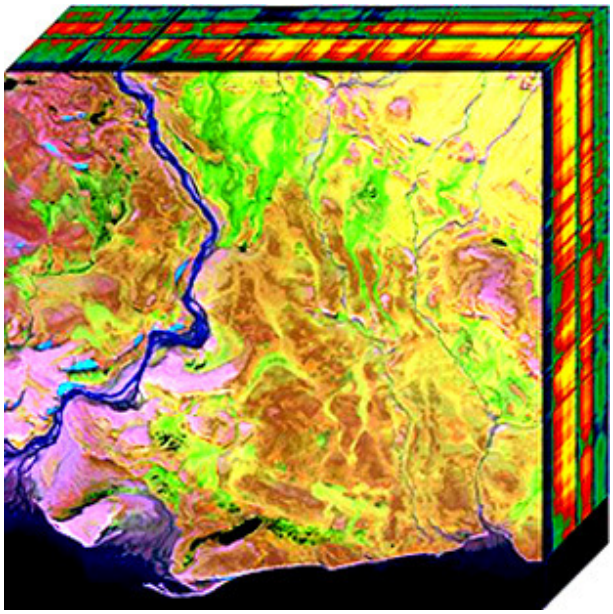


Figure 1: Visualization of hyperspectral image, i.e. a hyperspectral cube (Wikimedia Commons 2007c).

radiation, and this generates a distinct *absorption spectrum* for each material. HSI was chosen for this thesis because it gives access to a lot more data, and therefore the detection of anomalies becomes, not simpler but easier as there might be important features outside the visible spectrum and/or hidden amid the visible range. The anomaly detection method itself should work on any imaging data, not only hyperspectral.

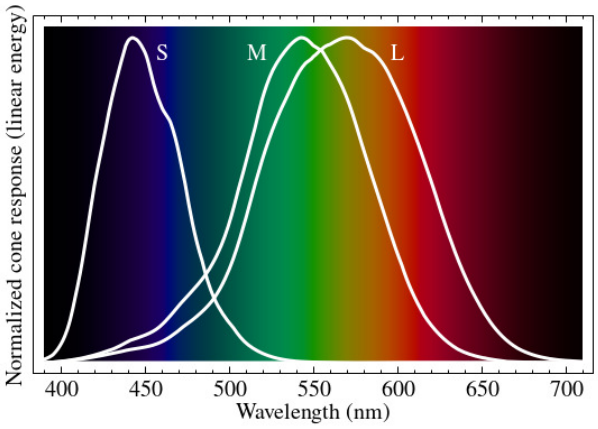



Figure 2: Normalized response of human cones to different spectras of light (Wikimedia Commons 2009)

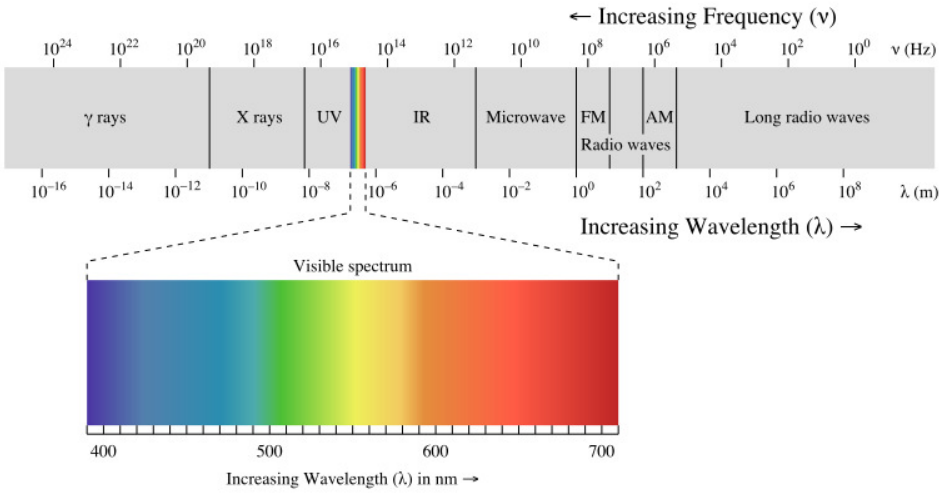



Figure 3: Electromagnetic spectrum (Wikimedia Commons 2007b)

## 2.2 Anomaly detection

Anomaly detection is the act of detection anomalous data points from dataset of which distribution is somehow known. It is also closely related to statistics, and more recently machine learning. Statistical methods to detect anomalies could be, for example the use of probability distributions and a machine learning one the use of clustering. Anomaly detection is used in a wide array of fields, from fraud detection to medical imaging. Anomaly detection is also known as outlier or novelty detection, though novelty detection is associated with previously unseen event while anomaly detection is necessarily not (Chandola, Banerjee, and Kumar 2009).

To detect anomalies the first question to ask is: what are anomalies. Chandola, Banerjee, and Kumar 2009 divide anomalies to three main groups: *point*, *contextual* and *collective*. Point anomalies being the simplest ones: a single data point that is anomalous in either local or global neighborhood. For example a single large credit card purchase overseas is a point anomaly in two regards: the abnormally large sum and location, putting these together it is a large point anomaly. Contextual anomalies are anomalous in a specific context, not otherwise. To continue the credit card example: say every morning you have a cup of tea at cafe A, and every evening at cafe B. Now one day the purchase are switched around. In large scheme of things, neither is anomalous because both cafe's A and B are visited frequently, but

in the context of time series they are anomalous. Lastly collective anomalies are anomalous as a group, but not alone. Again a credit card analogy: a sudden increase in payments in a short time might indicate stolen credit card, and someone trying to max it out. Any single purchase itself might not be anomalous, but as a group they are. The border between different anomalies can be a bit fuzzy in real world applications, but generally anomalies do fall into one of these categories. Though the interpretation on what is point, contextual or collective anomaly can be very dependent upon the problem setting.

For the purpose of this thesis, two basic anomalies are viewed: spectral and spatial. These can, of course, both be present at the same time. The anomalies searched in the current configuration of the proposed method are global collective anomalies. With modification to the network it is conceivable, that local ones could also be found. While both are collective anomalies, they are however a bit different. The way the method works actually finds anomalous areas, making them collective anomalies. Normally spectral anomalies would be point anomalies, but because the method finds anomalous areas, they are viewed as collective. Since spatial anomalies are by definition dependent on a collection of data points arranged in some abnormal way, they are always collective anomalies (Chandola, Banerjee, and Kumar 2009). The size of these areas is dependent on the parameters of the method, and is explained in detail in chapter 3. The two anomaly types of interest are fairly straightforward: spectral anomalies are abnormal spectras for a single area, and spatial ones are abnormal shapes in an area. In this method the two can be intertwined, and for spectral anomaly the spectras don't have to be same across all the area, i.e. in RGB-image this kind of anomaly could for example be a wave-like change in colors. The change in color making it spectral and the wave-like structure a spatial one.

Anomaly detection in hyperspectral images, or in any images, is not a new idea. Since spatial anomalies are not dependent on hyperspectral data, when studying them there is no need to use hyperspectral data. In this thesis both are looked for, because the proposed method can find both of them. Generally when speaking about hyperspectral anomaly detection, the focus has been spectral anomalies. When searching for spectral anomalies, they are usually done using statistical methods, specifically anomaly is considered anything abnormal from the background, either local or global(Stein et al. 2002). On of the more common hyper-

spectral anomaly detection methods was proposed by Reed and Xiaoli 1990, and has since achieved the status of a benchmark in hyperspectral anomaly detection (Banerjee, Burlina, and Diehl 2006). The RX detector is relatively simple one: it operates per pixel basis, comparing the pixel under scrutiny to a local background. This background is assumed to follow Gaussian distribution. The target pixel is then compared to background mean vector and can be classified as normal or anomalous. RX detector can bee seen as a statistical anomaly detection method. While widely used it still has a few weaknesses: firstly the assumption of background distribution is rarely true, and the algorithm computationally costly. Still it has kept its status as a benchmark, and multiple different variations of the algorithm have been proposed (e.g Chang and Chiang 2002; Stein et al. 2002).

### 2.2.1 HDBSCAN/GLOS

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a fairly widely used unsupervised clustering method. Originally proposed by Ester et al. 1996, it filled a gap in clustering methods. DBSCAN clustering has two main advantages: firstly unlike traditional clustering algorithms (i.e. k-means) it does not require *apriori* knowledge of clusters in the data. For unlabeled data this provides a large advantage: instead of finding the number of clusters by trial-and-error, the algorithm does this automatically. Second main advantage is the ability to find clusters of different shapes. For example: k-means clustering finds clusters of roughly circular shape and fails with non-linear datasets. DBSCAN is capable to detect non-linear clusters, or clusters of any arbitrary shape by basing them on density. Shortly the DBSCAN algorithm works by going through all the data points, and if they are closer than parameter $\varepsilon$ to each other, they are considered to be in a same cluster. In terms of graph-theory, each node $n$ is considered to be in a cluster $C$, if they can be connected with a walk $w$ to any other point in the cluster, and where the distance between any two adjacent points in the walk is not greater $\varepsilon$. Formally: $\forall n_i, n_j \in C \exists w = \{n_i = v_1, v_2, ..., v_n = n_j\} : dist(v_k, v_{k+1}) < \varepsilon$, and where $dist$ is some distance function, commonly Euclidean distance. On top of $\varepsilon$, the second parameter required by DBSCAN is the $p_{min}$ parameter. Parameter $p_{min}$ is the minimum amount of points for a point to be considered a *core* point. It also determines the minimum amount of points need

to form a cluster (at least one core point is required to form a cluster). If points are not in any cluster, they are classified as *noise*. This idea is depicted in figure 4, blue nodes being noise, red core (at least $p_{min}$) and yellow are non-core points belonging to the cluster (sometimes *border* points).

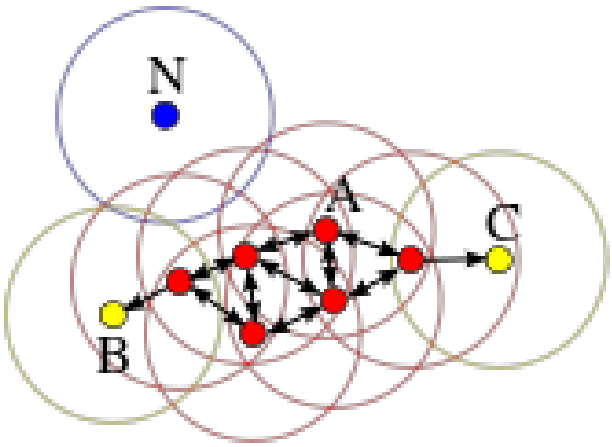



Figure 4: Illustration of DBSCAN (Wikimedia Commons 2007a)

While DBSCAN is straightforward and well performing algorithm, it does require two parameters both of which depend on the distribution of data. There have been some variations to the base algorithm to overcome this issue, by estimating these parameters (e.g. Smiti and Elouedi 2012), and one of these variations is the Hierarchial DBSCAN (HDBSCAN) algorithm proposed by Campello et al. 2015. HDBSCAN belongs to a group of *hierarchical* clustering algorithms. These can be divided into two main groups: *agglomerative* and *divisive*. Both of these work by building a hierarchy of clusters. Agglomerative is a ground-up method, where each point is its own clusters, and these are then combined to larger and larger clusters. Divisive is the opposite: each point is in one cluster and this cluster is the being divided and divided. While HDBSCAN is an improvement upon the standard DBSCAN method, it still does require one parameter: $p_{min}$. However $\varepsilon$ is not required anymore.

DBSCAN algorithm starts the clustering by finding core points with respect to $\varepsilon$. HDBSCAN algorithm starts in a similar way, but instead of finding core points it ask: for how large $\varepsilon$ point is a core point. This value for point $x_p$ is called *core distance*: $d_{core}(x_p)$ and is the distance to the $p_{min}$:th neighbor of point $x_p$. With this value an another important definition can be made: the *mutual reachability distance* between two points $x_p$ and $x_q$ which is defined

as:

$$d_{mreach}(x_p, x_q) = max(d_{core}(x_p), d_{core}(x_q), d(x_p, x_q)).$$

This value is the minimum value for $\varepsilon$ so that $x_p$ and $x_q$ are $\varepsilon$-reachable, i.e so that $x_p \in N_\varepsilon(x_q)$ and $x_q \in N_\varepsilon(x_p)$, $N_\varepsilon(x_p)$ being the $\varepsilon$-neighborhood of point $x_p$ (all points in $\varepsilon$ radius from $x_p$). With these two definitions the third and last one can be constructed: the *mutual reachability graph*: $G_{m_{pts}}$. This being a complete weighted graph of the dataset, where the weights correspond to mutual reachability distances. HDBSCAN algorithm now works by manipulating the graph. Removing all edges from $G_{m_{pts}}$ where the weight is greater than some $\varepsilon$, a new graph $G_{m_{pts},\varepsilon}$ is created. Clusters now formed in this new graph are the connected components of core points of DBSCAN with parameters $p_{min}$ and $\varepsilon$ (Campello et al. 2015). This graph is the hierarchical representation of DBSCAN, and in practice can be computed using Minimum Spanning Tree (MST). This MST (figure 5) can then be transformed to an hierarchy (figure 6).

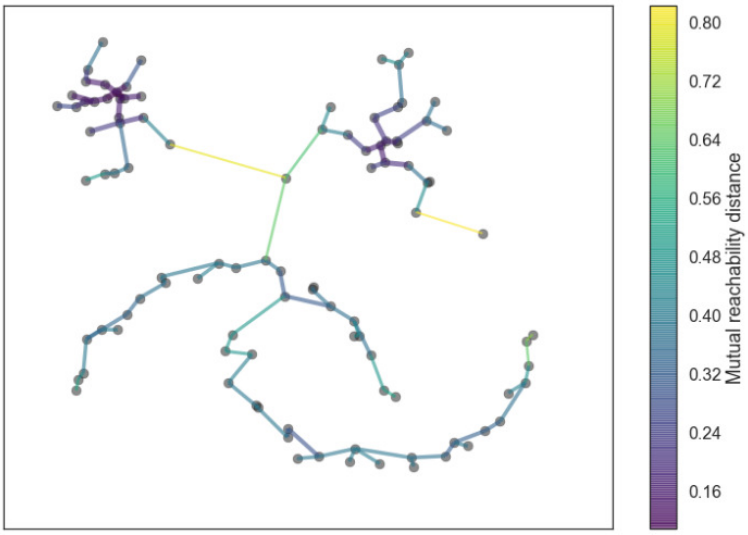



Figure 5: MST illustration of an $G_{m_{pts}}$ graph. (McInnes, Healy, and Astels 2017)

This hierarchy is then pruned to a smaller one using $p_{min}$ parameter, by considering each split: if the split creates a new cluster, that has less points than $p_{min}$, it is not considered a true split, but noise removed from the cluster. When the noise is removed we are left with a smaller tree (figure 7). From this representation final clusters are then chosen based on their stability: stable clusters that persist longer during the pruning process are preferred over unstable ones that are discarded quickly. Intuitively this means, that clusters that are

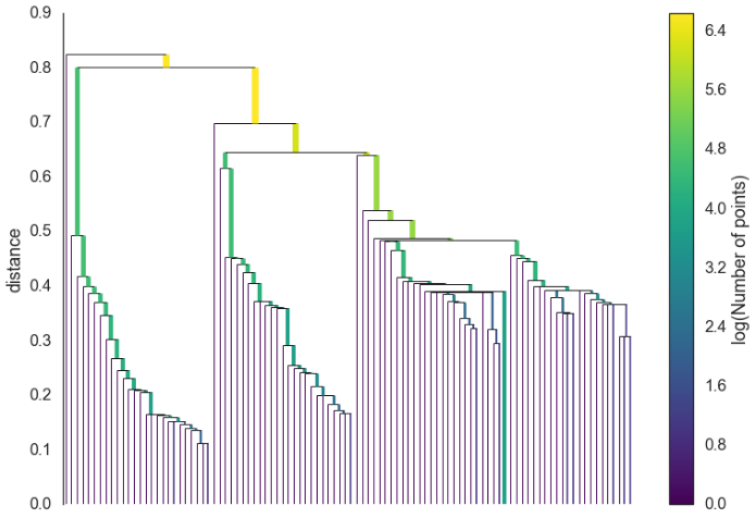



Figure 6: Hierarchical illustration of figure 5 (McInnes, Healy, and Astels 2017)

"long" in figure 7 are preferred. Note that when selecting one cluster, one cannot select it's sub-clusters anymore. To compute clusters stability, a metric is required:

$$\lambda = \frac{1}{\varepsilon}$$

With this metric, the stability ($S$) of a cluster $C_i$ is computed by

$$S(C_i) = \sum_{p \in C_i} (\lambda_{max}(p, C_i) - \lambda_{min}(C_i))$$

where $\lambda_{min}(C_i)$ is the minimum density where cluster exists, and $\lambda_{max}(p, C_i)$ is the density after which point $p$ does not belong to the cluster anymore.

To select clusters from figure 7, the hierarchy is traversed from bottom-up. Firstly all leaf nodes are selected as clusters and Stabilities are computed by equation **??**. Next the stabilities of upper-level nodes are compared to the sum of the child-node stabilities. If the stability of parent is greater than the sum of the stabilities of children, the parent node is selected, and child nodes deselected as cluster. This is continued up to root node, and selected nodes are the final clusters (shown in figure 8).

To sum up HDBSCAN firstly creates the MST for the data using equation **??** as weight metric (figure 5). From this tree, a hierarchy is created (figure 6) and further pruned (figure 7). From the pruned hierarchy clusters are then selected based on stability equation 7 (figure 8).

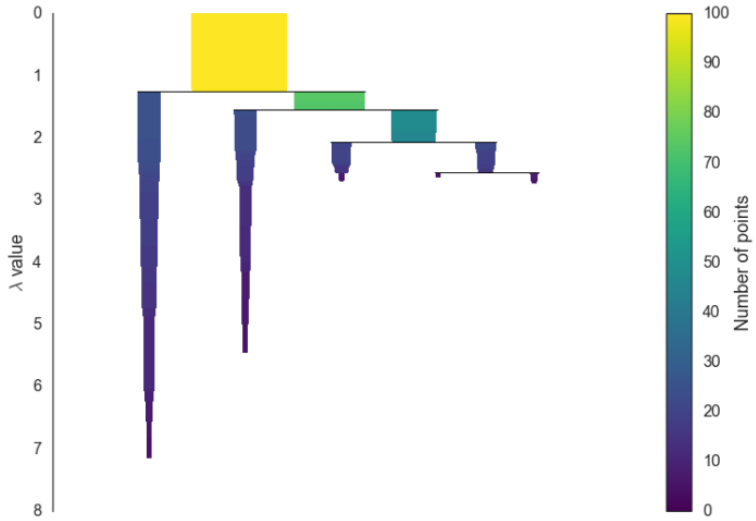



Figure 7: Condensed version of figure 6 (McInnes, Healy, and Astels 2017)

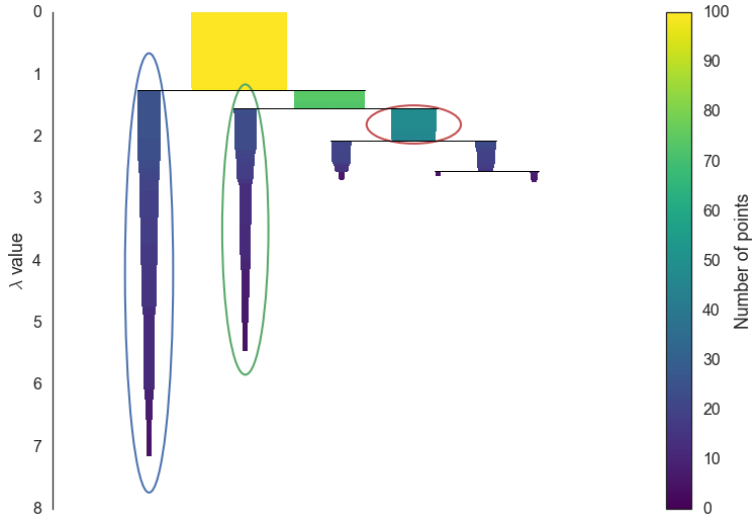



Figure 8: Selected clusters from figure 7 (McInnes, Healy, and Astels 2017)

HDBSCAN is a clustering algorithm, not an anomaly detection one. To detect anomalies another one needs to be introduced. Of course one could simply label all noise as anomalies, but to gain more control over the detector, it should assign *anomaly score* to each data point. This value is typically between 0 and 1, and tells how anomalous the point is with 0 being normal and 1 being an anomaly. By thresholding these scores, a balance point between False Negative Rate (FNR) and False Positive Rate (FPR) can be chosen. In their paper Campello et al. 2015 proposed a method called Global-Local Outlier Score from Hierarchies (GLOSH) built on top of HDBSCAN. Similar to an older anomaly detection algorithm: the

*local outlier factor (LOF)*, the GLOSH algorithm supports both local and global outliers. Local outliers being points, that in the large scale dataset are not anomalous, but compared to local neighborhood are. For any point $x_i$ this value is computed by

$$GLOSH(x_i) = \frac{\lambda_{max}(x_i) - \lambda(x_i)}{\lambda_{max}(x_i)}$$

where $\lambda(x_i)$ is the lowest density below which $x_i$ gets attached to some cluster, and $\lambda_{max}(x_i)$ the highest density above which all points of the said cluster are considered noise. Value $\lambda_{max}(x_i)$ translates to the density of the densest are of the cluster, and $\lambda(x_i)$ is the density of the point (when considered part of the cluster). Note that if $x_l$ is the densest core point (i.e $\lambda(x_l) = \lambda_{max}(x_i)$ ), then

$$\lim_{x_i \to x_l} \frac{\lambda_{max}(x_i) - \lambda(x_i)}{\lambda_{max}(x_i)} = \frac{\lambda(x_l) - \lim_{x_i \to x_l} \lambda(x_i)}{\lambda(x_l)} = \frac{\lambda(x_l) - \lambda(x_l)}{\lambda(x_l)} = 0.$$

So *GLOSH*-outlier score of a point is close to 0 when in the dense area, and for points far from any clusters $\lambda(x_i) \approx 0$, since $\varepsilon(x_i)$ (the distance for $x_i$ to be considered in a cluster) is large and $\lambda(x_i) = \frac{1}{\varepsilon(x_i)}$, and thus $GLOSH(x_i) \approx \frac{\lambda_{max}(x_i)}{\lambda_{max}(x_i} = 1$

## 2.3 Neural networks

The core principle behind the method proposed in this thesis is built on the use of a neural network, specifically deep neural network (deep nets consisting of multiple layers of standard networks). Next the building blocks for the networks used in this thesis are presented. Note that this thesis works under the assumption that the reader is familiar with the basics of neural networks. If not the book by Haykin 1998 is a good introduction to them.

### 2.3.1 Autoencoders

Autoencoders (AEs) are one of the oldest manifestations of neural networks. They have been around since the early days of neural networks. In fact the basic autoencoder is structurally identical to a Multilayer Perceptron Network (MLP) network. Like the name suggests autoencoders are neural networks whose function is to encode and decode data in a unsupervised manner, though term unsupervised is not strictly true. AEs fall to a class of neural

networks that are *self-supervised*. That is, networks that do not require user defined labels, but generate them from the used data (int the case of AEs the labels are the data itself). To the outside these networks behave similarly to unsupervised ones. AEs are able to extract information from the data, and as such are used for example dimensionality reduction or features extraction (Goodfellow, Bengio, and Courville 2016). This dimensionality reduction effect can been seen most prominently, if the activation functions are linear. In this case the AE learns to span the same subspace as PCA[1] (Goodfellow, Bengio, and Courville 2016).

AE network consists of three layers: input, hidden and output. The function between input and output layer $f : \mathbb{R}^n \to \mathbb{R}^k, f(x) = s(Wx + b) = h$ is called the *encoder* function with parameters $\theta = \{W, b\}$, where $W$ is the weight matrix, and $b$ is the bias vector. Similarly function between hidden and output layer $g : \mathbb{R}^k \to \mathbb{R}^n, g(h) = s(W'h + b')) = y$ is called the *decoder* function parameterized by $\theta' = \{W', b'\}$. Both of the functions contain non-linear mapping $s$. Commonly used functions are: *tanh*, logistic function or ReLU (Chen et al. 2014). The mappings $h \in \mathbb{R}^k$ of $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^n$ of $h$ are called the *code* and *reconstruction* respectively. In some cases the matrix $W'$ may have constraint $W' = W'^T$, in this case the autoencoder is said to have *tied weights*.

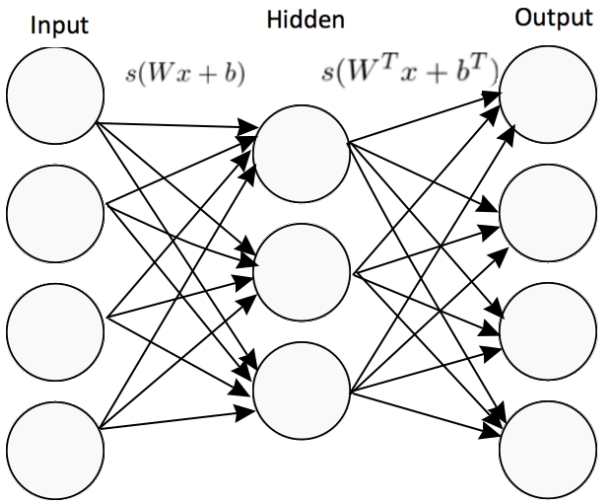



Figure 9: Simple under-complete autoencoder

Note that while AEs do contain an output layer, it is seldom used. The output-layer aims to reconstruct the original input, and as such does not contain any information. However

1. Principle Component Analysis

there are variations where the reconstruction is the wanted output, for example the *denoising autoencoder* which aims to remove noise from the input. The number of neurons in autoencoder networks is also a point of interest; in simple cases hidden layer has less neurons than the input/output layer. In this case the autoencoder is said to be *under-complete*, if the number of hidden neurons is greater than that of input/output, then the autoencoder is *over-complete* (Goodfellow, Bengio, and Courville 2016). If the dimensions of all the layers are the same, or if the network is over-complete the activation functions and/or the training phase need to be modified to prevent the network from learning trivial mappings (identity mapping), for example sparsity constraints (Goodfellow, Bengio, and Courville 2016).

So far the differences between MLP's and autoencoders are non-existing, at least in the structure of the networks. The differences becomes apparent in the use-cases for the two networks, and in the training procedure. Autoencoders are symmetrical in respect to the hidden layer, but so can be MLPs. Since AEs are structurally MLP networks, the training algorithm is usually the same Back-Propagation (BP) method. However it's noteworthy to mention, that autoencoder can be trained with re-circulation algorithm, or it's variant the *GeneRec* (*generalized re-circulation algorithm*), but this has more to do with neuroscience [2], and no studies using it were found. The difference in the training phase between autoencoders and MLP's is the target of minimization. MLP's minimize the *classification error*, whereas autoencoders minimize the *reconstruction error* shown in equation 2.1 (Goodfellow, Bengio, and Courville 2016). So autoencoders encoder function compresses the input to code, and decoder decompresses the code to reconstruction the original input. The error between original input $x$ and reconstruction $y$ is to be minimized. This way the autoencoder is forced to learn only the salient features of the input, and ignore the rest, less meaningful features (Goodfellow, Bengio, and Courville 2016). Note that autoencoder do require a minimum of three layers to work (input-code-output), and can thus be classified as deep networks (like in Goodfellow, Bengio, and Courville 2016). However, the number of code layer can be increased to increase the depth of an autoencoder, and creating a "real" deep network.

2. O'Reilly, R. C. (1996) Biologically plausible error-driven learning using local activation differences: The generalized re-circulation algorithm. Neural computation, 8(5), 895-938.

$$\arg\min_{\theta,\theta'} \sum_{i=1}^{n} L(x^{(i)}, y^{(i)}) \tag{2.1}$$

We can now also see why certain changes needs to be made if the layers are same size or over-complete. In either cases the network can learn identity function, or any bijective function, and the reconstruction would match exactly to the input. The problem is, that code vector would also match, and therefore contain no additional information about distribution of training data.

### 2.3.2 Convolutional neural networks

Convolutional Neural Networks (CNNs) were first proposed by Cun et al. 1989 and have since gained a lot of popularity. While originally they were designed to extract information from images, they work with other types of data also. Specifically if data can be interpreter as signals, CNNs may be used. For example time-series data, videos, speech, images etc. The core of CNNs is the mathematical convolution operation. Generally CNNs are standard neural networks, but instead of simple matrix multiplication, convolution is used at least in one layer (Goodfellow, Bengio, and Courville 2016 ). Convolution is a linear operation where two signals $x$ and $y$ are convolved producing a third signal $s$. Signals meaning functions in this case. Formally this is

$$s(t) = (x * w)(t) = \int x(a)y(t-a).$$

Note that this formal notation does have some restrictions with regards to functions (Goodfellow, Bengio, and Courville 2016). Function $x$ is often called as *input*, function $y$ as *kernel* and output as *feature map*, especially when dealing with CNNs these are used. Since data in neural network applications is rarely truly continuous, this form of convolution is not used when dealing with CNNs Instead a discrete one is used. In discrete convolution, the integral is simply replaced with a sum

$$s(t) = (x * w)(t) = \sum x(a)y(t-a).$$

This notation (and the continuous one) can be easily expanded to multiple dimensions, such as two dimensions for images, or three for images with spectral axis. For image $I$, and two

dimensional kernel $K$ the 2-dimensional discrete convolution is

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) =$$

$$(K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

Since the data used in this thesis contains 3 dimensions (two spatial and one spectral), 3-dimensional convolution is used.

Also note that when speaking about machine learning convolution can also mean a similar cross-correlation operation

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

These two are sometimes used interchangeably (Goodfellow, Bengio, and Courville 2016).

Convolutions are used in neural networks because by making the kernels smaller than the input they can find features present in a small part of the input data. For images this is especially useful: traditional neural networks find features that apply across the whole input. This also allows CNNs to find common spatial features from the images, such as edges. Another added benefit for this is the reduction in memory consumption. Memory requirements for a fully connected layers is a lot larger than for a convolutional one. This property is called spare interactions (Goodfellow, Bengio, and Courville 2016 ), or sometimes local connectivity; i.e. each neuron in output is connected to some local area, not the whole input as in fully connected networks. A parameter governing the size of this neighborhood is the size of the kernel, and is sometimes called *receptive field* of the output neuron.

Convolution as an operation might not be that clear from the mathematical notation. In figure 10 a simple visualization of a 2-dimensional convolution can be seen. Indistinctly in CNN, the kernel corresponds to some feature of interest in the image, for example a shape. When convolution is run with this kernel, the output tells how prominent that feature was in each section of the input image. Note that the outermost areas of the input in figure 10 are 0's this is *padding* of the image, and is one of the parameters of the convolutional network. Other parameters include the *stride* of the kernel. Meaning how much the kernel is moved across the image. In equation **??** this corresponds to the amount $n$ and $m$ are increased across

the sum. Recently one more optional parameter for the convolution is introduced: *dilation*. Normally the kernels are continuous, but it it possible that they may have gaps in them, making kernels checkered" (Yu and Koltun 2015).

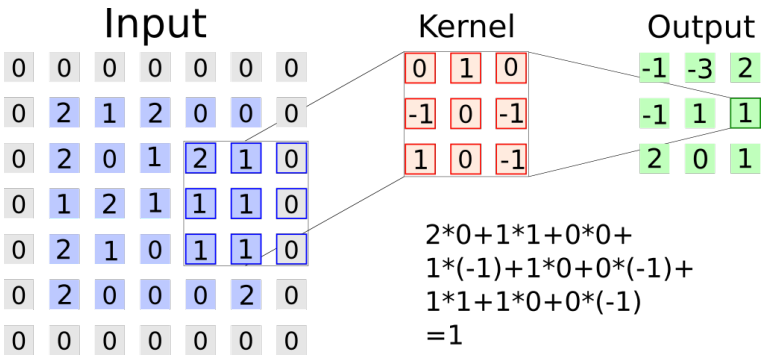



Figure 10: Example of 2-D convolution with padding, $3 \times 3$ filter and a stride of 2

Usually the convolutional layer in a neural network is divided int three parts. The first one being the convolution, second being activation, and third *pooling* (Goodfellow, Bengio, and Courville 2016 ). As with standard neural networks, in the activation the feature-maps generated by convolutions are run through some (non-linear) activation function. The third (optional, e.g. *AlexNet*) part, often *max-pooling* is a fairly simple operation, in which for example 2-dimensional input is shrunk to a smaller size by dividing the input into section, and for example storing only the largest value from them. An example can be seen in figure 11. Pooling operation also has kernel size and stride parameters. The function of pooling is to make the convolutional layer invariant with respect to small changes, making the presence of a feature more interesting than the exact location of it (Goodfellow, Bengio, and Courville 2016). As an added bonus they serve to reduce the size of the feature maps, an important bonus when making deep networks, with multiple convolutional layers.

Since any neural network, including CNNs, need to be trained convolutional layers alone are not enough: one needs some output-layer to train the network. Usually after a number of convolutional layers, with pooling or not, one or two fully connected layers are added. The last of these being the output layer. The training is done using traditional BP method with respect to some training targets, for example classification labels. In CNNs the weights optimized by the training procedure correspond to the convolutional kernels. This way when training the network for say a classification, kernels that represent some meaningful features

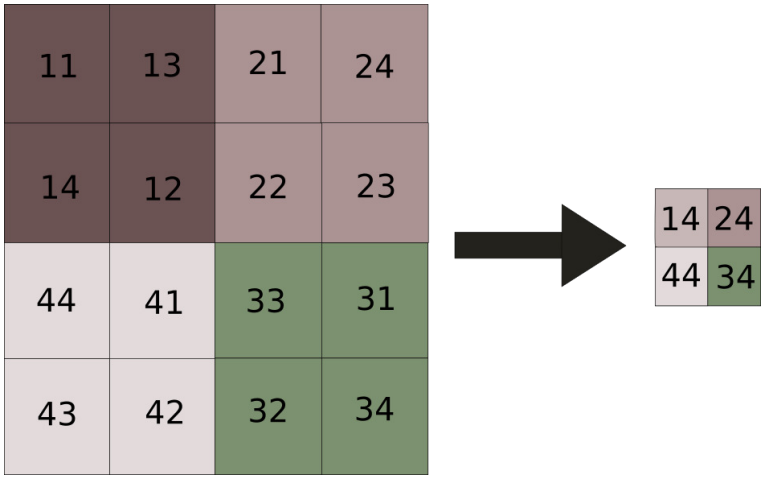



Figure 11: Max-pooling operation with $2 \times 2$ filter and a stride of 2

for these classes are learned. Since the use-case of CNNs is usually fairly complex, convolutional layers are stacked on top of another to form deep CNNs. Since the feature maps are inputs to the next level of convolutions, each convolutional layer learns more complex features than the one before. In image 12 this can been seen: first level features are very simple, and later more complex.

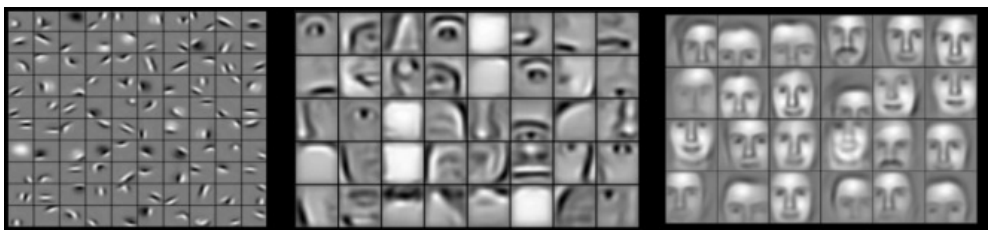


Figure 12: Example of kernels learned by a 3-layer CNN (Lee et al. 2009)

Comprehensive study of CNNs would constitute a thesis on its own, and this section only aims to provide some basics. CNNs being a very interesting field of study a lot of different variations and tweaks exists, and not included in this section. The reader should however have now basic knowledge of CNNs, and be ready to move to the next part.

### 2.3.3 Convolutional autoencoders

CAEs are the combination of the convolutional operations and autoencoder networks. This kind of network was proposed Masci et al. 2011 in an attempt to develop an unsupervised

neural network that could efficiently work with image data. Any image can be converted to one dimensional vector which in turn can be fed into an appropriate neural network, for example an autoencoder. The problem is that this kind of transformation on the image loses most of the positional relationships between the points of the image (Du et al. 2017). Masci et al. 2011 wanted to develop a model for neural network where this information is preserved. By combining CAEs with AEs, one is left with a neural network that can learn the best features (kernels in the case of CAE) for the current task. These networks are useful when training large CNNs, which cannot be trained in traditional methods because of the vanishing gradient problem. The kernels learned by CAEs can be transferred to CNN's with similar topology, and further trained using traditional methods (Masci et al. 2011; Du et al. 2017).

Logically the structure of CAE is exactly what one would expect: the two networks stacked on top of one another. An autoencoder which gets the output of a convolution operation as an input. The input is in vector form, but as the input itself is a feature map it already contains positional information about the original image, and the transformation to vector form does not lose *as much* information as on the raw data. It does lose some information, higher level abstraction of the data (i.e. features of features), and it's up to the user to decide which level of abstraction is wanted. Like with CAEs, pooling operation can, and should be used in CAE networks (Masci et al. 2011). In the encoder phase pooling works the same as with CNNs. Problems arise in decoder phase. As pooling is not an injective operation and thus not invertible, one needs to reverse it somehow. This is called *unpooling* (also *up-sampling*), and it functions to reverse the pooling operation (Zeiler, Taylor, and Fergus 2011). Like with pooling, unpooling can be done in different ways. The way used in this thesis is depicted in figure 13. Note that pooling operation does lose some of the information contained in the input. No matter what method for pooling is used, this lost information can be recovered only in special cases (e.g. the feature maps has only single values).

Logically a CAE (without pooling) is two networks stacked on top of another, but mathematically it is an autoencoding network in which the input and code vectors are convolved (Masci et al. 2011). The encoding function is shown in 2.2 and decoding in 2.3. $\delta$ is some activation function, $*$ is 2-dimensional convolution, $x, y$ and $z$ are the input, code and reconstruction vectors, $W, b, W', b'$ are the weight matrices and bias vectors. Note that mathematically the

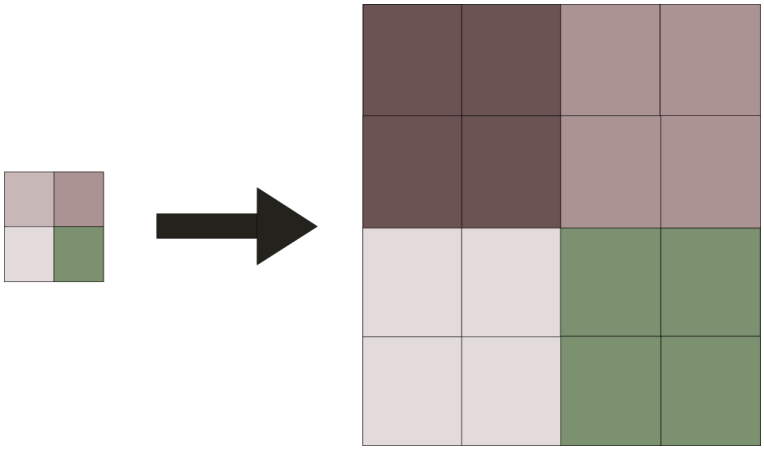


Figure 13: unpooling operation

decoding, i.e. *de-convolution* is also a convolution. So in practice CAE network is a collection convolutional layers arranged to form an autoencoding structure (i.e. the network contain two mirrored parts). CAEs are trained similarly to standard autoencoder: using BP method with some error function, for example Regulized Mean Squared Error (RMSE)).

$$y^k = \delta(x * W^k + b^k) \tag{2.2}$$

$$z^k = \delta(\sum_{k \in H} y^k * W'^k + b'^k) \tag{2.3}$$

Like with standard autoencoder, there exists variations for CAEs. Masci et al. 2011 proposed one of these: the Stacked Convolutional Autoencoder (SCAE), which is analogous to stacked autoencoder. In these networks the output of previous layer is the input of the next layer. Du et al. 2017 proposed a variation of CAE in which autoencoder were replaced with *denoising autoencoder (DAE*, the resulting network (*convolutional denoising autoencoder (CDAE)* being less prone to noise. Du et al. 2017 also included additional processing in their network, namely *whitening* layers. Whitening is an operation that removes correlation from the data. The topology of the network proposed by Du et al. 2017 is shown in figure 14. The network in question uses several CDAE networks and as such forms a deep network called *stacked convolutional denoising autoencoder (SCDAE)*.

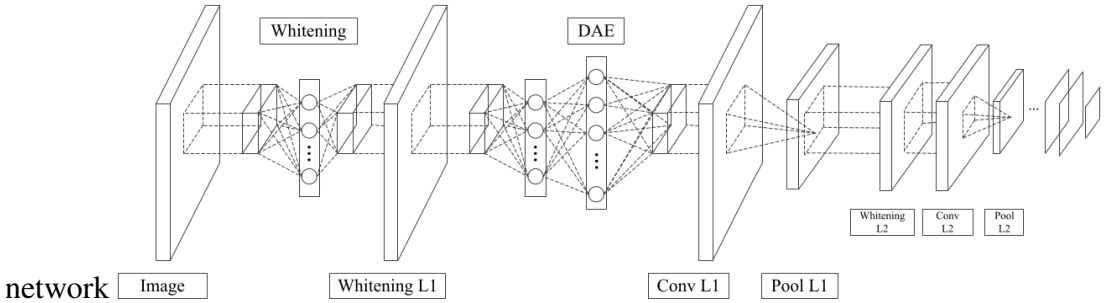



Figure 14: SCDAE-network (Du et al. 2017)

In the first section of this chapter some basic background information on hyperspectral imaging and anomaly detection was provided. More detailed presentation of the HDBSCAN/GLOSH anomaly detection algorithm was also included. In the second part three neural networks were formally introduced. The first two, AEs and CNNs being the building blocks for the third: the CAE. The main goal of this chapter was two provide the reader, first some background knowledge on the problem, but more importantly the two main tools used in this thesis: HDBSCAN and GLOSH algorithms and CAE neural networks.

# 3 Materials and methods

The previous chapter presented the basic building blocks of the proposed method for the detection of anomalies. And this chapter will continue to combine these blocks to form one possible configuration of the method. The chapter is divided to two parts: the first part will present the imaging data used in this thesis: where and how it was gathered and how it was processed, and the second part will construct block by block the method used to detect anomalies from the data. The order of these sections is not arbitrary; the structure of the data creates some constrains for the method, and the data is presented first (though the reverse also holds in some parts).At the end of this chapter the reader should have an understanding how the method works and how the experiment was designed. The results of this experiment will be presented in chapter 4

All of the techniques and algorithms presented in this paper were implemented on Python 3. Convolutional autoencoders were build with Keras framework, using Google's Tensorflow with GPU backend.

## 3.1 Materials

Since the fundamental purpose of the proposed method is to detect anomalies from a large datasets the data gathering process was a bit problematic. There isn't that many readily available HSI datasets, and with the added restrictions of size and the *type* of data, choices drop to zero. The type of data in this case means the kind that isn't too heterogeneous. If the images in the dataset are for example of distinct objects then the data would probably be too heterogeneous and most, if not all, images would be classified as anomalies. Thesis advisor did propose the use of openly available satellite data, specifically data from ESA's Sentinel 2 satellites. Thankfully this data is freely available through ESA's Copernicus Open Access Hub, and with the provided Sentinels Application Platform (SNAP)-application fairly easily transformed to usable format.

ESA's Sentinel 2 satellite system consist of two identical satellites: Sentinel 2A and 2B in the same polar orbit phased 180 degrees apart. Both satellites contain MSI instrument, which

is technically not a hyperspectral sensor, but as the name implies a multispectral one. These MSI sensors collect data from 13 bands ranging from VIS[1] to SWIR[2]. Information about these bands can be seen in table 1. Other specifications are: radiometric resolution (i.e. bitdepht) of 12 bits, temporal resolution (i.e. revisit time) of 5 days on equator and swath width of 290$km$ (European Space Agency (ESA) 2017).

| Spatial Resolution (m) | Band Number | S2A | | S2B | |
|---|---|---|---|---|---|
| | | Central Wavelength (nm) | Bandwidth (nm) | Central Wavelength (nm) | Bandwidth (nm) |
| **10** | 2 | 496.6 | 98 | 492.1 | 98 |
| | 3 | 560.0 | 45 | 559 | 46 |
| | 4 | 664.5 | 38 | 665 | 39 |
| | 8 | 835.1 | 145 | 833 | 133 |
| **20** | 5 | 703.9 | 19 | 703.8 | 20 |
| | 6 | 740.2 | 18 | 739.1 | 18 |
| | 7 | 782.5 | 28 | 779.7 | 28 |
| | 8a | 864.8 | 33 | 864 | 32 |
| | 11 | 1613.7 | 143 | 1610.4 | 141 |
| | 12 | 2202.4 | 242 | 2185.7 | 238 |
| **60** | 1 | 443.9 | 27 | 442.3 | 45 |
| | 9 | 945.0 | 26 | 943.2 | 27 |
| | 10 | 1373.5 | 75 | 1376.9 | 76 |

Table 1: Sentinel 2 satellites MSI instrument specifications (European Space Agency (ESA) 2017).

Sentinel 2 data is categorized to different products, depending on how much the raw data is processed. The raw sensor data (level 1B) is not provided to public at large. Instead the data is compiled to top-of-atmosphere reflectance in $100km \times 100km$ cartographic geometry[3] (European Space Agency (ESA) 2017). This data is further processed to bottom-of-atmosphere reflectance (level 2A) product on the SNAP program.

All imaging data used in this thesis was gathered through Copernicus Hub, specifically S-2B

1. Visual light, portion of EM spectrum ranging from about 390nm to 700nm
2. Short Wave infrared, portion of EM spectrum ranging about 1000nm to 2500nm
3. UTM/WGS84 projection

PreOps Hub[4], based on some rough criteria (mainly homogeneity of data). After some rough visual scanning of the data, a dataset consisting of 13 images was chosen. Geographically all these images are from the Alaska Peninsula, and locations of the used images can be seen in figure 15. RGB color images of the used data are listed in appendix A The data is loaded to SNAP and exported to ENVI format. Like shown in table 1, the bands are of three different spatial resolutions: $10m$, $20m$ and $60m$, these correspond to different size layers: $10980 \times 10980$, $5460 \times 5460$ and $1830 \times 1830$ pixels respectively. Before exporting data from SNAP, layers were resized based on the most restrictive: $1830 \times 1830$. Downsampling was done using mean method. From this point onward all processing is done using Python.

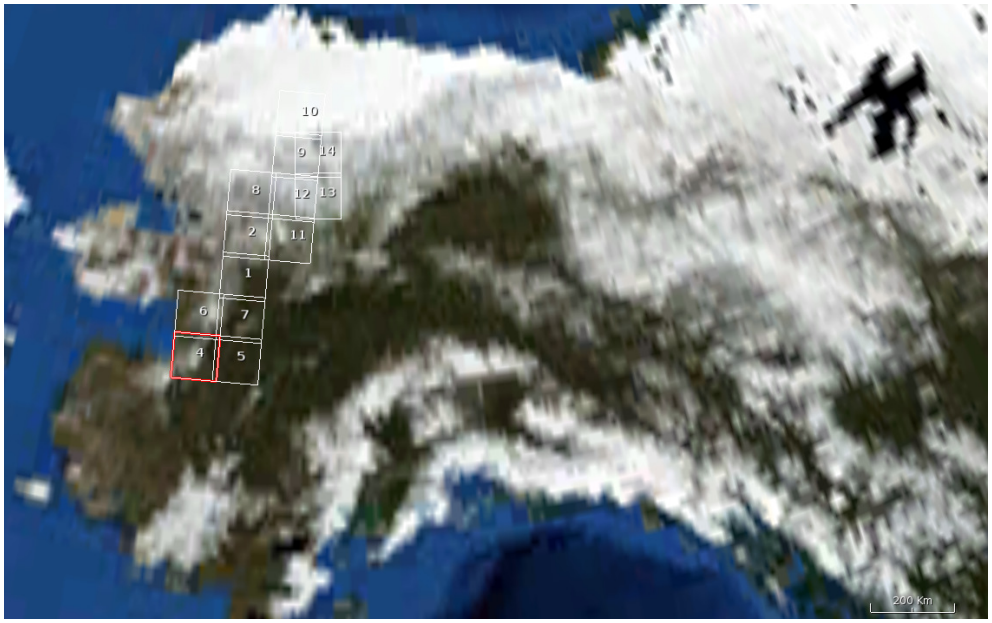



Figure 15: Geographical location of the used data

Since SNAP exports each band as it's own image, some further processing was required to combine each band to a single image cube. At this point the images are also relatively large, and each of these was further split into windows of $128 \times 128$ pixels. Since the dimension of the images are not divisible by this window size, there is some overlap on the right and lower edges. The window size of 128 was decided after some reflection on performance, number of images and the depth of the network. Since convolutional layers are coupled with pooling layer, the dimensions of the images need to be chosen with this in mind. Specifically

4. https://scihub.copernicus.eu/s2b

the dimension should be divisible by the "size" of the pooling times the number of pooling layers. Each of the max-pooling layers divide the dimensions of the input data. So by using two max-pooling layers, both dividing the dimensions by two, the input data dimensions need to be twice divisible by two. At this point the data consists of 2925 npy files (each image is windowed to 225 windows), each containing a single $128 \times 128 \times 13$ matrix. While these files do contain all 13 bands, only 12 are used because of the pooling operations, band 9 being the unused one. With 12 bands, the depth of the network is also to 2 layers, or more precisely the number of pooling layers is limited to 2. With further reduction of bands to 9, this could be increased to 3, but to preserve as much data as possible, this was disregarded. This data will constitute the training dataset for the network.

One of the more difficult problems when dealing with unsupervised methods, is the validation of results. For labeled data it's simple to calculate different performance metrics, but when no labels are available values such as True Positive Rate (TPR) or FPR cannot be computed: what is positive value when there are no labels? There are some methods to overcome this problem on some cases. Depending on the method/algorithm used, one might have a feasible method of validation, but no such luck for this case. One could manually search anomalous areas using SNAP, in effect label the data, but this method does not work that well for hyperspectral images. Since human eye cannot see beyond visual range, it would require massive amounts of labor to both learn what is normal and then to find abnormal areas in hyperspectral images. One of the purposes of the proposed method was to outsource this kind work to a machine.

To combat the problem caused by the lack of labels, a method to synthetically add anomalies to the used data was proposed. As mentioned before, the definition of anomaly is not as clear cut as it would seen. The first task of creating this synthetic anomalous data, was to decide upon what kind and how to generate these synthetic anomalies. A relatively simple way was chosen: increase the values of pixels based on the distribution of the raw values. This method does not differentiate between spatial and spectral anomaly, but instead creates ones that are both. This process began by studying the distribution of each band. Distribution of bands can be seen in figure 16, and information about the mean and error of the bands in figure 17. Note that raw values in band 10 are a lot smaller than in other bands. Because of this

the standard deviation of this band is not visible in figure 17. the method for adding these anomalies to the data is fairly rough, and based on relatively simple statistics. Still it was thought to be adequate, but since the purpose of this thesis is to provide proof-of-concept implementation for the method.

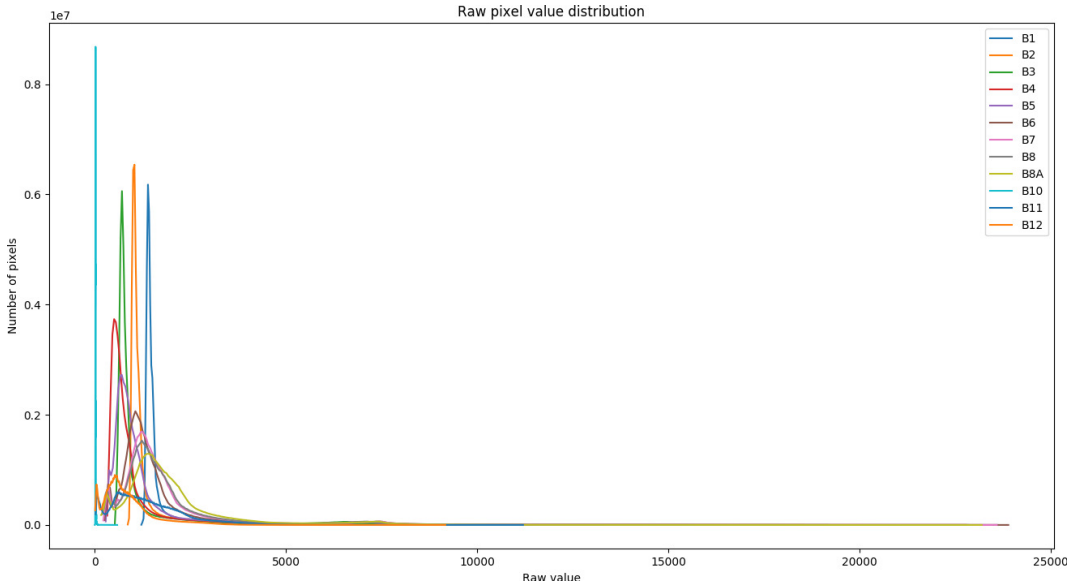



Figure 16: Distribution of raw pixel values per band

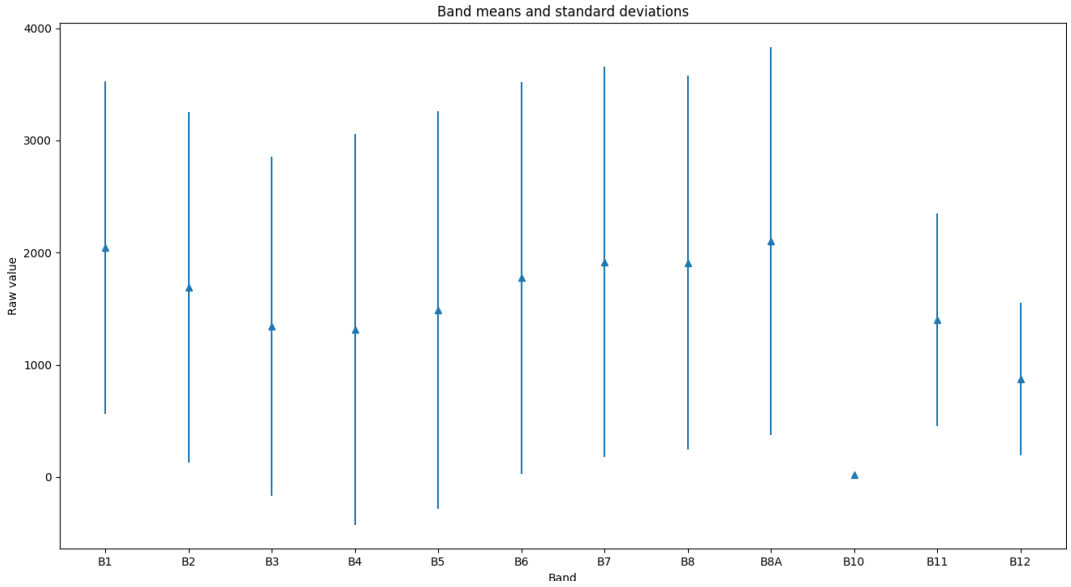



Figure 17: Means and standard deviations of raw values per band

The first step in the actual generation of the synthetic data was to decide on some parameters and to calculate vectors $\vec{v}_{mean}$ and $\vec{v}_{std}$ containing the means and standard deviations for all

bands. Parameters for the generation algorithm were: $p_{anom}$ = probability of anomalous image, $s_{anom}$ = size of anomaly and $[c_{min}, c_{max}]$. Also values [$c_{min}$ and $c_{max}$] were chosen to give the multiplication coefficients on how many standard deviations are to be used for the anomaly. these values were chosen as follows: $p_{anom} = 0.05$, $s_{anom} = 20$ and $[c_{min}, c_{max}] = [3,5]$. The process of generating the synthetic data is as follows: Firstly from the 2925 images a random subset was chosen based on $p_{anom}$. Next for each anomalous image $M_{img}$ a location of the anomaly was randomly chosen, and a mask was created containing zeroes everywhere except at the position of the anomaly where the values were 1. For example if images were of size 3x3 and $s_{anom} = 2$ the mask could be

$$mask_{anom} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The next step is to create matrix

$$M_{rand} \in \mathbb{R}^{s_{anom} \times s_{anom} \times 12}, M_{rand_{i,j,k}} \in [c_{min}, c_{max}), \forall i, j \in \{1, \ldots, s_{anom}\} \text{ and } k \in \{1, \ldots, 12\}$$

and

$$\hat{M}_{coeff} = M_{mean} + (M_{std} \odot M_{rand})$$

$$M_{mean_{i,j}} = \vec{v}_{mean}, \forall i, jin \in \{1, \ldots, s_{anom}\}$$

$$M_{std_{i,j}} = \vec{v}_{std}, \forall i, jin \in \{1, \ldots, s_{anom}\}$$

where $\odot$ denotes *element-wise multiplication*. Matrix $\hat{M}_{coeff}$ is an anomaly specific matrix, that contains information on the magnitude and shape of the said anomaly. The next step is to expand matrix $\hat{M}_{coeff}$ to a new matrix $M_{coeff}$ with same shape as $M_{img}$,. This new matrix contains 1, except for the masked area where it contains the old $M_{coeff}$ matrix. That is

$$M_{coeff_{i,j}} = \begin{cases} \vec{0}, \text{ if } mask_{anom_{i,j}} = 0 \\ \hat{M}_{coeff_{\hat{i},\hat{j}}}, \text{ if } mask_{anom_{i,j}} = 1 \end{cases}$$

Next the this matrix is divided element-wise with the original image matrix, and we get the final multiplication matrix $M_f = M_{coeff} \oslash M_{img}$. The final anomalous image is generated with the help of this matrix

$$M_{synthetic} = M_{img} \odot M_{coeff}$$

In this matrix original data is preserved, except for the location masked by $mask_{anom}$, where pixel values are increased based on the random factor explained above. Labels for the validation phase, and the location of anomalies are saved (figure 18). Full scale binary masks are also created for visualization purposes. One of these can be seen in figure 19.

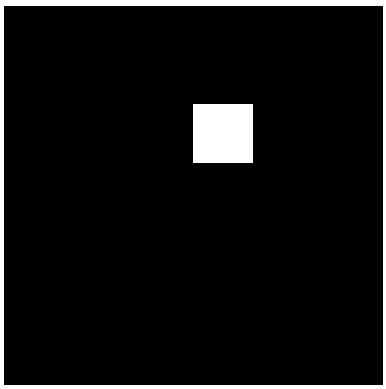


Figure 18: Location of synthetic anomaly for image_01, section 4

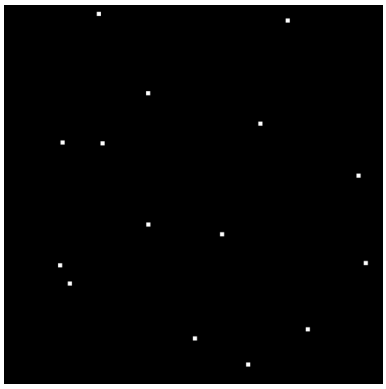


Figure 19: Location of synthetic anomalies for image_01

## 3.2 Methods

The foundation of this method is the use of CAEs, especially the deep variety. The method itself is quite simple: by using convolutional autoencoders to extract common meaningful features from the data, and by analyzing these features one can estimate which images or areas of images are normal, and which are not. The method itself works in three phases: in the first phase the SCAE is trained. In the second phase the trained network is used to extract the raw feature-maps, which further distilled into the final feature. In the third phase

these features are run through some anomaly detection algorithm, in the case of this thesis HDBSCAN/GLOS.

Neural networks themselves are a vast field of study. Without any prior knowledge on the performance of CAEs, the build of the method would be trial-and-error. Because of this, some preliminary testing of CAEs was required, and worked as a starting point. The first step were to study the performance of the SCAEs networks on simple black-and-white images. The point of these test were to establish some base knowledge on network sizes, optimization methods, loss functions and kernel sizes. Detailed results of these test are not presented since they are outside the domain of this thesis, but based on these tests the optimizer was chosen to be Adadelta and as a loss function: Mean Squared Error (MSE). Some tests were also made with the number and size of kernels, and the number of convolutional layers. In the end these did not weight much on the final structure of the network due to practical restrictions imposed by the structure HSI data, and by computer performance. Especially the number of bands in the data restricted the use of max-pooling layers. Though these pooling layers are not always essential (for example: Du et al. 2017),

### 3.2.1 Phase 1: structure and training

After the preliminary test, the actual encoder to be used was decided to consist of 2 convolutional layers, both containing 48 kernels of size $5\times5\times5$. Since the data used is hyperspectral, it was deemed best to use 3-dimensional kernels to capture both spatial and spectral features at the same time. Other methods could also have been used, for example: 2-dimensional kernels over different bands learning spatial features. Each convolutional layer was coupled with a max-pooling layer, all of which divided the dimensions of input by 2. All the convolutional layers used padding so, that input and output shapes of each layer are the same. No bias was used on any layer. Feature scaling was also applied to dataset before the training

$$D_{train} = \frac{D_{raw} - min(D_{raw})}{max(D_{raw} - min(D_{raw})}.$$

The corresponding decoders structure is a mirror of the encoder with max-pooling layers substituted for up-sampling layers, and the one extra convolutional layer at the end.The structure of this network is summarized in table 2. Total number of trainable parameters was 876,193.

As the purpose of this network was to only demonstrate the feasibility of this method, the net was trained only with 10 epochs[5]. Even with the low number of epochs, it was able to achieve reconstruction error of 0.0017. Testing showed that this could be reduced further if required. The network trained using the full 2925 (non-modified) images. The training of this network is the single most time consuming operation and something that can be done before the actual application of the method. During the creation of thesis the network was trained beforehand on a more suitable computer and then stored for later use.

Table 2: Summary of used neural network

| **Layer** | **Output shape** | **Parameters** |
|---|---|---|
| *Input* | (128, 128, 12) | 0 |
| *Conv1* | (128, 128, 12, 48) | 6048 |
| *MaxPool1* | (64, 64, 6, 48) | 0 |
| *Conv2* | (64, 64, 6, 48) | 288048 |
| *MaxPool2* | (32, 32, 3, 48) | 0 |
| *Deconv1* | (32, 32, 3, 48) | 288048 |
| *Upsampling1* | (64, 64, 6, 48) | 0 |
| *Deconv2* | (64, 64, 6, 48) | 288048 |
| *Upsampling2* | (128, 128, 12, 48) | 0 |
| *Deconv3* | (128, 128, 12, 1) | 6001 |

Since the innate property of any autoencoder is the ability to learn meaningful features from the data, the network should now have an idea of what is normal in the training dataset. Note that unlike most unsupervised anomaly detection methods, the training data itself is not as a whole thought as normal. Like with any method in the domain of machine learning, what is considered normal is widely reliant upon the used training data. But the way this network works allows it to learn commonly occurring features from the data, and so the data itself can contain anomalies. This makes the method quite robust in terms of training data. Normally it would require a fair amount of manual work to go through the training data, and make sure that there are no anomalies in it, or to label them. This also enables the method to use

5. One forward pass and one backward pass of all the training examples.

same data in train and detection phases, often anomaly detection uses distinct train and test datasets.

Note that, unlike in figure 12, the kernels learned by this network cannot be visualized in any meaningful manner. Since the convolution is 3-dimensional, the kernels are also 3-dimensional, and thus any visualization would be meaningless to human eye. This makes the manual observation of the internal workings of this network impossible, a property commonly observed with deep neural networks.

### 3.2.2 Phase 2: feature extraction

After the training of the SCAE is done, this network can be used to extract feature-maps from the data. Note that since it's the encoder part of the network that actually learn the features, the decoder is unused after the initial training, and if necessary could be discarded completely. Before the actual feature-map generation, the test data was scaled similarly to the training data:

$$D_{test} = \frac{D_{test\_raw} - min(D_{test\_raw})}{max(D_{test\_raw} - min(D_{test\_raw})}$$

The feature extraction can, and was done in parallel for all images at the same time, but next the road of a single image is explained.

The image is run through the network, and feature-maps (also: activation maps) are collected. For each of the convolutional kernel a single map is created. Since each convolutional layer in a this network is a collection of multiple separate networks (one for each kernel in the layer), the total number of feature-maps is the total number of kernels in all the convolutional layers of the encoder part of the network. In the network used in this thesis that means, that a total of 96 feature-maps were collected for each image. Each feature-map itself is a two dimensional matrix, the shape of which is dependent on it's location on the network ( the pooling causes layers to decrease in size the deeper they are in the network). The sizes of feature-maps is shown in table 2. After the image is run through the network, a total of $128 \cdot 128 \cdot 12 \cdot 48 + 64 \cdot 64 \cdot 6 \cdot 48 = 10616832$ features per image are collected. Now this is a lot of information, and the storage alone for all images takes 276GB using 64bit floating point numbers. 10 Million features per data point is also quite impractical for any anomaly

detection algorithm; the sheer amount of computing power required to analyze dataset of this size in a practical amount of time is massive. This moves us to the second function of phase two: to take these raw feature-maps, called $f_{c1} \in \mathbb{R}^{128\times128\times12\times48}$ and $f_{c2} \in \mathbb{R}^{64\times64\times6\times48}$, and to extract more compact feature-vectors from them. There is a lot of different ways to extract these features, and for demonstration purposes two were used in this thesis. The feature-vectors themselves were stored in 32bit format.

Each of the feature-maps tells on how largely each feature is represented in an area of image, and themselves can be considered as images. Therefore they need to be transformed to vector form. Since each feature, by the definition CAEs, is a commonly occurring feature, abnormally large values in the feature-maps are generally not interesting. They can of course contain anomalies, but from the point of view of this thesis they are not to be considered anomalies. The smallest values of feature-maps are also not interesting: there might be a portion of a image, say cloud cover, in which some features can be completely absent, but who do occur in the rest of the image. Under this reasoning storing information about the largest values of each feature-map gives the information we want. Large maximas are considered normal, but small maximas are anomalous: a small maxima across a single feature-map tells that this image did not have some feature that is considered common. both of these features were extracted for each image, are based on this reasoning.

At the beginning only two features were extracted. A low-dimensional simple one for general features, and a higher-dimensional one for complex features. For both of features, feature-maps from both convolutional layers were concatenated along the fourth axis, forming a single feature matrix $f_c \in \mathbb{R}^{128\times128\times12\times96}$. Since the dimensions of these feature-maps mismatch (table 2), matrix $f_{c2}$ was up-sampled to match the dimensions of matrix $f_{c1}$. Up-sampling was done simply with Kronecker product (i.e. the operation shown in figure 13). The first feature vector $\vec{f}_1$ was then computed by

$$\vec{f}_{1_l} = max(f_{c_{i,j,k,l}}) \begin{cases} i,j \in \{1,\dots,128\} \\ k \in \{1,\dots,12\} \\ l \in \{1,\dots,96\} \end{cases} \qquad \vec{f}_{1_l} \in \mathbb{R}^{96}$$

So feature $\vec{f}_1$ contains information about the maximas of each feature-map across all bands.

The creation of the second feature was a bit different. It is essentially the same as $\vec{f}_1$, but it does not compute the maxima over all the bands. Instead a maxima is computed for each band individually. The second, more drastic feature is the fact, that each $128 \times 128$ is image further subdivided int 16 regions in a 4-by-4 grid. So feature 2 is not actually a vector but a matrix instead. This matrix is created by using max-pooling layer of size $2 \times 2$ with a stride of 2 along the first and second dimensions. This max-pooling layer functions as both the maximum function to collect feature data and as a tool to section the image to 16 regions at the same time. From the raw feature matrix this produces a matrix of shape $4 \times 4 \times 12 \times 96$ which is then resized to the final shape. Programmatically this feature is handled in a same way as the full images in section 3.2. Meaning that is each image is split into smaller images of size $32 \times 32 (128/4)$, and they are handled as their own images. The values of this feature matrix are

$$\hat{f}_{2_{i,j,k,l}} = max(\hat{f}^{i,j}_{c_{k,l}}) \begin{cases} i,j \in \{1,2,3,4\} \\ k \in \{1,\dots,12\} \\ l \in \{1,\dots,96\} \end{cases} , \hat{f}_2 \in \mathbb{R}^{4\times4\times12\times96}$$

where $\hat{f}^{i,j}_c \in \mathbb{R}^{12\times96}$ is a submatrix of $f_c$ given by

$$f_c = \begin{bmatrix} \hat{f}^{1,1}_c & \hat{f}^{1,2}_c & \hat{f}^{1,3}_c & \hat{f}^{1,4}_c \\ \hat{f}^{2,1}_c & \hat{f}^{2,2}_c & \hat{f}^{2,3}_c & \hat{f}^{1,4}_c \\ \hat{f}^{3,1}_c & \hat{f}^{3,2}_c & \hat{f}^{3,3}_c & \hat{f}^{3,4}_c \\ \hat{f}^{3,1}_c & \hat{f}^{4,2}_c & \hat{f}^{4,3}_c & \hat{f}^{4,4}_c \end{bmatrix}$$

By reshaping we get $\hat{f}_2 \to f_2 \in \mathbb{R}^{4\times4\times1152}$, where $1152 = 12 \cdot 96$. The use of sectioning increases the accuracy of features. Since each atomic area (essentially a single data point), is smaller, there is less change that anomalies are drown in the normal data.

At this point, it was thought that since the used network is deep one or meant to be, features from the second layer should be considered individually. This called for two more features. These third and fourth features basically duplicates of first and second, with the exception that they only took as a raw features (though up-sampled to mach the input size) from the second convolutive layer. The size of all features is summarized in figure 20.

Like mentioned at the beginning, this feature extraction was done in parallel for all images,

$$\vec{f}_1 \in \mathbb{R}^{96}$$

$$f_2 \in \mathbb{R}^{4\times4\times1152}$$

$$\vec{f}_3 \in \mathbb{R}^{48}$$

$$f_4 \in \mathbb{R}^{4\times4\times576}$$

Figure 20: Summary of extracted features

not for single ones as explained here. As such, the features $\vec{f}_1$ and $\vec{f}_3$ are no actually vectors, but matrices, where each row corresponds to a single image. Similarly, features $f_2$ and $f_4$, which are already matrices, gain one additional dimension to mark the image from which of the original 2925 images they belong to. The output of phase 2 was thus a collection of 4 matrices, one for each feature.

### 3.2.3 Phase 3: anomaly detection

In the third and last phase of this method, the extracted features from phase two are run through some anomaly detection algorithm. since each phase is more or less independent from the others, the anomaly detection algorithm can be chosen freely, depending on the problem setting and goals. It's conceivable to even use supervised algorithms if labeled data is available, though in this thesis the point is in unsupervised methods.

Considering the problem setting proposed in this method, the choice for the anomaly detection algorithm was not a clear cut problem. Most of the anomaly detection algorithms, even unsupervised ones depend on some parameters that are data-specific. Since this in this thesis aims to demonstrate anomaly detection method that works out-of-the-box, most algorithms were ruled out because they require data-specific optimization of parameters. The choice of anomaly detection algorithm was GLOSH/HDBSCAN introduced in section 2.2.1. While HDBSCAN still requires parameter $p_{min}$, it can be fairly easily calculated from data (say size of data), and this was deemed fine. The implementation of the used algorithms was provided by McInnes, Healy, and Astels 2017.

The GLOSH/HDBSCAN algorithm requires two parameters: in addition to the already men-

tioned $p_{min}$ for the minimum number of points, an another one for the threshold of anomalies. Since GLOSH/HDBSCAN algorithm assigns outlier scores (also: *anomaly scores*) $s_{glos} \in [0,1]$ for each point in the dataset, this another value: $t_{anom}$ is required for a decision boundary between normal and anomalous point. The first parameter $p_{min}$ was chosen during some preliminary testing to be 100. Another values were tested but this seemed to work well with this data. This value meaning that at least 3% of all data points need to be "close" to one another to be considered a cluster. Again this value is dependent on the definition of anomaly, but for the purpose of this thesis this was fine. The second value $t_{anom}$ was chosen after manually going through results of GLOSH/HDBSCAN algorithm to be 0.7. These results contain both the $s_{glos}$ scores and real labels for each file. Boundary value of 0.7 worked well: it detected most of the anomalies, while classifying bulk of the data as normal. Note that as previously mentioned, these images do contain non-synthetic anomalies, and these cannot be prevented. There were some synthetic anomalies that were not detected, but a lower $t_{anom}$ value would have resulted in a lot more non-synthetic anomalies. So much so that they were deemed to be false positives, even though no labels for these were available.

The detection phase itself was relatively simple: the extracted features from phase two were loaded, and again values were scaled to $[0,1]$. Without scaling choosing $t_{anom}$ would be difficult and would have to be re-chosen for every new dataset. The scaling for each feature was done simply by

$$f_i = \frac{f_i - min(f_i)}{max(f_i) - min(f_i)}, i \in \{1,2,3,4\}$$

After the features are scaled they are run through HDBSCAN clustering, and GLOSH values are extracted. Each data point for which $s_{glos} \geq t_{anom}$ was labeled as anomaly, and the rest as normal. Since features $f_1$ and $f_3$ are vectors per image and $f_2$ and $f_4$ are matrices image, two different runs of anomaly detection were required. Features $f_1$ and $f_3$ are straightforward clustering and labeling for each image, but features $f_2$ and $f_4$ required additional processing to split each image into sections. These section were then processed as individual images and labeled. For the purpose of validation performance metrics were also calculated. These were done using the smallest divisible element, i.e. for features $f_1$ and $f_3$ whole images and for $f_2$ and $f_4$ sections of images. All of the results were saved for later analysis. Note that since synthetic labels exist only for whole images, the labels for each section needed to be

calculated using position of synthetic anomalies (the positions were saved in generation of synthetic data: figure 18).

In this chapter, we first started by introducing the data used in this thesis. This dataset consistent 13 $100km \times 100km$ multispectral satellite images from Alaska. Another dataset based on this was also introduced. This dataset contained synthetic anomalies, and was created for validation purposes. After the data was introduced, the used method was proposed. This method consists of three different phases. Each of these phases is a distinct component of the whole process, and can be customized for the problem. This makes the method very adaptable, and only the first method: the convolutional autoencoder is, to some degree, fixed. The feature selection can be done in innumerable different ways, and the algorithm in phase three can be chosen based on the goals. In this thesis four features were extracted and GLOSH/HDBSCAN was used for phase three. These are only to demonstrate the feasibility of the core idea, i.e. using convolutional autoencoder to learn normal model for images and thus gaining knowledge on what is not normal.

# 4 Results

In this chapter, results from a total of four different runs of the method introduced in the previous chapter are presented. In section 4.1 results from the unlabeled original data, i.e. the same data used to train the network are presented. In section 4.2 corresponding results for the synthetic data from section 3.2 are presented. Two runs were conducted for both of these datasets. One for full datasets and on for partial.

The structure of the method was described in section 3.2.1, the used features in section 3.2.2, and the anomaly detection phase in 3.2.3. Since the used net, features and anomaly detection algorithm are more or less static parameters (changing them require re-training and re-feature extraction), the optimizable parameters are the two parameters for the HDBSCAN/GLOSH algorithm. To recall these two parameters were chosen to be: $p_{min} = 100$ and $t_{anom} = 0.7$, and all of the results were gathered using these same parameters.

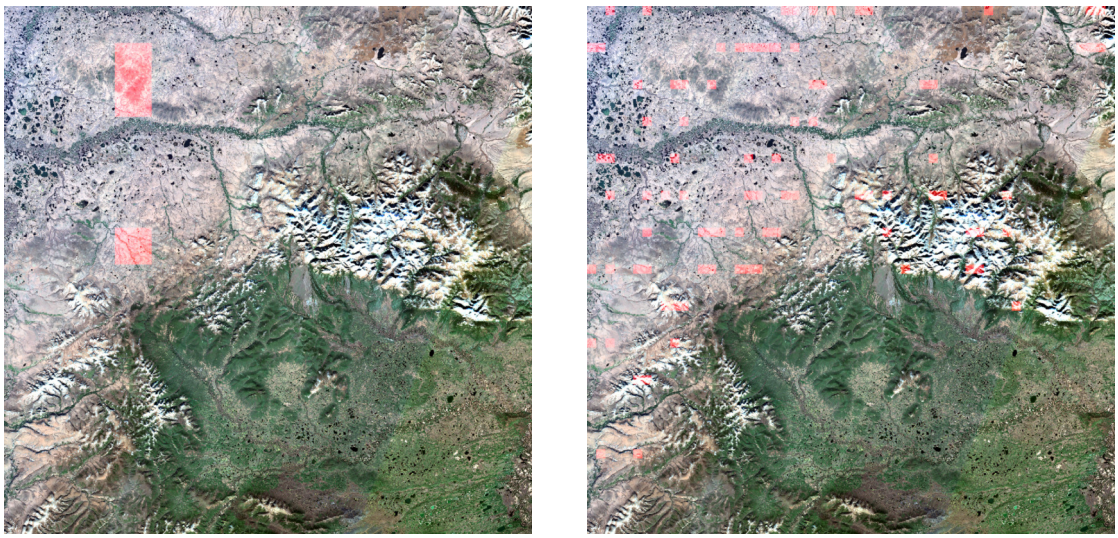
## 4.1 Unmodified data

After some preliminary testing of the method, the very first results gathered were the anomalies detected from the unmodified images. These results correspond most closely to the originally visioned use-case of the method: a large amount of unlabeled data from where anomalies are searched. Anomalies being something on the outskirts of the distribution of the same data from where they are searched. This result gathering process was started by running all of the 2925 unmodified images through the method and saving the results. These small images were then combined back to original full-scale images. In these images each anomaly was masked by a red area. This resulted in a total of 72 images for the original data: two runs, one for all 13 images and another for 5 images. Both containing results from four different features ($4 \cdot 13 + 4 \cdot 5$). Only a select few of these images are presented for demonstration purposes.

Since the unmodified data does not have any labels, nor does the method give out any human-readable explanation of the anomaly, no real validation of these results was done. The anomalies are overlaid to the full RGB images, but since each image is hyperspectral,

these visualizations are subject to personal interpretation, and thus this section does contain a fair amount of speculative conclusions.

The first results were gathered by running all of the 2925 unmodified image through the method, and generating the masked RGB-images where location of anomalies are portrayed. One of these masked images can be seen in figure 21a.



(a) features $f_1$ (b) features $f_2$

Figure 21: Anomalies found in original image_02 using features $f_1$ and $f_2$.

From visually inspecting these images some conclusions were drawn. Firstly most of the uniform areas of the images were labeled as normal, and anomalies were mostly restricted to areas with heavy could-cover (e.g. figure 22) or mountains (e.g. figure 23). Interestingly the frequency of cloud cover anomalies decreased when switching to features $f_3$ and $f_4$ (the ones using only second convolutive layer). This might indicate, that generally the clouds were thought of being normal , but the network could not find any common low level features, only more general ones (and thus not prominent when considering only second layer). This same effect was not noticed for the mountainous anomalies.

Anomalies from the cloud-cover are probably reason to the fact, that only one of the images contained a large amount of clouds (figure 31d). As for the anomalies in mountain areas, no definitive reason was thought of. Although considering the working principle of CAEs (they aim to learn common features across all images), one reason was thought to be plausi-

ble. Most of the anomalies across mountainous areas seem to be centered around the peaks and ridges. Areas which form fractal-like jagged forms. These forms are all, while visually similar, mathematically distinct from each another and thus the CAE cannot find any common features for them. This would result in areas where all of the feature-maps would have abnormally low values compared to the rest of the images (i.e. anomalies).

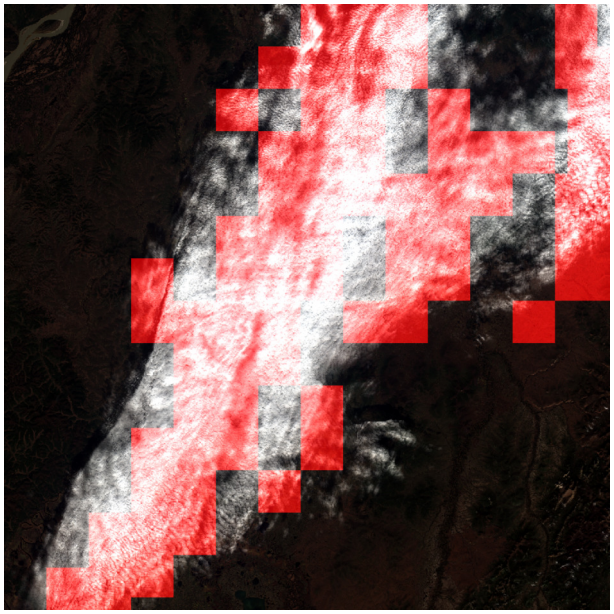


Figure 22: Anomalies found in original image_04 using features $f_1$

It was also noticed, that since the more detailed features $f_2$ and $f_4$ retain the information of each kernel across all bands, they found anomalies not present in the more generalized features $f_1$ and $f_3$. This was predicted behavior, and is likely caused by the small anomalies drowning in the surrounding data when features $f_1$ and $f_3$ were created. This effect can been seen by comparing images 21a and 21b. With some images this actually produced so much more anomalies, that they could even be considered as noise. Such as in image 24a.

It was noted that images was ordered as such that image from the original $1-5$ did not contain as many anomalies as images from the original $6-13$. This corresponds to the geographical location of the images, with $6-13$ containing more mountains and as such, anomalies. Since the anomaly detection phase is based on clustering and is sensitive to the input data, there was a change that these later images masked some more subtle anomalies from the earlier images. Based on this reasoning, a second set of results was collected by

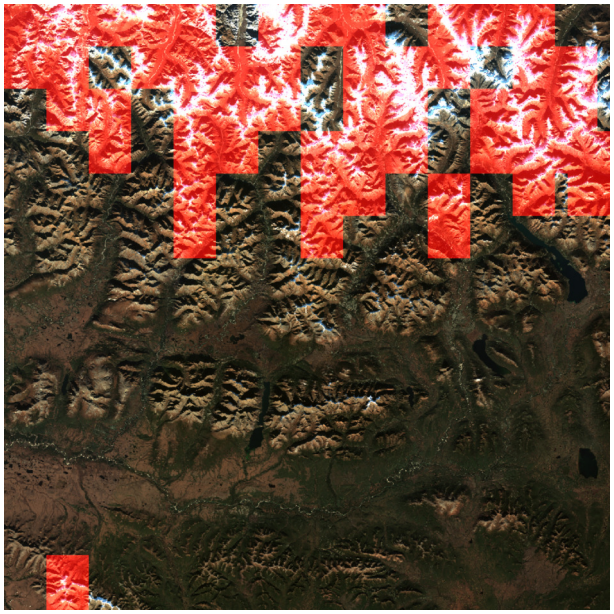


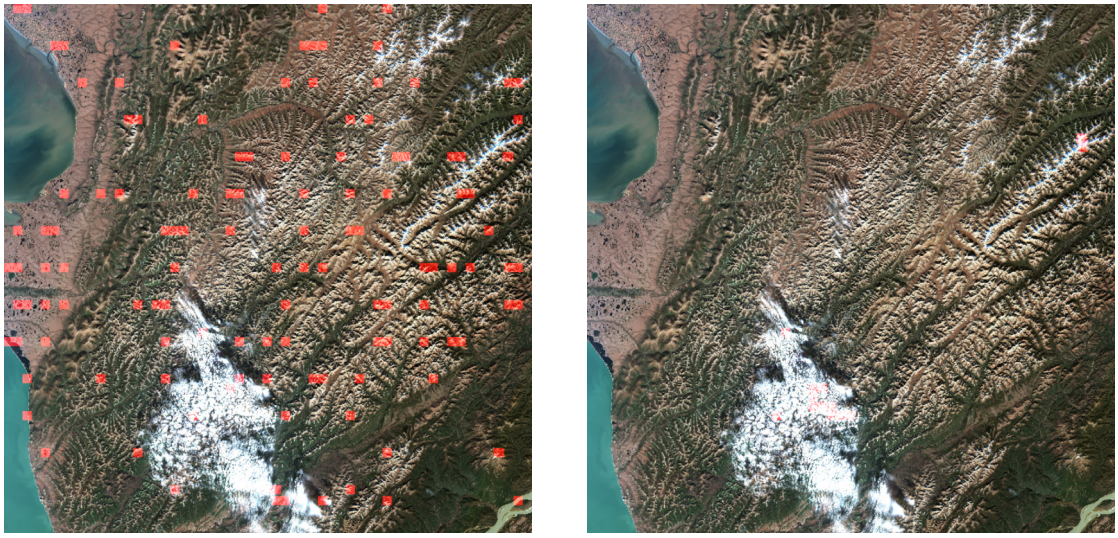
Figure 23: Anomalies found in original image_11 using features $f_1$

running only images 1-5 through the method.

The results of this second, partial clustering were mixed. Generally the idea was to remove the heavily anomalous images to produce more anomalies from the rest of the images. The effect however was opposite. The second partial clustering produced less anomalies in general. This was most heavily seen with the images using features $f_2$ and $f_4$ (images with these anomalies being more anomalous in general). The exact reasons for this was not clear. Considering that the single step in the method that is sensitive to input data is the clustering phase, it's likely, that removing the "noisy" data the hierarchical part of HDBSCAN was able to generate more compact cluster and/or more clusters in general. This in effect would result in a less of the data points having a high anomaly scores from the GLOSH algorithm. This, again is speculation, but would be one possible explanation for these effect.

While the original reason for reducing the dataset size to the first five images was not accomplished. The results were actually "better", though better being an subjective property. The images contained less anomalies, and generally the anomalies were located in visually better areas. This can be seen by comparing figures 24b and 24a. The former image being quite noisy, and in the latter anomalies detected only in cloud cover and mountains. Similar results were gathered for other images also: most anomalies were removed with partial clustering

leaving mainly cloud-cover and mountain anomalies.



(a) Full clustering

(b) Partial clustering

Figure 24: Full and partial clustering of original image_05 using features $f_2$

All in all these results were not gathered as a proof of the method, but to give some sense on the workings of the method. By visually inspecting the resulting masked images, first soft hints were given that the method could work. This was considered a success since the up to this point there have not been any evidence that the method would work outside the theoretical framework. The resulting images will also be of use later with the synthetic data as a comparison point.

## 4.2 Synthetic data

In the previous section some exploratory results were introduced, and in this section a more valid results are presented. Two runs were done based on the results presented in the previous section by using the synthetic data created in section 3.2. The resulting anomalies are also visualized, but unlike in the previous sector, RGB images are not used as the base for the visualizations. Since the images from which the synthetic data was created are same as used in the previous section, the RGB images are the same (excluding the synthetic anomalies). As such the base images for visualizing the results with synthetic data are the ground-truth masks generated in parallel with the synthetic data. One of these masks was depicted earlier

in figure 19.

For the synthetic data, validation can be run as the data is now labeled. Receiver Operating Characteristics (ROC)-curves were drawn and five different metrics were gathered:

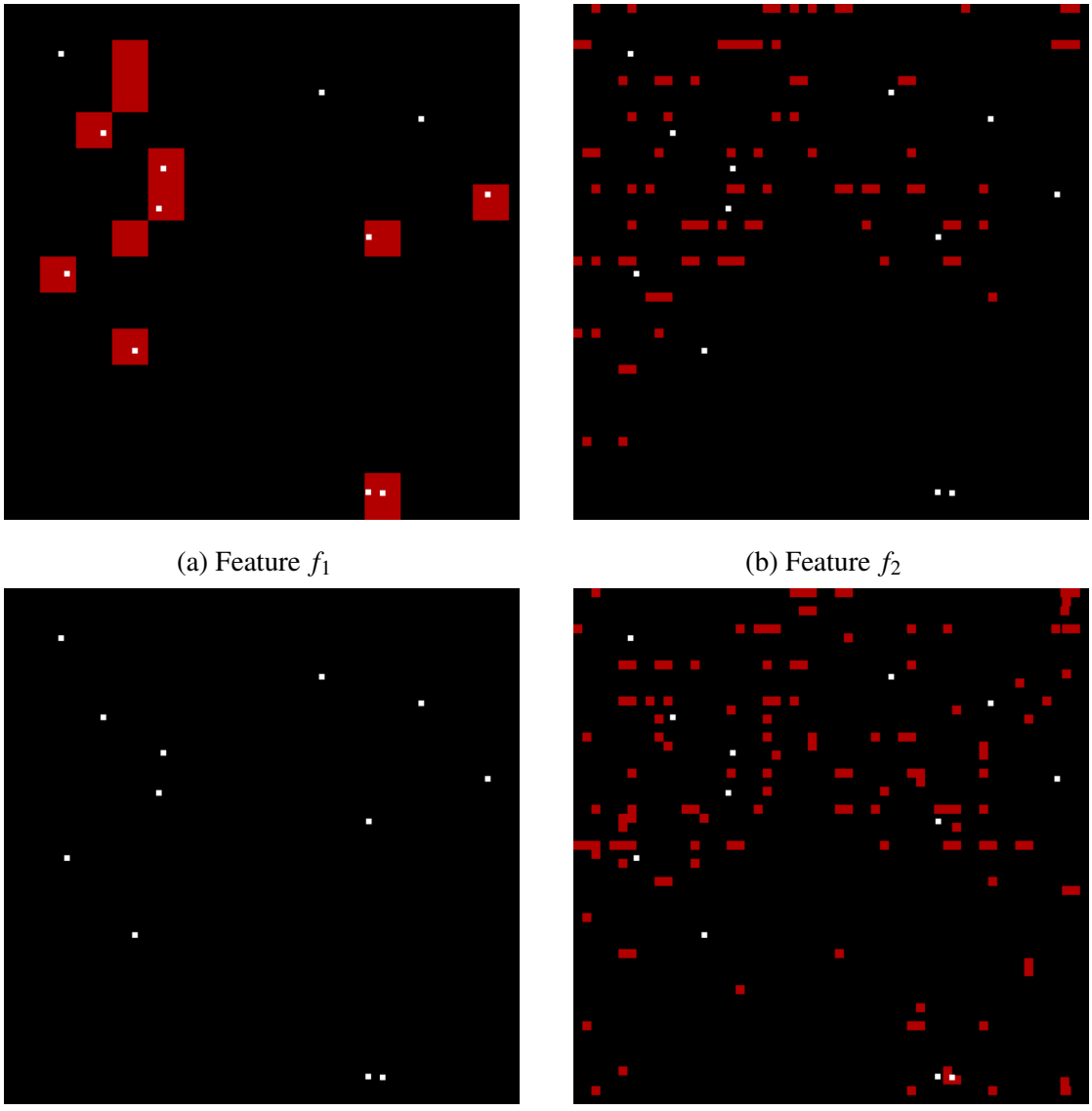
- TPR
- FNR
- $Precision = \frac{true\ positive}{true\ positive + false\ positive}$
- $F\text{-}score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$
- Area Under ROC-Curve (AUC)

Note that labels only contain synthetic anomalies; that is, all non-synthetic images are labeled as normal, event though the method labels them as anomalies. These anomalies are referred to as "natural" anomalies. Now this does affect some of the metrics. TPR- and FNR-values can be calculated accurately with respect to the synthetic labels, but precision cannot. With precision false positives cannot be calculated, or more precisely all positive results that were not synthetic anomalies are considered as false positives regardless if they are actual anomalies or not. To correct this would require a considerable amount of manual labor to go through all of these images, and label them. This manual labeling would also be subjective. This causes precision to be lower that it *might* be, and as an extension f-score.

TPR and FNR should be self-explanatory. F-score is a single-valued measure of the "goodness" of the detector, with higher the better. ROC-curves are graphs where on the x-axis is the FPR and on y-axis the TPR. ROC-curves are closely linked to the $t_{anom}$ parameter and answer the question: *"If I lower the threshold by this much, how much my true positive and false positive values change?"* In an essence they give the trade-off between better TPR and worse FPR. AUC is the area under the ROC-curve, 1 being maximum (= the detector has the ability to detect all anomalies with any FPR rate, especially 0).

The result gathering process for the synthetic data itself is identical to the unmodified data, with the single exception that visualization images are generated on top of the ground-truth masks. The first results collected were those for full clustering. Event though the clustering phase is sensitive to input data, the introduction of synthetic anomalies did not change the detection of natural anomalies. One of the questions before the results were collected, was

whether or not these synthetic anomalies will mask the natural ones. Luckily this was not the case (as seen by comparing figures 21a and 25a). This effect was observed for all of the used features.



(a) Feature $f_1$

(b) Feature $f_2$

(c) Feature $f_3$

(d) Feature $f_4$

Figure 25: Anomalies found in image_02 of synthetic data using full clustering

After the results from full clustering in previous section, similar results were expected with the synthetic data. Especially since the synthetic anomalies did not mask the natural ones, no drastic changes were excepted. Because of this, the results obtained from the full clustering were not excepted to very good. By visually inspecting the resulting images gained

two conclusion were drawn. Firstly, consistent with the earlier results, features $f_1$ and $f_3$ two produced a lot less noisy results. Feature $f_1$ worked decently in detecting the synthetic anomalies (figure 25a), while feature $f_3$ did not (figure 25c). Features $f_2$ and $f_4$ gave fairly noisy data, and did not perform well (figures 25c and 25d). Since synthetic anomalies are created by increasing the values of the anomalous areas, and because feature $f_1$ stores the largest values across each feature and band it is more likely that it would catch these anomalies. Interestingly, while features $f_2$ and $f_4$ also store max-values (across sections for each band and features) they did not catch these anomalies (as seen by comparing figures 25a and 25b). This might be because the full clustering masks these anomalies. The reason for the bad behavior of feature $f_3$ was not clear. It might be due to the reason how convolutional layers work. Since $f_3$ contains anomalies only from the second convolutional layer, the "simple" anomalies generated by increasing the values of the anomalous areas might not traverse to the more generalized deeper convolutional layer.

The performance metrics for the full clustering can be seen in table 3, and the ROC-curves in figure 26. From these values the TPR and FNR should be considered most, since they are the only values that can be accurately computed with only synthetic anomalies. Like predicted, these values are not very good. Especially the effects of the noise in features $f_2$ and $f_4$ can be seen clearly. Features $f_1$ and $f_3$ did fare a little better, but not still falling short. However, while visually inspecting the result, they did seem to work fairly well for the earlier images. This would hint that partial cluster would give better results, as with the unmodified images. The ROC-curves in figure 26 are also not very promising. But since the x-axis of the ROC-curve itself cannot be correctly computed, the whole figure needs to be taken with a grain of salt. The x-axis should not be considered as a FPR meter, but a general *natural anomaly rate*. This would greatly increase the usability of the ROC-curve for the purpose of this thesis: since the function of the method is to generally find these natural anomalies, a metric such as FPR does not really exist from this point of view. Though one could argue that TPR also loses meaning then; the anomalies are in fact synthetic. All of the ROC-curves except the one for feature $f_3$, tend to act in a similar manner: a slow steady rise on diagonal until about 0.2 FPR, and then a rise to 0.8 TPR with different steepness. This effect is most prominent with feature $f_1$. Comparing the TPR value for feature $f_1$ presented in table 3 with the corresponding ROC-curve, would place the FPR-value of $f_1$ in about 0.11. With

this in mind the performance of full clustering with respect to feature $f_1$ could be increased by lowering the $t_{anom}$ parameter until the TPR takes the sharp turn shown in figure 26 and raises to level 0.8. This would still present a FPR of less than 0.2. Same kind of increase of TPR cannot be gained for the rest of the features without significant rise FPR. This effect can also be seen in figure 27: by lowering the plane corresponding to the $t_{anom}$ value, more of the synthetic anomalies (marked on the floating plane as red) would be above it, and thus classified as anomalies.

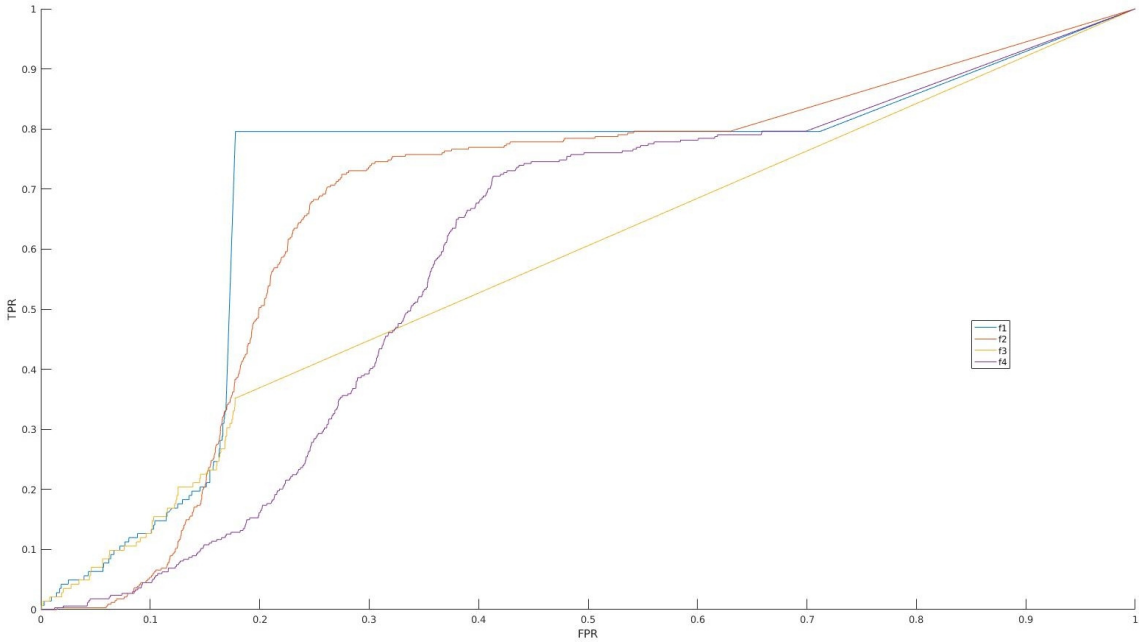



Figure 26: ROC-curves for full clustering of synthetic data

Table 3: Performance metrics for full clustering of synthetic data

| **Feature** | **TPR** | **FNR** | **Precision** | **F-score** | **AUC** |
|---|---|---|---|---|---|
| $f_1$ | 0.143 | 0.857 | 0.796 | 0.242 | 0.709 |
| $f_2$ | 0.001 | 0.999 | 0.003 | 0.001 | 0.579 |
| $f_3$ | 0.061 | 0.939 | 0.07 | 0.065 | 0.680 |
| $f_4$ | 0.001 | 0.999 | 0.006 | 0.002 | 0.594 |

Based on these somewhat disappointing results of the full clustering and the results from unmodified data, the second partial clustering of synthetic data was also done by using the

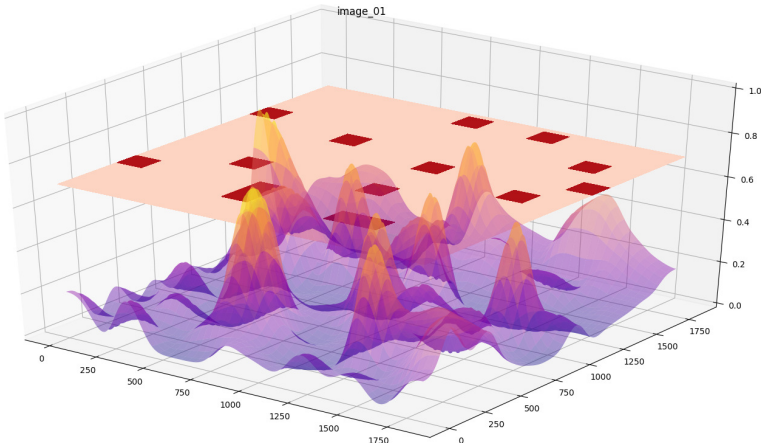


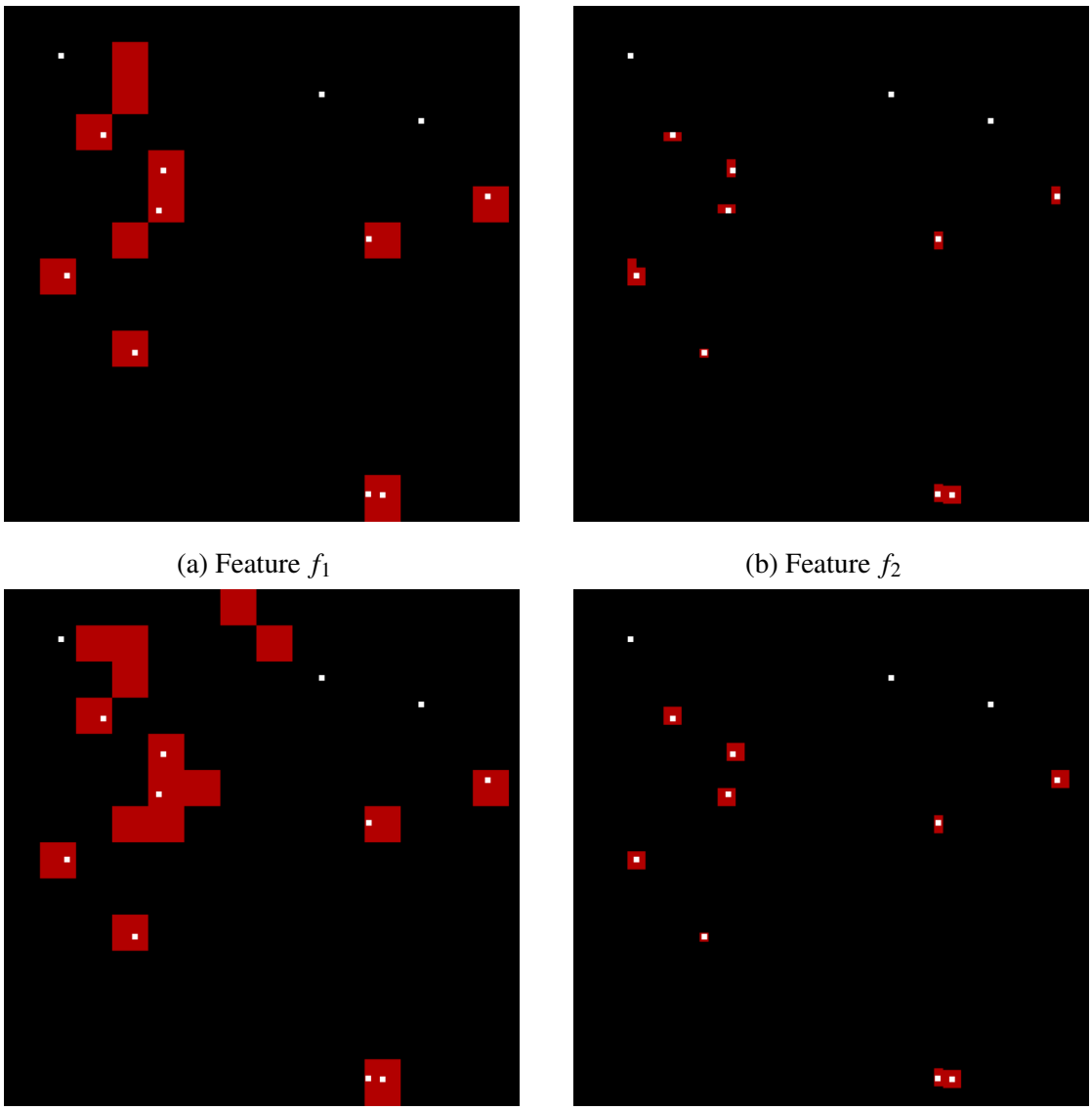

Figure 27: Anomaly scores of image_01, $f_1$ using full clustering

same images as with unmodified data. These results should be better if the same effect can be observed as with unmodified data.

Based on the visual inspection of the resulting mask from partial clustering of features $f_1$ and $f_2$, similar conclusion were drawn as with the unmodified data. Anomalies generated by features $f_1$ were mostly the same, though full clustering did found a few more natural anomalies (as with the unmodified data) but otherwise identical. Feature $f_2$ did likewise perform as excepted: the data contained a lot less anomalies with partial clustering and the synthetic anomalies were detected a lot more likely. This can bee seen from figures 25b and 28b. One change with respect to the partial clustering of unmodified images was with feature $f_3$. With synthetic data, $f_3$ behaved opposite to how it behaved with the unmodified data. With unmodified data, no great change was observed with $f_3$, but with the synthetic data $f_3$ worked a lot better with partial clustering. This can been seen by comparing figures 25c and 28c. Feature $f_4$ likewise performed differently from unmodified data: it found significantly less anomalies with partial clustering than with full clustering. Like with feature $f_4$ it also performed better with the detection of synthetic anomalies, as seen by comparing figures 25d and 28d.

So mostly partial clustering worked as excepted: the already decent feature $f_1$ stayed more or less the same, and the rest of the features preformed better. This was all predicted behavior, but the fact that features $f_3$ and $f_4$ performed a lot better with partial clustering was not. This is especially interesting since both of these features work only on the second convolutional

layer, and the reason for the bad behavior of these with full clustering was postulated to be that the synthetic anomalies have little to no effect on the second convolutive layer. These results however seem to invalidate this reasoning. These results would indicate, that images 6-13 seem to have some features on the second convolutional layer that masked the existence of the synthetic anomalies. This could be plausible since these images, in general, do contain a lot of anomalies, especially in the mountainous areas.



(a) Feature $f_1$

(b) Feature $f_2$

(c) Feature $f_3$

(d) Feature $f_4$

Figure 28: Anomalies found in image_02 of synthetic data using partial clustering

As with the full clustering, ROC-curves were drawn and performance metrics were gathered.

The ROC-curves can be seen in figure 29, and the metrics in table 4. These are in line with the results observed through visually scanning the the resulting masks. By analyzing the ROC-curves two conclusions can be drawn. Firstly by noticing that the ROC-curve for each of the features very sharply, almost vertically, to TPR level of 0.7, one can infer that the partial clustering is able detect a fair amount ( 70% to be exact) of the anomalies with few to none natural anomalies. Also by comparing the TPR values of features $f_1$ and $f_3$ to the corresponding ROC-curves, the performance of these two features could be raised to TPR-level of 0.7 without any meaningful increase of FPR. This can also be seen in figure 30. From the same image one can also observe, that some synthetic anomalies did not produce any increase in $s_{anom}$ (higher left-hand anomaly shown in figure 30). The second conclusion drawn from the ROC-curves is the shape of each of the curves. Meaning that even by lowering $t_{anom}$ parameter, no gains cannot be gained over the TPR-value of 0.7 without a drastic rise in FPR to $0.6 - 0.7$. This is interesting since with full clustering the TPR-value could be raised to a higher level (up to 0.8) with lower FPR-values. This effect is not true for feature $f_3$ which performed quite badly in full clustering. This makes the partial clustering generally better than full, but if one is ready to accept higher levels of FPR-values say $0,3$ then full clustering might perform better. This is especially true when using feature $f_1$.

All in all the method did produce decent results for the synthetic data. As predicted from the results of the unmodified data, the partial clustering did perform better, but if on is willing to accept a higher FPR value, full clustering could also work.

Table 4: Performance metrics for partial clustering of synthetic data

| **Feature** | **TPR** | **FNR** | **Precision** | **F-score** | **AUC** |
|---|---|---|---|---|---|
| $f_1$ | 0.307 | 0.693 | 0.722 | 0.431 | 0.766 |
| $f_2$ | 0.632 | 0.368 | 0.705 | 0.431 | 0.757 |
| $f_3$ | 0.275 | 0.725 | 0.722 | 0.431 | 0.755 |
| $f_4$ | 0.89 | 0.11 | 0.698 | 0.431 | 0.764 |

In this chapter we presented the results gathered from the method proposed in chapter 3. This was begun by presenting the results from two different sets of the unmodified images. One for all the images, and one for a subset. These first results aimed to provide some basic

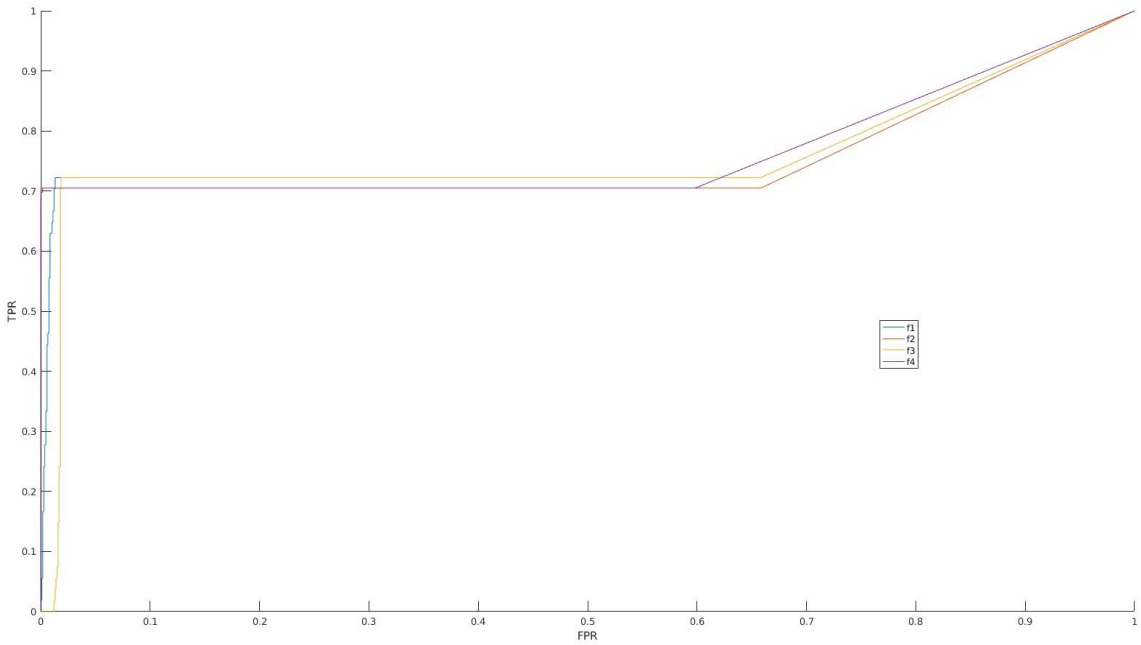



Figure 29: ROC-curves for partial clustering of synthetic data

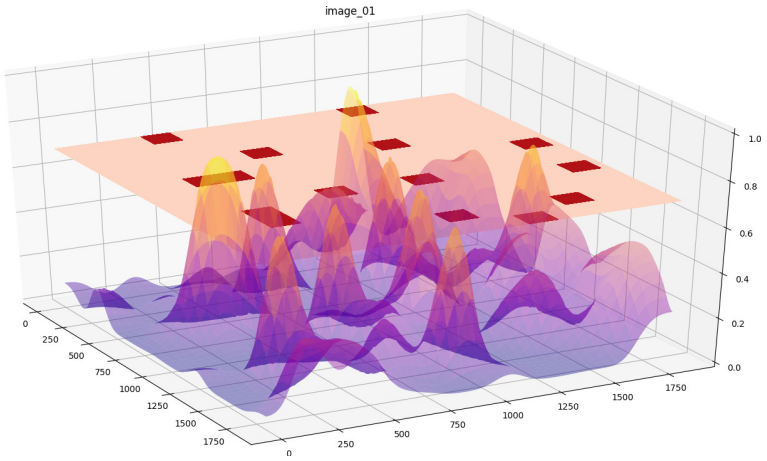



Figure 30: Anomaly scores of image_01, $f_1$ using partial clustering

knowledge of the performance of the method before moving to validation phase. They also simulate a possible real-world use-case of the method and provide some comparison point for the later runs. In the second part of this chapter we presented the similar results for the synthetic data. These now included validation metrics.

# 5 Discussion

The goal was to present a method capable to detect anomalies from large sets of HSI data. This was achieved by leveraging the innate property of autoencoder, one is able to learn the distribution of the encoded data. This was cornerstone upon which the whole method was built on. By combining auto encoding networks and convolution, the resulting network can now learn not only spectral distribution, but also spatial. This is an important point of the method. Detecting spectral anomalies can be done fairly easily by using existing anomaly detection algorithms, such as RX or even by simple statistical analysis. However, the detection of spatial anomalies is a lot more complex affair. So much so, that before the wide-spread application of CNNs, it was not feasible. The adaptation of CNNs making it possible to learn meaningful spatial features without the need to manually define them. By expanding this feature from two to three dimensions, the proposed method is now able to learn both spatial and spectral features at the same time. This being one of the most important features of the method. While CNNs have been used to learn features from images, even in unsupervised manner by using CAEs (Du et al. 2017; Masci et al. 2011), no studies were shown where CAEs were used for anomaly detection specifically. Masci et al. 2011 and Du et al. 2017 used their implementation to initialize CAEs for classification purposes. One study however was found, where convolutional autoencoders were used to detect anomalies from videos, i.e. spatio-temporal anomalies (Chong and Tay 2017). This lack of previous studies would indicate the proposed method does fill a hole in the field of anomaly detection.

The data used in this thesis was hyperspectral. This was due to the fact that I was working with hyperspectral images at the time of the writing, but also: HSI data gives the ability to learn spectral features on top of spatial ones. While the idea was to use a massive dataset for this method ,13 images is not that massive. Even by splitting them to smaller images, the nearly 3000 images generated while adequate, is not a very large dataset. However, like mentioned in section 3.1, there are no readily available hyperspectral dataset that would have filled all the criteria for this thesis. Luckily ESA does provide the Sentinel 2 data freely, but the collection, and transformation to usable format did require a fair amount of manual labor. This caused the dataset used in this thesis to be on the smaller side. The images

themselves should have been selected more carefully also. This would have been apparent by more carefully inspecting the images. The latter halve of the dataset did contain images not exactly suitable for experimentation. Because of this, the method was run with smaller dataset also: the so-called partial clustering. This could have been prevented also by simply using more data. The choice of geographical area for the data could have also be though in more detail. There was no particular reason why Alaska was chosen. The Copernicus-hub probably gave it as one of he first areas when searching for usable data. However the data in there is quite heterogeneous. Yes, this was one of the original criteria for the data, but after viewing the results a little more homogeneous data would have worked better. Saudi-Arabia, or any desert environment was a brief consideration, though this would have been a double-edged sword. While the results would have most likely been better, the reliability of the results would have been suspicious (the detection of anomalies would have been *too* easy).

While hyperspectral data was used, it does not have to be. By reducing the size of the spectral dimension of the convolutional kernels, this method should work for any spectral dimension. Reducing the size of this dimension to one, will essentially produce a 2-dimensional convolutive network. That is, a network capable of learning spatial features only. Either for one grayscale layer, or for multiple layers (e.g HSI). This makes the proposed method extremely versatile. Also the current structure and training method for network makes it learn global features, but with small modification, it should be able to learn local also. This can be achieved by either training each network for single images (time consuming and computationally expensive), or by making the network two-part. One part that is pre-trained to learn global features, and one that is trained for each image individually. So instead of training the whole network again for each image, say the last few layers are retrained. This idea was something that was thought at the earlier phases of this thesis, but to keep the scope manageable it was discarded. Other ideas were also thought of. For example: more complicated structures for the network. Such as more layers or different ones, like the ones presented by Du et al. 2017. These were also discarded mainly due to practical reasons, and partly due to the data. The number of bands restricted the number of possible pooling layers. Though in hindsight, the pooling operation could have been done across spatial dimensions only, thus removing this restriction. Still the point of this thesis was to provide proof-of-concept of

the method. It was deemed, that further studies of the different networks would have made the process unnecessary complicated. Taking in mind that the depth and type of layers are not the only changeable parameters in the network: size and number of kernels, pooling, activation function etc. making the number of possible variations for the network countless. However, this does open a possible road for future research. Especially using deep networks might provide better results. Maybe not with hyperspectral images, but I would be interested to study the application of this method to the detection of spatial anomalies.

The second phase of the method is also suspect to countless variations. How the features are extracted, how many features to use, what dimensionality reduction techniques should/could be used etc. At the beginning of the result gathering process, only two features were meant to be used: $f_1$ and $f_2$. However, since this neural network used in this method could be characterized as deep network, (though whether or not two layers is deep is subject to debate) it was thought that using this "deep representation" (i.e. features $f_3$ and $f_4$ from the second convolutional layer) was required to full understand the potential of the method. With more convolutional layers these would have been more meaningful, but considering the results from these features particularly from the partial clustering, these still contained useful information. The importance of these deep features would probably increase when searching for spatial anomalies. With more layers more complex features (i.e. forms, as was shown in figure 12) can be learned, and more complex anomalies detected. When searching for spectral anomalies, one convolutional layer could be enough, but when searching for say a building in wilderness this kind of anomaly would probably not be present in the lower-level feature-maps. Considering that the used dataset was from Alaskan wilderness, this would likely have been a wasted effort in this thesis, though there is one city in the dataset. In figure 32c, one can see an airport runways in the middle of the image (at the middle of the image, intersection of three rivers). It would have been interesting to see if the method could have detected this area as anomalous. Although resolution of the image quite poor for this. While the four features used in this thesis were still fairly simple; one low dimension and one high dimensional both simply storing the largest values. These features are also grandfathered by the original idea. At the very beginning when there was only an inkling of an idea, I thought of a simple way to detect anomalies by scaling all the feature-maps to same size, and layering them on top of another and then storing the largest value. In this case a

small value would indicate an area where there are no commonly found features. While this form of anomaly detection was later refined to the currently used, the idea of max-valued features still survived. Another ways to extract were also thought of, but were dropped from the scope of this thesis. Some were simply differently chosen values, such as means or minimums instead of maximums, but some of these would have required the existence of labels. With labels, one could have chosen meaningful features based on them. This however would change the method from unsupervised to supervised. Since when selecting the used features this way, they tend to represent the anomalies present in the labeled data. In this case novelty detection (anomalies not present in the training data) would suffer. But if this is what is required, the method could certainly be changed for supervised one. Although in this case a simple CNN with fully-connected classification layer might be all that is needed, (making it essentially a classification problem). Any number of the already proposed methods would be suitable for this, like the one in Li, Zhang, and Shen 2017. However, by using CAE, one would still retain the option for novelty detection. While the methods requiring labeled data were overruled, there was one method that might have been able to choose more meaningful features. The RELIEF method proposed by Kira and Rendell 1992. This was ruled out because of two reason. Firstly the method is computationally heavy, and works by dropping out a random set of features and seeing if the results are better. This would have significantly increased the time required to run the method. The second reason has again to do with labels. RELIEF does need some metric on how the method performs, and in the case of this thesis they are based on the synthetic labels. This would have caused RELIEF to choose features representing the synthetic anomalies. Not exactly hoped for behavior. But taking the idea behind RELIEF a working feature extraction/refining algorithm might have been devised. For example one removing redundant features from the data. Somewhat similar to PCA, but automatically choosing the dimensions for the projected data. Another inkling of an idea I had was to use the features extracted in phase 2 to segment the dataset into similar parts. Since the first part should extract meaningful features, similar areas should have some commonalities in these. This could be leveraged to segment the dataset to different areas, and retraining a new network for each of the areas. Then when actual data is driven through the method, first it's classified to one of these segments, and then the segment-specific network is used to extract features. This way the anomalies detected in an image, would not be that

of the whole data, but of similar areas to the image. This should reduce the FPR value. This was only an idea and not explored further. It might improve the performance of the method with the cost of computation time and resources.

As with the first and second phase of the method, the internal structure an algorithms of the third phase is also the subject to choice. In this thesis the used HDBSCAN method was chosen for ease-of-use and the ability to work without extensive parameter optimization. Different algorithm were thought of, but ruled out. One of the ideas I had during the early process was to modify a previously used method to use multiple different algorithms for anomaly detection. This worked by grading each algorithm based on their accuracy, and using this factor as a weight for the prediction each of the algorithms would give. However, since the method already had a lot of moving parts, this was rejected (as it would have introduced more of them). However it does give an another road for possible future research. While no complex multi-algorithm was chosen for this thesis, the used HDBSCAN might have still benefited from some optimization for the $p_{min}$. The chosen value of 100 was determined very early in the research process, and was not revisited. HDBSCAN algorithm is quite care-free, and the phase three of the method is the simplest in the configuration of methods presented in this thesis. Still it caused no small amount of problems. The largest of which being the validation. Like already mentioned multiple times, the data does not contain labels, and as such no validation can be done. This, of course, is not true for the synthetic data. Because of this the validation results presented in chapter 4 are not completely reliable. They are correct, however one needs to understand the limitations caused by the lack of labels for the natural part of the image, and interpret these results with this in mind. This is a common problem in unsupervised learning, and cannot be resolved easily. One way would be to manually label the data. This being immensely laborious and subject to the labelers interpretation of an anomaly. During the validation process, it was thought to use the natural anomalies found from the unmodified data as labels for the synthetic data. This would however have some quite serious limitations. Firstly the labels would not be absolute, but would depend on the parameters for the method. The second more serious flaw would be the resulting circular thought: the method itself would be used to validate itself. What if the method would not have worked? In this case the labels used for validation might have been wrong, the worst case scenario being that the validation results would have

proved that the method works while it actually does not. It was also debated whether or not a comparison of the methods performance to some other algorithm, such as RX, should be done. While this would have been interesting to see, it would not have been without problems. The main problem being the different premise on which the proposed method and RX works. The purpose of RX is to find spectral anomalies from local neighborhoods, while the proposed method finds both spectral and spatial anomalies from global normal model (i.e. "global" neighborhood). While some conclusions could have been done by comparing the performance of these two, it was thought that in general they are not comparable. The possibility of this kind of comparison is included in the future research, and at some point, when the method is refined to its final form this could be conducted. However, in the view of this thesis it was omitted as to not present more results not based on solid ground. While no comparable analysis was done, and the used validation method is flawed, at least the flaws are known and can be taken into account when analyzing the results.

All in all, while the validation results cannot be conclusively proven, I feel confident that the original purpose of this thesis was fulfilled. That is, to provide a proof-of-concept that the method would have at least the possibility of working. The validation results are for the described configuration of the method and might not be generalization to all, or any configurations. However there is no proof that the method would not work in different configuration. With this in mind, and considering that each of the three phases are more or less independent of each other, there is a plethora of possible roads for future research. Following the flavor of the month, I'm especially interested in the possibilities provided in deep learning, and as it pertains to the performance of the first phase of this method.

# 6 Conclusions

In this thesis a novel unsupervised anomaly detection method for the detection of spectral and spatial anomalies from hyperspectral data was proposed. A proof-of-concept implementation was also suggested. This implementation uses a two-layer convolutional autoencoder to learn a total of 96 commonly occurring features from the data. Feature-maps generated by this network is then distilled into 4 feature-vectors. Lastly using HDBSCAN/GLOSH algorithm, these vectors are used to detect anomalies. Any of the three phases of the method can be modified to suit the problem in hand. While convolutional neural networks and even convolutional autoencoders have been studied, no methods for detecting anomalies in this manner has been proposed previously. The detection of spatial anomalies in general is a difficult task, and thus this method does seem to fill a void. In this thesis the method was proposed with hyperspectral data in mind, but it does work out-of-the-box for any data containing multiple spectral dimensions (such as RGB images). With small modifications it can also be applied for images with only one spectral dimension.

While the idea was to use hyperspectral images, no available datasets of the required size and type were found, and method was tested with multispectral one. This data was gathered by ESAs Sentinel 2 -satellite, and collected from ESAs Copernicus-hub. The dataset is a collection of 13 geographically adjacent multispectral images from central Alaska (each sized $100km \times 100km$). These full-sized images where then split into 225 smaller images to form the final dataset of 2925 images. Two runs were conducted for the detection of anomalies. First with the full dataset, and a second one with a partial dataset. This was because of suspicion, that some of the images had a masking effect.

A second test was conducted for validation purposes with data containing programmatically generated synthetic anomalies. Since the location of each of the synthetic anomaly is known, performance metrics were calculated. As with the unmodified dataset, two runs were conducted. One for full dataset and one for partial dataset. While the reliability of these results is not great, they do provide reasonable proof to support the idea behind the method. That is, convolutional autoencoders can learn a normal model for hyperspectral dataset, and by using this model anomalies or outlier can be detected. In order to get more valid results, the

method should be tested with fully labeled data. However, this is not as straight-forward as it seems. Firstly the definition of an anomaly in an image is subject to interpretation. Secondly, suitable fully labeled hyperspectral datasets do not exist, and manually labeling a dataset, as small as the one used in this thesis, requires a massive amount of work.

In the view the original research questions: *"Can CAEs or SCAEs be used to detect anomalies from hyperspectral data?"*, and the more general one: *"Wow can one detect any anomalies, spectral or spatial, from any kind of hyperspectral dataset without any prior knowledge of the said dataset?"*, this thesis can be considered an success. At the beginning there was no indication if the original idea might work, and the goal was to execute an exploratory study of whether or not it would. The first question is answered by the results introduced in chapter 4 indicating a success. The second research question answered in chapter 3 by proposing a method for unsupervised detection of spatio-spectral anomalies. With these two question both answered, the possibility to move forward from exploration research to actual empirical one is opened, to study and refine the three different phases of the proposed method.

# Bibliography


Adams, Jim, Ken Parulski, and Kevin Spaulding. 1998. "Color processing in digital cameras". *IEEE micro* 18 (6): 20–30.

Banerjee, A., P. Burlina, and C. Diehl. 2006. "A support vector method for anomaly detection in hyperspectral imagery". *IEEE Transactions on Geoscience and Remote Sensing* 44 (8): 2282–2291. doi:10.1109/TGRS.2006.873019. http://ieeexplore.ieee.org/document/1661816.

Campello, Ricardo J G B, Davoud Moulavi, Arthur Zimek, and Jörg Sander. 2015. "Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection". *ACM Trans. Knowl. Discov. Data* 10, number 1 (): 5:1–5:51. doi:10.1145/2733381. http://doi.acm.org/10.1145/2733381.

Chandola, Varun, Arindam Banerjee, and Vipin Kumar. 2009. "Anomaly Detection: A Survey". *ACM Comput. Surv.* 41, number 3 (): 15:1–15:58. doi:10.1145/1541880.1541882.

Chang, Chein-I, and Shao-Shan Chiang. 2002. "Anomaly detection and classification for hyperspectral imagery". *IEEE Transactions on Geoscience and Remote Sensing* 40, number 6 (): 1314–1325.

Chen, Yushi, Zhouhan Lin, Xing Zhao, Gang Wang, and Yanfeng Gu. 2014. "Deep learning-based classification of hyperspectral data". *IEEE Journal of Selected topics in applied earth observations and remote sensing* 7 (6): 2094–2107.

Chong, Yong Shean, and Yong Haur Tay. 2017. "Abnormal Event Detection in Videos using Spatiotemporal Autoencoder". *CoRR* abs/1701.01546. http://arxiv.org/abs/1701.01546.

Cun, Y. Le, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, and W. Hubbard. 1989. "Handwritten digit recognition: applications of neural network chips and automatic learning". *IEEE Communications Magazine* 27, number 11 (): 41–46. ISSN: 0163-6804. doi:10.1109/35.41400.

Du, Bo, Wei Xiong, Jia Wu, Lefei Zhang, Liangpei Zhang, and Dacheng Tao. 2017. "Stacked convolutional denoising auto-encoders for feature representation". *IEEE transactions on cybernetics* 47 (4): 1017–1027.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. "A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining,* 226–231. KDD'96. Portland, Oregon: AAAI Press. http://dl.acm.org/citation.cfm?id=3001460.3001507.

European Space Agency (ESA). 2017. "Sentinel 2 mission website". https://sentinel.esa.int/web/sentinel/missions/sentinel-2.

Goetz, Alexander. 2009. "Three decades of hyperspectral remote sensing of the Earth: A personal view". *Remote Sensing of Environment* 113:S5–S16.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* MIT Press.

Haykin, Simon. 1998. *Neural Networks: A Comprehensive Foundation.* 2nd. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0132733501.

Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. 2006. "A Fast Learning Algorithm for Deep Belief Nets". *Neural Comput.* (Cambridge, MA, USA) 18, number 7 (): 1527–1554. ISSN: 0899-7667. doi:10.1162/neco.2006.18.7.1527. http://dx.doi.org/10.1162/neco.2006.18.7.1527.

Kira, Kenji, and Larry A. Rendell. 1992. "The Feature Selection Problem: Traditional Methods and a New Algorithm". In *Proceedings of the Tenth National Conference on Artificial Intelligence,* 129–134. AAAI'92. San Jose, California: AAAI Press. ISBN: 0-262-51063-4. http://dl.acm.org/citation.cfm?id=1867135.1867155.

Lee, Honglak, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. 2009. "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations". In *Proceedings of the 26th Annual International Conference on Machine Learning,* 609–616. ICML '09. Montreal, Quebec, Canada: ACM. ISBN: 978-1-60558-516-1. doi:10.1145/1553374.1553453. http://doi.acm.org/10.1145/1553374.1553453.

Li, Ying, Haokui Zhang, and Qiang Shen. 2017. ”Spectral–Spatial Classification of Hyperspectral Imagery with 3D Convolutional Neural Network”, 9 (): 67.

Masci, Jonathan, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. 2011. ”Stacked convolutional auto-encoders for hierarchical feature extraction”. *Artificial Neural Networks and Machine Learning–ICANN 2011:* 52–59.

McInnes, Leland, John Healy, and Steve Astels. 2017. ”hdbscan: Hierarchical density based clustering”. *Journal of Open Source Software* 2 (11): 205. doi:10.21105/joss.00205. http://joss.theoj.org/papers/10.21105/joss.00205.

Reed, I. S., and Yu Xiaoli. 1990. ”Adaptive multiple-band CFAR detection of an optical pattern with unknown spectral distribution”. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 38, number 10 (): 1760–1770.

Smiti, A., and Z. Elouedi. 2012. ”DBSCAN-GM: An improved clustering method based on Gaussian Means and DBSCAN techniques”, 573–578. IEEE. doi:10.1109/INES.2012.6249802.

Stein, D. W. J., S. G. Beaven, L. E. Hoff, E. M. Winter, A. P. Schaum, and A. D. Stocker. 2002. ”Anomaly detection from hyperspectral imagery”. *IEEE Signal Processing Magazine* 19, number 1 (): 58–69.

Wikimedia Commons. 2007a. ”DBSCAN-Illustration”. Visited on October 24, 2017. https://commons.wikimedia.org/wiki/File:DBSCAN-Illustration.svg.

———. 2007b. ”EM spectrum”. Visited on October 24, 2017. https://commons.wikimedia.org/wiki/File:EM_spectrum.svg.

———. 2007c. ”HyperspectralCube”. Visited on October 24, 2017. https://commons.wikimedia.org/wiki/File:HyperspectralCube.jpg.

———. 2009. ”Cone-fundamentals-with-srgb-spectrum”. Visited on October 24, 2017. https://commons.wikimedia.org/wiki/File:Cone-fundamentals-with-srgb-spectrum.svg.

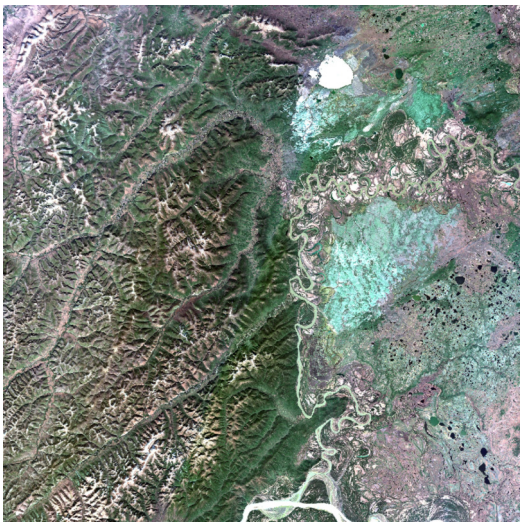
Yu, Fisher, and Vladlen Koltun. 2015. ”Multi-Scale Context Aggregation by Dilated Convolutions”. *CoRR* abs/1511.07122. http://arxiv.org/abs/1511.07122.

Zeiler, M. D., G. W. Taylor, and R. Fergus. 2011. ”Adaptive deconvolutional networks for mid and high level feature learning”. In *2011 International Conference on Computer Vision,* 2018–2025.

# Appendices

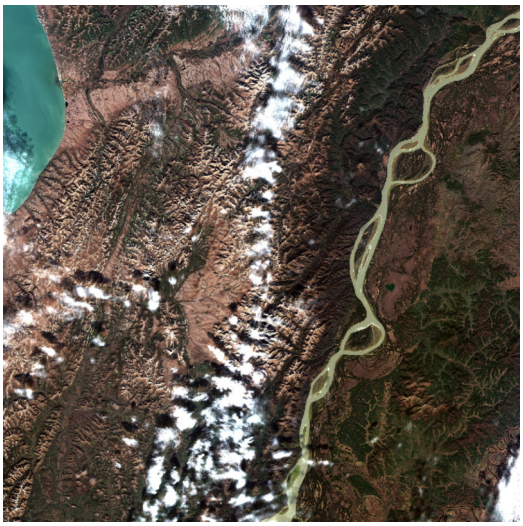
# A ESA raw data RGB images

This appendix contains the RGB projection for each of the used hyperspectral images. These images are are generated directly by the SNAP-application. The projection is done by choosing the red values from band 4 ( 665$nm$), green values from band 3 ( 560$nm$) and blue values from band 2 ( 490$nm$).

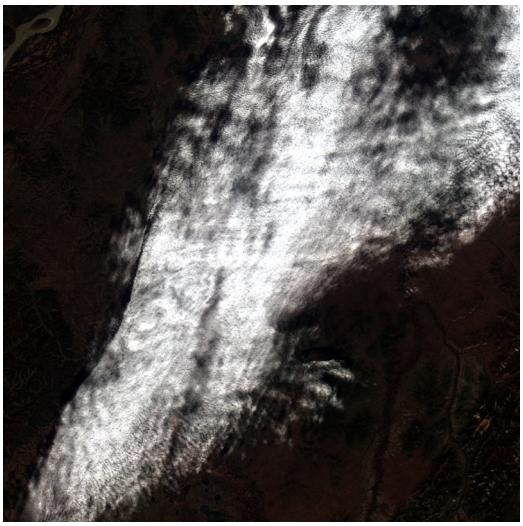


(a) image_01.jpg

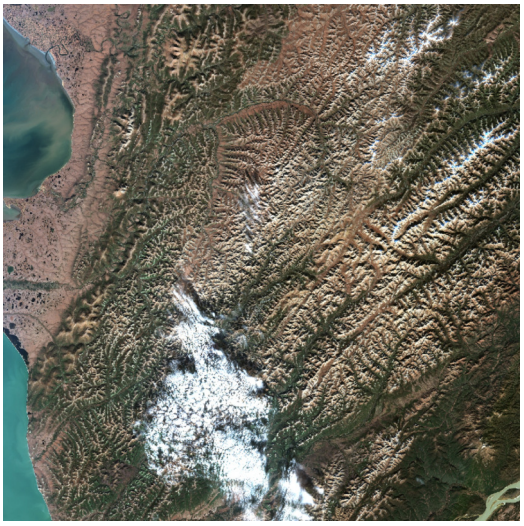
(b) image_01.jpg



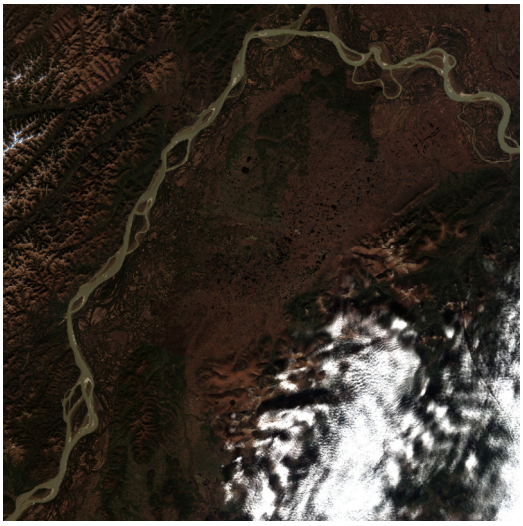
(c) image_03.jpg



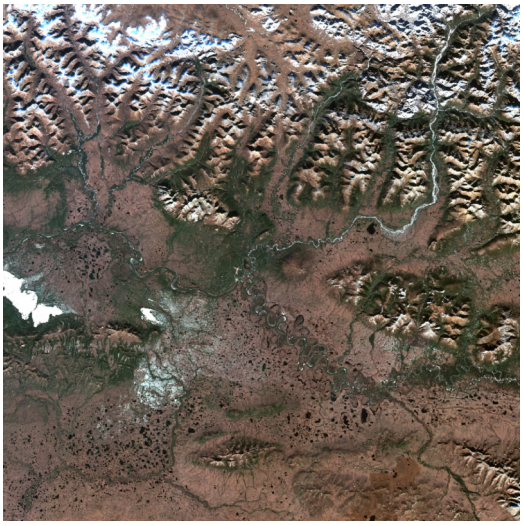
(d) image_04.jpg



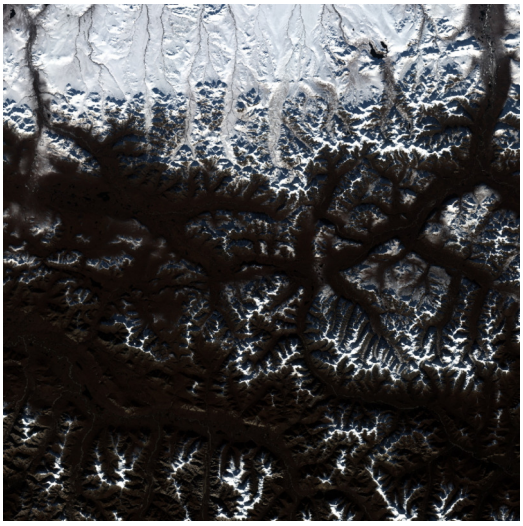
(a) image_05.jpg



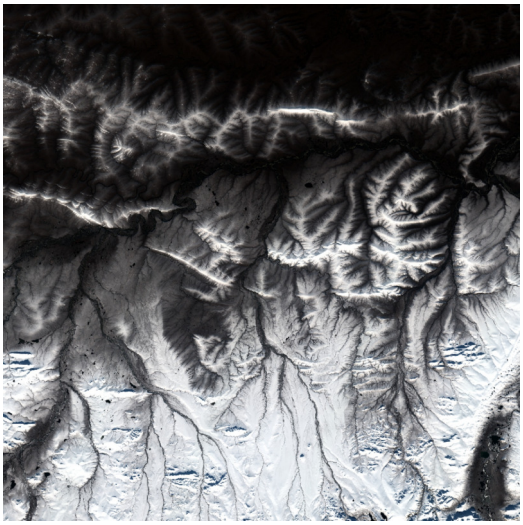
(b) image_06.jpg



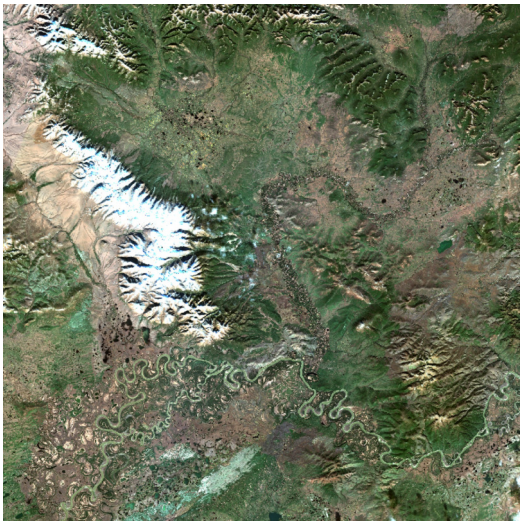
(c) image_07.jpg



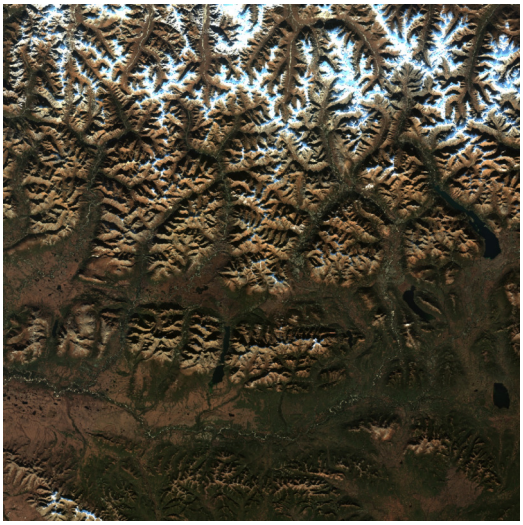
(d) image_08.jpg



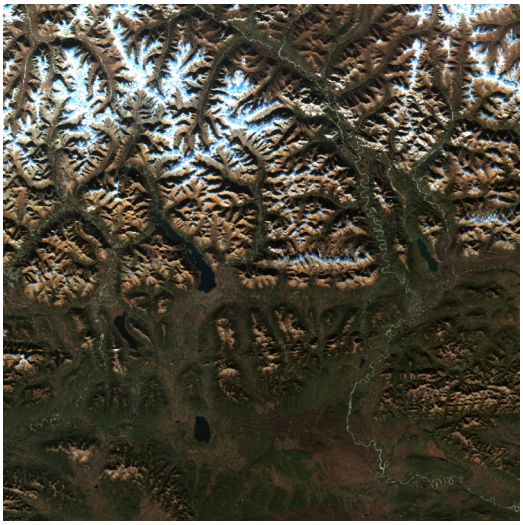
(a) image_09.jpg



(b) image_10.jpg



(c) image_11.jpg



(d) image_12.jpg

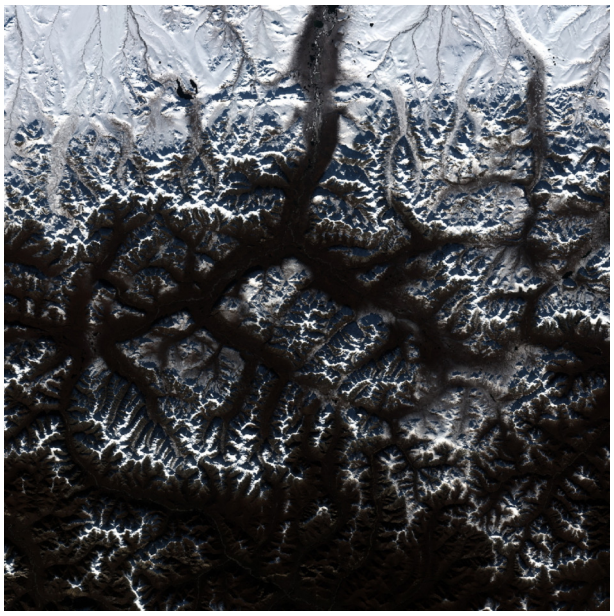


Figure 34: image_13.jpg