NTNU
Kunnskap for en bedre verden

Department of Electronic Systems

TFE4580 - Electronic Systems Design and Innovation,
Specialization Project

Autoencoder based anomaly detection in hyperspectral images

Author:
Aksel Lindbæk Gundersen

Supervisor:
Milica Orlandić

December, 2020

Table of Contents

List of Figures ii

List of Tables ii

1 Introduction 1

1.1 The HYPSO project . . . 1

1.2 Hyperspectral imagery . . . 1

1.3 Anomaly detection . . . 2

1.4 Outline of report . . . 3

2 Background 3

2.1 Anomaly detection . . . 3

2.2 Anomaly detection in HSI . . . 3

2.2.1 The Reed-Xiaoli algorithm . . . 3

2.2.2 Collaborative representation based . . . 4

2.2.3 Machine learning . . . 5

3 Autoencoders 5

3.1 Introduction . . . 5

3.2 Anomaly detection . . . 7

3.3 Deep Belief network . . . 7

3.3.1 Restricted Boltzmann Machines . . . 7

3.3.2 Contrastive divergence . . . 9

3.4 Deep Neural Network Matlab Toolbox by Masayuki Tanaka . . . 10

3.4.1 Initialization of the DBN . . . 11

3.4.2 Pre-training . . . 11

3.4.3 Supervised training with back-propagation . . . 12

3.5 HSI classification Matlab toolbox by Kang Xudong . . . 13

3.5.1 Previous use in anomaly detection . . . 13

4 Implementation in Matlab 13

4.1 Preprocessing of hyperspectral images . . . 14

4.2 Parameter selection . . . 14

4.3 Initialization of DBN . . . 15

4.4 Pre-training . . . 15

4.5 Fine-tuning algorithm . . . 16

4.6 Finding the reconstruction error . . . 16
4.7 Adaptive weights strategy . . . 16

5 Results **16**
5.1 Result metrics . . . 16
5.2 Description of test data . . . 16
5.3 Compared with Ma et al. [2018] . . . 16
5.4 Compared with Kang et al. [2017] . . . 16

6 Discussion **16**
6.1 Alternative training algorithms . . . 16
6.1.1 adaptive moment estimation optimizer (Adam) . . . 16
6.1.2 Sparse Encoding Symmetric Machine (SESM) . . . 16
6.2 Different variants of loss-function in fine-tuning . . . 16
6.3 Optimizing parameter values . . . 16
6.4 Edge-filters on anomaly map . . . 16

Bibliography **17**

Appendix **19**
A Hello World Example . . . 19

List of Figures

1 Gangwar [2017] . . . 1
2 Illustration of HSI for remote sensing from Bioucas-Dias et al. [2012] . . . 2
3 Illustration of the concept of autoencoders. Taken from Badr. . . . 5
4 General structure of a autoencoder. Taken from Jeong et al. [2020] . . . 6
5 General structure of a artificial neuron. Taken from Roos [2019] . . . 6
6 Illustration of a single RBM. Taken from Ma et al. [2017] . . . 8
7 Taken from Hinton et al. [2006] . . . 9
8 **?** . . . 9
9 Illustration of the encode-decode process of a HSI with a DBN AE. Taken from Ma et al. [2018]. . . . 13

List of Tables

1 Introduction

1.1 The HYPSO project

NTNU Small Satellite Lab is an initiative at Norwegian University of Science and Technology started to increase the amount of space-related activities. It is a ambitious project consisting of Master students, PhD-students and professors all working together on various projects. One of the projects that are currently in operation at NTNU small satellite Lab is the Hyperspectral smallsat for Ocean observation (HYPSO) project. This is a project where the goal is to send a small satellite up into space to observe oceanographic phenomena with a hyperspectral camera.

The oceans contain large amounts of information describing the effects climate change are causing. The challenge is to obtain this information, consisting of the entire range from changes in ocean micro-biology to large algae blooms and hurricanes. Remote sensing from satellites is a efficient method for obtaining this type of information since it can cover large areas compared to using ships or even airplanes. Traditional satellites are both financially expensive and very time consuming during design and implementation. An alternative solution is small and highly specialized satellites called cube satellites (CubeSat). This is small satellites built up by multiple standardized cubes of 10 cm x 10 cm x 10 cm. These small satellites (SmallSats) have the advantage of being quite fast end cheap to design and implement relative to the other alternatives.

1.2 Hyperspectral imagery

Equal to common digital imaging hyperspectral imaging (HSI) obtain information by measuring the electromagnetic spectrum at chosen wavelengths or frequencies. The difference is that HSI captures a wider fraction of the spectrum for each pixel compared to common imaging methods. The human eye is able to see a relatively narrow part of the electromagnetic spectrum, hence measuring a wider part with HSI enables us to detect, identify and monitor objects or other phenomenons that are not visible to the human eye. The resolution of the spectrum measurements are also higher with HSI, so it is able to obtain more information about the visible part of the spectrum as well. Typically each pixel in a hyperspectral image consist of measured samples of the spectrum taken at different frequencies ranging from infrared to well beyond the visible. Figure 1 illustrates the difference between HSI and normal images. We can observe that each pixel in HSI contains a continuous graph of reflectance over wavelengths while a pixel in a normal image contain 3 values representing the intensity of the measured spectrum at 3 wavelength corresponding to red, green and blue in the visible spectre. The graph from the HSI is called a spectral signature.

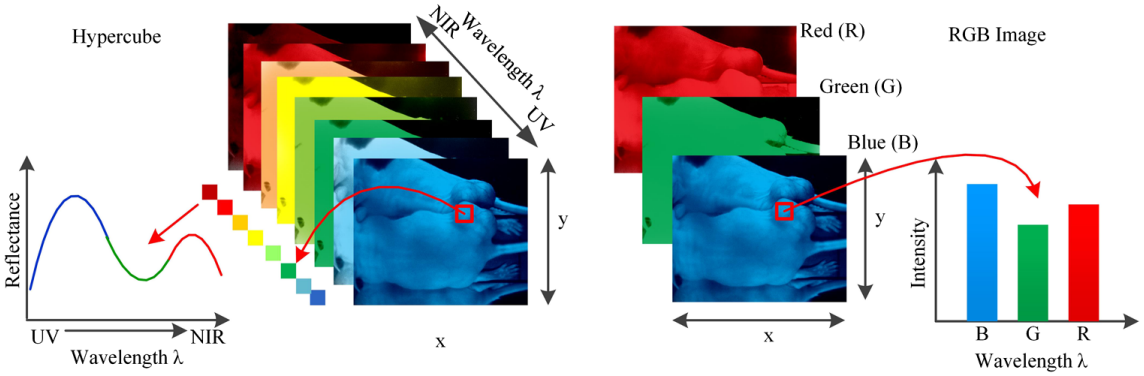

Figure 1: Gangwar [2017]

Since HSI carry more information than a regular image they are commonly refereed to as HSI cubes or hyper cubes. This is because each pixel contains more than hundred values making it common to represent the HSI as 3-dimensional matrix. The cube is a stack of 2-dimensional images, where each image then represent the reflectance measured from one narrow band of the electromagnetic

spectrum.

1.3 Anomaly detection

Remote sensing is a collective concept for detecting, classifying and monitoring objects without making physical contact with them. Usually this is done by acquiring reflected and emitted radiation at a distance. Most commonly remote sensing refers to different types of sensors on aircraft or satellites collecting spectral information of areas or objects on earth. HSI has in recent years emerged as a important technique for remote sensing. Compared to other solutions like low spatial resolution or multispectral images it has the advantage of capturing significantly more information regarding the area or object it is monitoring. This makes it possible to separate different objects based on their spectral signature thus making HSI a powerful tool for remote sensing. . Figure 2 illustrates the concept of different spectral signatures for different objects and elements in the HSI.

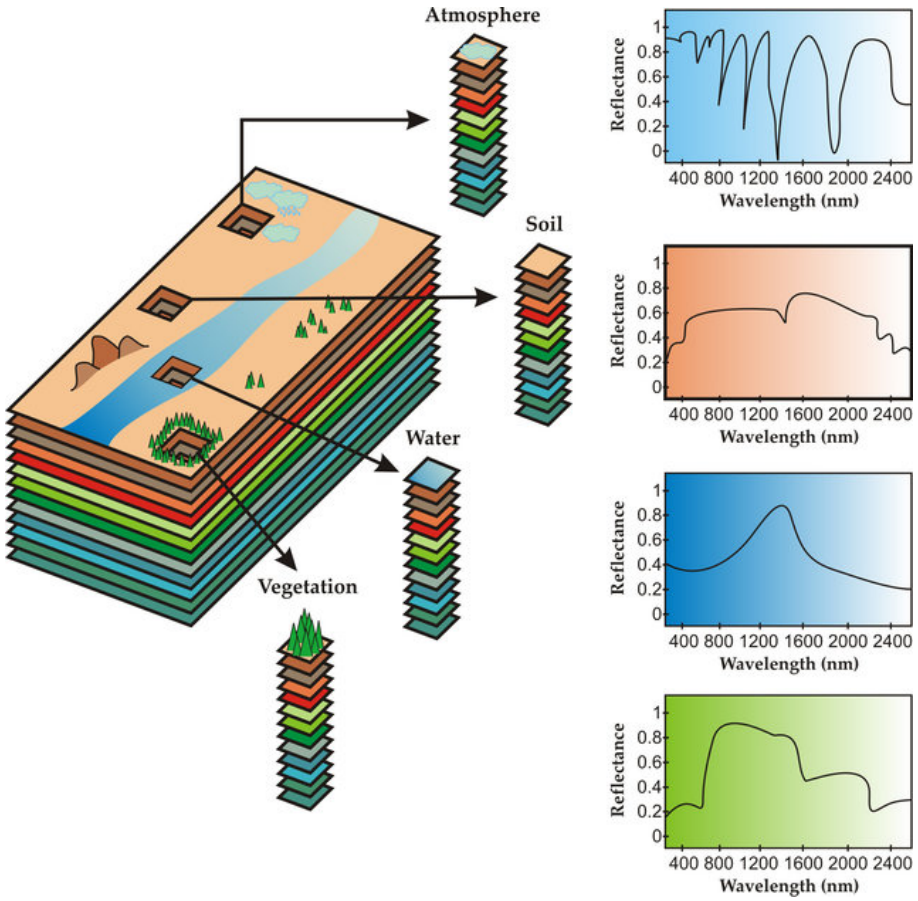

Figure 2: Illustration of HSI for remote sensing from Bioucas-Dias et al. [2012]

Multiple solutions for classifying tools have been presented, however most have the disadvantage of noe prior spectral information. This lead to the field of unsupervised anomaly detection (AD) in HSI. The general idea is to detect unnatural spectral signatures that differs significantly from its surroundings and with a low probability of occurrence. Typical examples includes ships in the sea and military equipment in battlegrounds. Traditional HSI AD solutions are mostly based on assuming a general distribution of the background, and then searching for spectral signatures that differ from that. This could be problematic since real HSI often do not correspond well to these background distribution assumptions. Modern advances in this field are among others the use of deep learning algorithms in order to find the actual features of a HSI to better model the background.

1.4 Outline of report

lag en pkt-liste over kapittelene

2 Background

2.1 Anomaly detection

An anomaly is by definition something that differ significantly from what it is compared too. There are numerous examples on anomalies in various fields. A common synonym is outlier, which falls outside the scope of what is considered to be regular. Anomalies are unknown, rare instances which means that when looking for anomalies you are looking for something unexpected. In addition to significantly deviate from the normal instances, it should also be observed quite seldom in order to fulfill the criteria of an anomaly.

In order to perform supervised AD need a fully labeled set of data. This means that you need a substantial amount of data that is representative of the data you are going to use your AD on. This data set also need to have been reviewed and labeled in order for a supervised AD to be possible to implement. Anomalies are not known a priori, thus they are anomalies. They should also by definition occur very rarely, therefor this labeled data set is very time consuming to create, if at all possible.

2.2 Anomaly detection in HSI

As described above an anomaly has certain characteristics. In order to find anomalies in a HSI one usually look for a pixel or a small group of pixels with less probability of occurrence than their neighbouring pixels. A HSI is a efficient method for detection these types of anomalies seeing as they carry way more information than a regular images. Since the anomaly is unknown, the actual size of the anomaly is also unknown. This leads to problems detecting anomalies of different sizes with the same algorithm. One effect of this phenomenon is called anomaly pixel contamination, and causes the AD to perform poorly due to other anomaly pixels. This happens because the anomaly pixel then does not stand out from the data set as clearly, which makes it harder to distinguish. There are multiple approaches to this challenge, where a common one is to use a double sliding window around the pixel under test (PUT). The pixels inside the inner window is then excluded from the detection, in order to avoid anomaly pixel contamination. The limit to how large anomalies that can efficiently be detected is then decided by the size of the inner window.

There are several known methods for AD in HSI. Many of the published methods the last 30 years are based on the assumption that the HSI follow a Gaussian multivariate distribution. The most famous one is the Reed-Xiaoli (RX) algorithm proposed in 1990. In the more recent years there have been proposed other types of AD, since the assumption of Gaussian multivariate distribution usually do not hold for real HSI.

2.2.1 The Reed-Xiaoli algorithm

This is the most widely used algorithm for Anomaly detection. It is a quite simple approach which make use of Mahalanobis distance between the background and your sample data set. The background is represented by the sample mean of the entire HSI and covariance matrix. The mathematical approach is to create the filter given in eq 1. This filter is the well known Mahalanobis distance, where $\mathbf{\Sigma}$ is the covariance matrix and μ is the sample mean. $\boldsymbol{r}$ represent a spectral signature given as a pixel vector.

$$\delta(\mathbf{r}) = (\mathbf{r} - \mu)^T \mathbf{\Sigma^{-1}} (\mathbf{r} - \mu) \tag{1}$$

There are several known variants of this algorithm. The original used the entire HSI to obtain the sample mean and covariance matrix, this is called the global RX (GRX). The local RX (LRX) is a spatially adaptive version where the covariance matrix and sample mean is calculated locally around every PUT using a double sliding window. The idea of the double sliding window is to prevent anomaly pixel contamination by not using the closest pixels in the calculation of the background statistics. The Dual-Window RX (DWRX) is essentially LRX, but instead of avoiding the closest pixels it utilizes them to optimize the detector.

Kwon and Nasrabadi [2005] proposed a non-linear version called the Kernel RX (KRX). A kernel is a reference to the kernel trick, which is to use a linear classifier on a non-linear classification task. In practice this means to map the observations into higher dimension space in order to make them linearly separable. This method can preform well but is known to be very computationally expensive.

Kucuk and Yuksel [2015] Compares the most widely used variants of the RX algorithm namely KRX, GRX, LRX, DWRX and a couple of others. The conclusion is that the DWRX was the method that yielded the highest performance wrt time consumption.

In order to use the RX method while maintaining the possibility to perform real-time computing, there exists a causal version called Casual RX. This uses a sample data data correlation matrix instead of the global co-variance matrix. The filter then changes from 1 to 2. Here $\mathbf{r_k}$ is the pixel vector currently being processed.

$$\delta_c(\mathbf{r}) = (\mathbf{r_k})^T \gamma(\mathbf{r_k})^{-\mathbf{1}}(\mathbf{r_k}) \tag{2}$$

$$\gamma(\mathbf{r_k}) = \sqrt{\frac{1}{K}} \sum_{i=1}^{K} (\mathbf{r_j})(\mathbf{r_i})^T \tag{3}$$

2.2.2 Collaborative representation based

For Collaborative representation the main idea is that all pixels in the background can be approximately represented by its spatial neighborhoods, while anomalies cannot. The collaboration is assumed to be a linear combination of neighbouring pixels. the collaboration of representation is reinforced by l_2 norm minimization of the representation weight vector.

The method utilize a double window strategy. Surrounding spatial neighborhoods in the outer region are linearly combined to produced a prediction for the test pixel within a sliding dual rectangular window

Each pixel x is in the middle of a inner rectangular window with length w_{inner}. This inner window is surrounded by by a outer rectangular window of length w_{outer}. The selected data of the outer window is used to create a 2-D matrix $\mathbf{X_s}$ containing all pixels in the outer window, but not the ones in the inner window. this matrix is now of size S times the number of spectral bands, where $S = (w_{outer})^2 - (w_{inner})^2$. The matrix $\mathbf{X_s}$ is obtained for each pixel $\mathbf{x}$.

The goal is then to find a vector α of weights such that the objective function 4 is fulfilled.

$$argmin_\alpha ||\hat{\mathbf{x}} - \hat{\mathbf{X_s}}\hat{\alpha}||_2^2 + \lambda || \sum_{\boldsymbol{x}} \alpha||_2^2 \tag{4}$$

For $\hat{\mathbf{x}} = [\mathbf{x}; \mathbf{1}]$ and $\hat{\mathbf{X}}_\mathbf{s} = [\mathbf{X_s}; \mathbf{1}]$ in order to impose a sum-to-one constraint in α. From Li and Du [2015] the solution is as follows.

$$\hat{\alpha} = (\hat{\mathbf{X}}_\mathbf{s}^\mathbf{T} \hat{\mathbf{X}}_\mathbf{s} + \lambda \boldsymbol{\Sigma}_\mathbf{x}^\mathbf{T} \boldsymbol{\Sigma}_\mathbf{x})^{-\mathbf{1}} \hat{\mathbf{X}}_\mathbf{s}^\mathbf{T} \hat{\mathbf{x}} \tag{5}$$

Where $\mathbf{\Sigma_x}$ is a diagonal matrix with the euclidean distance between the pixel $\mathbf{x}$ and all the pixels in $\mathbf{X_s}$. The resulting anomaly score is then calculated with the following equation

$$r = ||\mathbf{x} - \mathbf{X_s}\hat{\alpha}||_2 \tag{6}$$

2.2.3 Machine learning

Machine learning and in particular Deep Learning (DL) has emerged as a popular choice for AD in HSI after its great success in regular image recognition. DL has the advantage of being able to collect high-level features using unsupervised learning, which is highly relevant for HSI since it allows learning of the spectral and spatial features. Multiple methods has been published ranging from one-class support vector machines to convolutional neural networks.

Skriv mer her

3 Autoencoders

3.1 Introduction

In the field of machine learning an autoencoder is a neural network capable of learning how to efficiently compress and then reconstruct data in a unsupervised manner. Unsupervised is in this context referring to the ability to learn without the use of a labeled data-set. As shown in figure 3 the concept relies on the autoencoders ability to filter out all noise and find the most important features of the input.

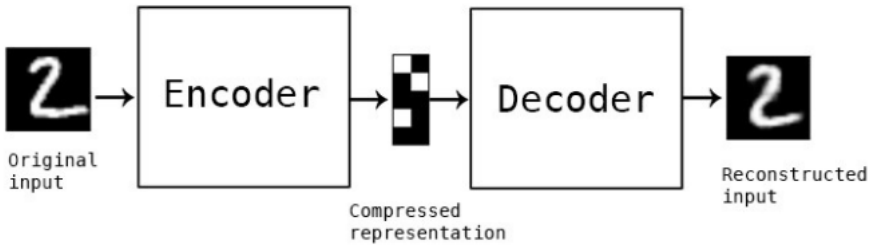

Figure 3: Illustration of the concept of autoencoders. Taken from Badr.

A more precise illustration of an autoencoder is given in figure 4. This figure shows a feed-forward neural network with the same dimension at input and output and a lower dimension in between. Each circle in the hidden layers figure represents a artificial neuron equal to the one shown in figure 5. This figure illustrate how each neuron takes in a vector of weighted values from the previous layer and a single value called the bias and sums them all together. It the outputs this sum through a activation function. The circles in the input and output layer represent the input data and the reconstructed data respectively.

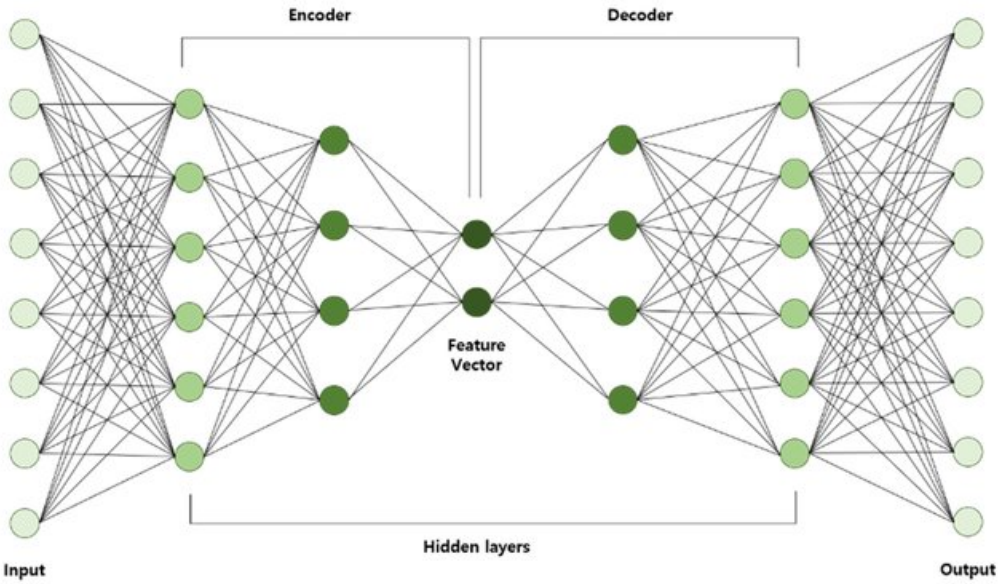

Figure 4: General structure of a autoencoder. Taken from Jeong et al. [2020]

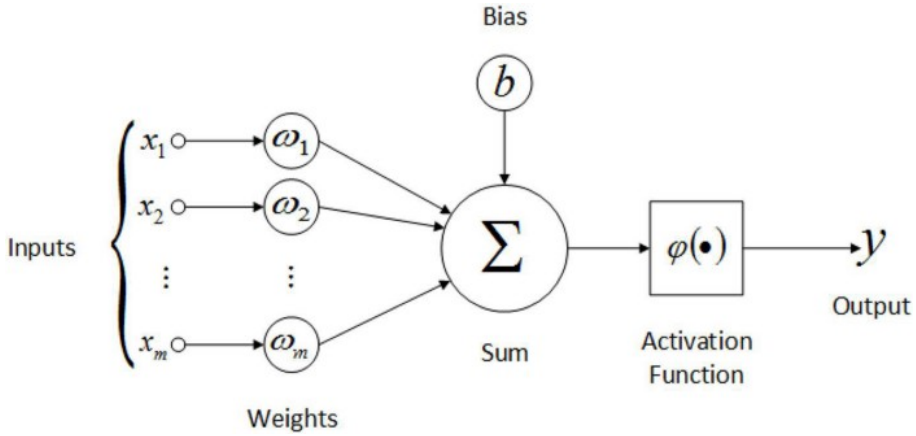

Figure 5: General structure of a artificial neuron. Taken from Roos [2019]

The activation function is a non-linear function, which enables the networks to learn complex patterns. The non-linearity is vital for neural networks as it allows them to learn non-linear patterns which is often desired. They also restrict the value of the output so it does not grow beyond or below given constraints. Typical examples of activation functions are Sigmoid, Tanh, ReLU and Leaky ReLU given in equations 7, 8, 9, 7 and 10 respectively. The expression for the output of a single neuron is then given in eq 11.

$$\phi(x) = \frac{1}{1 - e^{-x}} \tag{7}$$

$$\phi(x) = tanh(x) \tag{8}$$

$$\phi(x) = max(0, x) \tag{9}$$

$$\phi(x) = max(\alpha x, x) \tag{10}$$

$$y = \phi((\sum_{i=1}^{m} w_i x_i) + b) \tag{11}$$

When you set all the neurons together to the structure shown in figure 4 eq 11 is extended. We represent the input as vector $\boldsymbol{x}$ of length N_0. N_l is corresponding to the number of neurons layer l contains. The hidden value vector $\boldsymbol{h}^{(l)}$ of length N_l then refers to the values in . $\boldsymbol{W}^{(l)}$ is a N_{l-1} by N_l matrix containing all the weights between layer l and the previous layer. $\boldsymbol{b}^l$ is a vector containing the N_l bias-values belonging to the N_l neurons in layer l. With this notation equation 12 and 13 describes the propagation through the network.

$$\mathbf{h}^{(1)} = \phi(\boldsymbol{x}\boldsymbol{W}^{(\mathbf{1})} + \boldsymbol{b}^{(\mathbf{1})}) \tag{12}$$

$$\mathbf{h}^{(l)} = \phi(\mathbf{h}^{(\boldsymbol{l-1})}\boldsymbol{W}^{(\boldsymbol{l})} + \boldsymbol{b}^{(\boldsymbol{l})}) \tag{13}$$

3.2 Anomaly detection

The layer with the lowest number of neurons is called the bottleneck and is commonly the layer in the middle. This is the layer that after training provide the most compressed representation of the data. The performance of a autoencoder is given as reconstruction loss, which is a measure of how much the reconstructed data differ from the input. The bottleneck architecture is also the origin of anomaly detection with autoencoders. When trained the AE learns a condensed representation that best enables reconstruction, but since anomalies are rare occurrences they contribute marginally to the process of training, hence the autoencoder is not as good at reconstructing them. This leads to a simple yet efficient method of training the AE and then apply it to the data and use the reconstruction error as a indication of the probability that the input data is an anomaly.

skriv litt om forskjellige typer AD med autoencodere

3.3 Deep Belief network

A possible approach to implement an AE is to use a structure called Deep Belief Network (DBN). A DBN is a version of Deep Neural Network and consists of an input layer called the visible layer, and a chosen number of hidden layers. DBN is a popular implementation strategy to use in deep autoencoder architectures because of a technique for performing a fast layer-wise pre-training technique proposed by Hinton [2002]. The technique is based on viewing the network as a stack of restricted Boltzmann machines and training each of them separately to find a good starting point for the actual training of the AE.

3.3.1 Restricted Boltzmann Machines

In order to understand the theoretical background for layer-wise pre-training we look at DBN as a composition of simple unsupervised networks called Restricted Boltzmann Machines (RBMs). These RBMs are illustrated in figure 6 and can be stacked together to create a deep network which is easily scalable. RBMs are generative stochastic artificial neural networks consisting of two layers. The first layer is called the called the visible and correspond to the components of a observation. The second is called the hidden layer, and represent the latent variables. The two layers form a bipartite graph meaning that every single neuron in the visual layer are connected to all neurons in the hidden layer. The reason it is called restricted is the restriction that there are no connections between the neurons in the same layer.

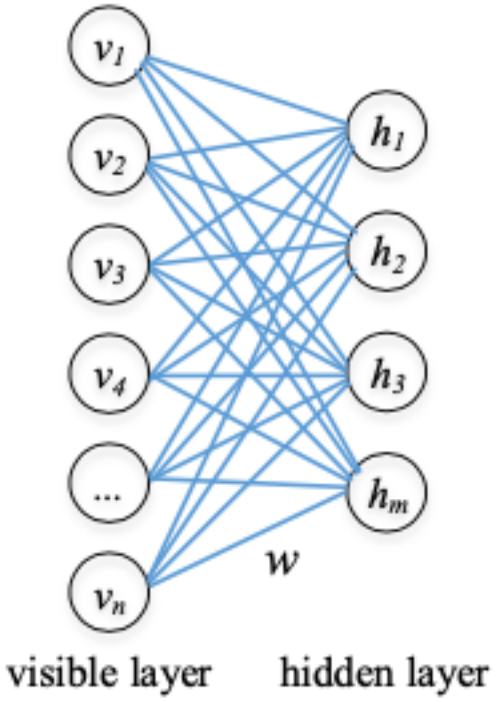

Figure 6: Illustration of a single RBM. Taken from Ma et al. [2017]

Hinton [2010] describes the RBM by considering the case where visible neurons $\boldsymbol{v}$ and hidden neurons $\boldsymbol{h}$ are binary vectors. Hopsfield [1982] then formulated the energy equation for a joint configuration of the visible and hidden neurons.

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{N_{visible}} a_i v_i - \sum_{j=1}^{N_{hidden}} b_j h_j - \sum_{i=1}^{N_{visible}} \sum_{j=1}^{N_{hidden}} v_i h_j w_{i,j} \tag{14}$$

v_i and h_j represent the binary value of visible and hidden neuron i and j, while a_i and b_j represent their bias. $w_{i,j}$ represent the weight between visible neuron i and hidden neuron j. Each possible pair of $\boldsymbol{v}$ and $\boldsymbol{h}$ is then assigned the probability from the model of Gibbs distribution in eq 15

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} \tag{15}$$

The probability of a visible vector $\boldsymbol{v}$ is then given by taking the sum over all possible hidden vectors $\boldsymbol{h}$ shown in eq 16. Fisher [2014] shows how this can be further manipulated to shows that a RBM can be considered a product of experts model.

$$p(\boldsymbol{v}) = \frac{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} \tag{16}$$

Eq 18 shows the rule used for stochastic steepest ascent given by the derivative of the log probability of the probability in eq 17. ϵ is a variable called learning rate which is used to determine the rate of change in the variables taken in the training for each iteration. The angle brackets denote expectations drawn from the distribution given in the subscript (Hinton [2010]).

$$\frac{\partial log(p(\boldsymbol{v}))}{\partial w_{i,j}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{17}$$

$$\Delta w_{i,j} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \tag{18}$$

The structure of a RBM allows us to find an unbiased sample of $\langle v_i h_j \rangle_{data}$ by using eq 19. v_i is sample i from the input data and ϕ represents an activation function.

$$P(h_j = 1|\mathbf{v}) = \phi(b_j + \sum_{i=1}^{N_{visible}} w_{i,j} v_i) \tag{19}$$

An unbiased sample of $\langle v_i h_j \rangle_{model}$ on the other hand is quite hard to obtain. It can be done by performing Gibbs sampling until it converges, which can be a infinite number of iterations which is illustrated in figure 7. Hinton [2002] proposed a solution which is widely recognized as the reason why RBMs are so commonly used today. He proposed to approximate the log-likelihood gradient by performing the Gibbs sampling k times instead of untill it converges. This is illustrated for k = 1 in figure 8 and is called Contrastive divergence (CD). Because of this most known training algorithms of RBMs use Gibbs sampling with a limited number of steps to obtaining sufficiently unbiased estimates of the log-likelihood gradient. Fisher [2014] describes Gibbs sampling as a simple Markov chain Monte Carlo algorithm which is used to obtain samples from the joint probability distribution of multiple random variables.

$$P(v_i = 1|\mathbf{h}) = \phi(a_i + \sum_{j=1}^{N_{hidden}} h_i w_{i,j}) \tag{20}$$

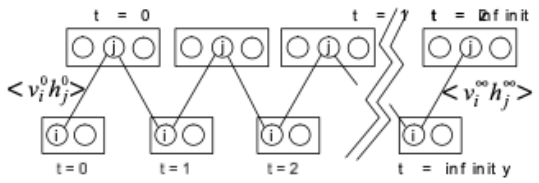

Figure 7: Taken from Hinton et al. [2006]

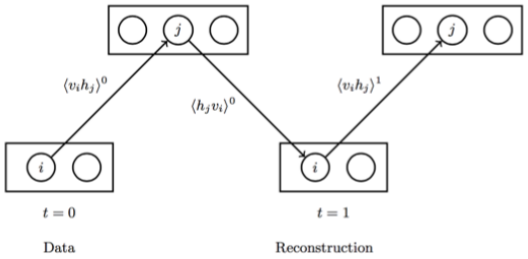

Figure 8: **?**

3.3.2 Contrastive divergence

Contrastive divergence is used in the first step of training called pre-training. In this step each layer or RBM is trained unsupervised in a fast fashion to initialize the model to enable better efficiency in the next step of training. The next step in training a DBN is called fine-tuning and is the process of training the entire DBN supervised with gradient descend. Pre-training enables each RBM to capture the most essential patterns in the data-set, and discarding the rest. Bengio et al. [2012] states that the very nature of this pre-training is yet to be fully understood, but states

that even though the samples from the model can be heavily biased they are at least moving in the right direction.

An important detail of CD is that it is commonly referred to as CD_k where k is a integer indicating the number of times Gibbs sampling should be performed where k = 1 is the most common approach. The reason this method is preferred is that it utilizes the fact that one single step of Gibbs sampling gives sufficiently unbiased samples for model training. Hence the method of CD is to run a Gibbs chain for only k steps instead of running it until it converges to finding the RBM distribution which is very time consuming.

The hidden states are activated (set to 1) if the probability given by eq 19 is greater than a random value that is drawn from a uniform distribution between zero and one. This trick is critical for the training algorithms since it regularize the information stream by only letting the hidden neurons send 1 bit of information back. For the visible values one can use the probabilities from eq 20 directly without the mapping to a binary vector. This actually reduce sampling noise and hence enabling the RBM to train faster.

The algorithm is then as follows.

Algorithm 1: CD_1-algorithm

Data : $\boldsymbol{V}^0$ is a matrix with l training vectors $\boldsymbol{v}^0$ each with n values ϵ [0 1]
Input : m: number of hidden neurons, ϵ: learning rate
Initializing
$\boldsymbol{W}$ initialized as a n by m matrix with uniformly distributed random numbers
$\boldsymbol{b}$ initialized as a vector of zeros with length m
$\boldsymbol{a}$ initialized as a vector of zeros with length n
for *(k = 1; k < l; s++)* **do**
  $\boldsymbol{r}$ is a uniformly distributed random vector of length m
  %**Gibbs**
  p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$) obtained with eq 19
  $\boldsymbol{h^0}$ activated if (p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$) > $\boldsymbol{r}$)
  $\boldsymbol{v^1}$ is obtained by $\boldsymbol{h^0}$ and eq 20
  %**Gradients**
  $\Delta\mathbf{W} = \boldsymbol{v}_l^0$ [p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$)]^T - $\boldsymbol{v}_l^1$ [p($\boldsymbol{h^1} = 1 \mid \boldsymbol{v}_l^1$)]^T
  $\Delta\mathbf{b}$ = p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$) - p($\boldsymbol{h^1} = 1 \mid \boldsymbol{v}_l^1$)
  $\Delta\mathbf{a} = \boldsymbol{v}_l^0$ - $\boldsymbol{v}_l^1$
  %**Update**
  $\mathbf{W} = \mathbf{W} + \epsilon\Delta\mathbf{W}$
  $\mathbf{b} = \mathbf{b} + \epsilon\Delta\mathbf{b}$
  $\mathbf{a} = \mathbf{a} + \epsilon\Delta\mathbf{a}$
end

3.4 Deep Neural Network Matlab Toolbox by Masayuki Tanaka

Tanaka and Okutomi [2014] describes a publicly available DBN toolbox implemented in matlab. The toolbox is based on the proposed theory in Hinton et al. [2006] and provide code for implementing and training DBN as a composite of multiple RBMs. It supports initialization of DBN, contrastive divergence pre-training, the sparse constraint, back-propagation, dropout techniques and performance measures. Cuevas-Tello and an dJuan A. Nolazco-Flores [2016] describes this toolbox in more detail by running an example and explaining in detail their choices. Hinton [2010] is a practical guide to training RBMs, which elaborates on the different parameters and how to decide their values.

3.4.1 Initialization of the DBN

The toolbox contains a function called randDBN() which takes in specifications regarding the number of layers and the number of neurons in each layer. It then call the function randRBM() once for each hidden layer. randRBM() initiates the weights from the previous layer as normally distributed random variables with 0.1 variance and all biases are set to zero.

3.4.2 Pre-training

The function pretrainDBN() takes in a DBN consisting of multiple RBMs which has randomly distributed weights. pretrainDBN() then performs layer-wise pre-training by calling pretrainRBM() for each RBM in the DBN. pretrainRBM() then perform the CD_1 unsupervised algorithm on each RBM for a chosen number of iterations. The relevant parameters used in the pre-training is described in the list below.

- MaxIter
  - This variable decides how many iterations of the CD_1 algorithm is performed for each RBM.
- BatchSize
  - This gives you the option to use stochastic gradient decent by using mini-batches of the input data vector instead of the entire input vector when updating the weights and biases.
- StepRatio
  - This is the learning rate ϵ described in eq 18.
- InitialMomentum
  - Momentum is a value used to increase the learning speed while in long ravines or intervals of consistent gradient direction. It can be described as a heavy ball accelerating downwards such as it builds up speed while the gradient has the same direction, but opposite when the gradient changes. InitialMomentum is the value used at the start of training.
- InitialMomentumIter
  - How many iterations the initial momentum value should be used.
- FinalMomentum
  - The momentum value used after InitialMomentumIter and to the end of pre-training.
- WeightCost
  - Hinton [2010] describes this variable as the a weight-decay adding an term to the normal gradient. This is a measure to reduce the risk of overfitting the model.

With these parameters and the DBN, each RBM is trained with an extended version of the CD_1

algorithm presented in the previous section. The extended algorithm is as follows.

Algorithm 2: CD_1-algorithm from Musiyaka Toolbox

Data :
$\boldsymbol{V}^0$ is a matrix with l training vectors $\boldsymbol{v}^0$ each with n values ϵ [0 1]
%
Input :
m: number of hidden neurons
ϵ: learning rate
MaxIter
InitialMomentum
InitialMomentumIter
FinalMomentum
BatchSize: Integer in range [1 l]
%
Initializing
$\boldsymbol{W}$ initialized as a n by m matrix with uniformly distributed random numbers
$\boldsymbol{b}$ initialized as a vector of zeros with length m
$\boldsymbol{a}$ initialized as a vector of zeros with length n
%
for *(i = 1; i < l; i++)* **do**
  %
  if $i <$ *InitialMomentumIter* **then**
    α = InitialMomentum
  else
    α = FinalMomentum
  end
  %
  $\boldsymbol{Ind}$ is a vector of length BatchSize with random unique integers between 1 and l
  for *(k = 1; k < l; k++)* **do**
    %
    $\boldsymbol{r}$ is a uniformly distributed random vector of length m
    %
    %**Gibbs**
    p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$) obtained with eq 19
    $\boldsymbol{h^0}$ activated if (p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$) > $\boldsymbol{r}$)
    $\boldsymbol{v^1}$ is obtained by $\boldsymbol{h^0}$ and eq 20
    %
    %**Gradients**
    $\Delta\mathbf{W} = \boldsymbol{v}_l^0$ [p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$)]^T - $\boldsymbol{v}_l^1$ [p($\boldsymbol{h^1} = 1 \mid \boldsymbol{v}_l^1$)]^T
    $\Delta\mathbf{b}$ = p($\boldsymbol{h^0} = 1 \mid \boldsymbol{v}_l^0$) - p($\boldsymbol{h^1} = 1 \mid \boldsymbol{v}_l^1$)
    $\Delta\mathbf{a} = \boldsymbol{v}_l^0$ - $\boldsymbol{v}_l^1$
  end
  %**Update**
  $\mathbf{W} = \mathbf{W} + \epsilon\Delta\mathbf{W}$
  $\mathbf{b} = \mathbf{b} + \epsilon\Delta\mathbf{b}$
  $\mathbf{a} = \mathbf{a} + \epsilon\Delta\mathbf{a}$
end

3.4.3 Supervised training with back-propagation

The Toolbox contains code for supervised fine-tuning with regular back-propagation.

3.5 HSI classification Matlab toolbox by Kang Xudong

3.5.1 Previous use in anomaly detection

The concept of using Deep Belief Networks (DBN) for AD was first mentioned in Xiong and Zuo [2016]. They based the proposed method on the low probability of occurrence that is the very nature of an anomaly. They proposed to use a DBN Autoencoder to find anomalies in geochemical data by encoding and decoding the data and observing how much each sample contributed to the decoding. Because of its low probability an anomaly will contribute in a smaller degree when training the DBN. Hence the reconstruction error after encoding and decoding will be larger for an anomaly.

Ma et al. [2017] was the first to propose a DBN AD for HSI. This method has the advantage that it does not depend on a assumption of the background distribution of the HSI. The energy model of DBN will learn the actual distribution. They later elaborated on the proposed DBN-method with adaptive weights in Ma et al. [2018]. This is to the best of the authors knowledge the current state-of-the-art performance for AD on HSI.

Figure 9 shows a more detailed illustration of how a HSI is fed to a DBN AE. The network takes in one pixel vector from a HSI at a time and encode it down to a feature-space called the code layer. Then it decodes it back to the original length and calculate the total reconstruction error of each pixel vector by taking the RMSE of the difference between $\mathbf{x}$ and $\hat{\mathbf{y}}$.

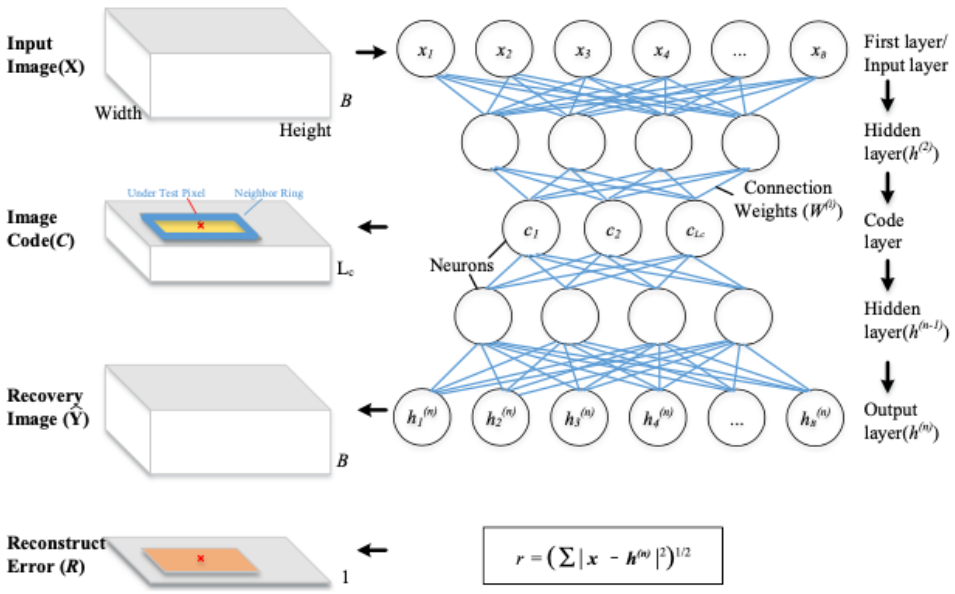

Figure 9: Illustration of the encode-decode process of a HSI with a DBN AE. Taken from Ma et al. [2018].

4 Implementation in Matlab

The implementation in matlab consists of code to initiate and train a DBN which is implemented starting from the toolbox described in section 3.4. The code is altered to fit the given case, and optimized for speed and precision. Code for performing a adaptive weights strategy based on the proposed solution in Ma et al. [2018] is also implemented. In order to evaluate and compare the implemented solution a script for obtaining relevant results and plots are also implemented.

4.1 Preprocessing of hyperspectral images

When downloading hyperspectral images the standard format is a 3-dimensional Matrix $\boldsymbol{HSI}$. The image is w pixels wide, h pixels high and each pixel contains n bands which results in HSI being a (w by h by n) matrix. Before performing anomaly detection it is necessary to change the data cube to a more fitting format. The changes are described in list 4.2

- Change from 3 dimension to 2 dimension
  - To simplify the further processing and anomaly detection the dimension of the HSI is changed. This is done by stacking all column of the (w by h) dimension of the cube creating a new 2 dimension matrix of size (p by n). p represents the number of pixels and is equal to the product of w and h.
- Remove unwanted bands
  - Some of the n spectral bands are less suited for anomaly detection. Based on Curran and Dungan [1989], Rodger and Lynch [2001] and Gao et al. [1993] the bands corresponding to the wavelengths 0.37–0.38 μm, 0.90–0.97 μm, 1.11–1.16 μm, 1.33–1.50 μm and 1.78–1.98 μm are removed. The reason being that they either absorb water or lie in the low SNR regime.
- Normalize pixel values
  - As described in section 3.3.2 the input to the DBN AE must be in between 0 and 1. To ensure this all negative values are set to zero, then the remaining values of the 2 dimensional HSI matrix is divided by the factor $\zeta = max(HSI) - min(HSI)$.
- Input anomalies
  - This is an optional feature enabling the user to change a chosen number of the pixels with pixels from other HSI. This is a functionality that aims to simplify the performance testing of anomaly detection.

4.2 Parameter selection

In the implementation there are several variables that can make an impact on the performance. Hinton [2010] elaborates on the process of deciding reasonable values based on extensive experience with training RBMs and DBNs. All of the variables are defined as a struct called **opts** to simplify the parameter list sent to different functions.

- LayerStructur
  - This variable is a array where each value correspond to the number of neurons in the layer corresponding to the position in the array. This is flexible since the code is written to support different depths and sizes of the autoencoder. The structure used in the experiments are consisting of three layers. The number of neurons in the input and output layers are set to the number of bands in each pixel of the HSI under test. The number of hidden neurons in the middle layer is 13 based on the proposed solution in Ma et al. [2018].
  - $opts.LayerStructure = [N_{bands}, 13, N_{bands}]$;
- MaxIter
  - The most important aspect of selecting this variable is to train sufficiently while not overfitting. In order to monitor the overfitting the average free energy is computed for every third iteration of running the CD_1 algorithm. If this value starts growing rapidly the model is overfitting and the pre-training should stop. The value used for experiments is 100 and is determined with empirical testing.

- $opts.MaxIter = 100$;

- Learning rate
  - Hinton [2010] states that the learning rate should be in the order of 10^{-3} times the weights. This leads to a learning rate of 0.01, which is further reduced for the last 10 iterations of training by a factor of 2.
  - $opts.stepRatio = 0.01$;

- Momentum
  - As described in section 3.4 there are three relevant values for momentum. Initial momentum is set to 0.5 enabling large initial decreases. This concludes after InitialMomentumIter = 5 iterations. After this the FinalMomentum is sat to 0.9.
  - $opts.InitialMomentIter = 5$;
  - $opts.InitialMoment = 0.5$;
  - $opts.FinalMoment = 0.9$;

- Batchsize
  - This value determines the trade off between the two biggest bottlenecks in training time. It decides the size of the largest matrix used for matrix multiplication and how many times the inner-most loop is ran. Testing has shown that number in close proximity to 10 yields the best performance, which is supported by Hinton [2010].
  - $opts.BatchSize = 10$;

- WeightCost
  - The most frequently used strategy here is to start at 0.0001 and try to increase. After testing with a range of values the one that lead to the best performance was 0.0002.
  - $opts.WeightCost = 0.0002$;

4.3 Initialization of DBN

When the HSI is in a two dimensional matrix format and all the variables above is decided the DBN is initialized. The function initDBN() takes in **opts** and utilize the variable *opts.LayerStructure* to call the function initRBM() N times, where *opts.LayerStructure* is of length (N+1). initRBM() Initialize a RBM in the following manner where dimV and dimH are the number of neurons in the visual and hidden layer respectively. randn(x, y) is a built in function in matlab returning a x-by-y matrix of standard normal distributed random values.

```
function RBM = initRBM(opts)
    RBM.W = randn(dimV, dimH) * 0.1;
    RBM.b = zeros(1, dimH);
    RBM.a = zeros(1, dimV);
end
```

After this initDBN() returns a struct called **DBN** consisting of a cell-array of N **RBM** structs as shown in the code above.

4.4 Pre-training

This implementation is an adapted version of the proposed pre-training implementation in the toolbox described in section 3.4. pretrainRBM() is a function that takes in one **RBM** struct and the **opts** struct. It updates the weights and biases of the given **RBM** struct with the CD_1 algorithm.

4.5 Fine-tuning algorithm

4.6 Finding the reconstruction error

4.7 Adaptive weights strategy

5 Results

5.1 Result metrics

5.2 Description of test data

5.3 Compared with Ma et al. [2018]

5.4 Compared with Kang et al. [2017]

6 Discussion

6.1 Alternative training algorithms

6.1.1 adaptive moment estimation optimizer (Adam)

A Vanilla AE is a reference to the simple autoencoder consisting of 3 layers, where the middle one is lower dimensions than in- and output. A very fast, yet sub-optimal algorithm called Adam optimizer is currently emerging as a popular choice for training such a AE.

6.1.2 Sparse Encoding Symmetric Machine (SESM)

Ranzato et al. proposed a novel algorithm used to extract sparse representations using a autoencoder. They compare it to the RBM-approach and conclude that RBMs are vastly more robust to channel noise while SESM is more efficient and can perform better

6.2 Different variants of loss-function in fine-tuning

6.3 Optimizing parameter values

6.4 Edge-filters on anomaly map

Bibliography

Will Badr. Auto-encoder: What is it? and what is it used for? (part 1). URL https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. 2012.

Jose M. Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, and Jocelyn Chanussot. Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2012.

Juan C. Cuevas-Tello and Manuel Valenzuela-Rendon an dJuan A. Nolazco-Flores. A tutorial on deep neural networks for intelligent systems. 2016.

P. J. Curran and J. L. Dungan. Estimation of signal-to-noise: a new procedure applied to aviris data. *IEEE Transactions on Geoscience and Remote Sensing*, 27(5):620–628, 1989. doi: 10.1109/TGRS.1989.35945.

Asja Fisher. Training restricted boltzmann machines: An introduction. *Pattern Recognition*, 2014.

Nikhil Gangwar. Hyperspectral image reconstruction from rgb image. 2017.

Bo-Cai Gao, Kathleen B. Heidebrecht, and Alexander F.H. Goetz. Derivation of scaled surface reflectances from aviris data. *Remote Sensing of Environment*, 44(2):165 – 178, 1993. ISSN 0034-4257. doi: https://doi.org/10.1016/0034-4257(93)90014-O. URL http://www.sciencedirect.com/science/article/pii/003442579390014O. Airbone Imaging Spectrometry.

Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. 1, 2002.

Geoffrey E. Hinton. A practical guide to training restricted boltzmann machines. *Gatsby Computational neuroscience unit*, 2010.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

Bomi Jeong, Hyunjeong Cho, Jieun Kim, Soon Kil Kwon, SeungWoo Hong, ChangSik Lee, TaeYeon Kim, Man Sik Park, Seoksu Hong, and Tae-Young Heo. Comparison between statistical models and machine learning methods on classification for highly imbalanced multiclass kidney data. 2020.

Xudong Kang, Xiangping Zhang, Shutao Li, Kenli Li, Jun Li, and Jón Atli Benediktsson. Hyperspectral anomaly detection with attribute and edge-preserving filters. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*, 55, 2017.

Sefa Kucuk and Seniha Esen Yuksel. Comparison of rx-based anomaly detectors on synthetic and real hyperspectral data. 2015.

Heesung Kwon and N.M. Nasrabadi. Kernel rx-algorithm: A nonlinear anomaly detector for hyperspectral imagery. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*, 42:388 – 397, 2005.

Wei Li and Qian Du. Collaborative representation for hyperspectral anomaly detection. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*, 53, 2015.

Ning Ma, shojun Wang, Yu peng, and Jinxiang Yu. A dbn based anomaly targets detector for hsi. *SPIE*, 48:1–30, 2017.

Ning Ma, Shojun Wang, Yu Peng, and Philip H. W. Leong. An unsupervised deep hyperspectral anomaly detector. *MDPI*, 2018.

Marc Aurelio Ranzato, Y-Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks.

Andrew Rodger and M.J. Lynch. Determining atmospheric column water vapour in the 0.4-2.5 m spectral region. *Proceedings of the AVIRIS Workshop*, pages 321–330, 01 2001.

Matthew Roos. Deep learning neurons versus biological neurons. 2019.

Masayuki Tanaka and Masatoshi Okutomi. A novel inference of a restricted boltzmann machine. 2014.

Yihui Xiong and Renguang Zuo. Recognition of geochemical anomalies using a deep autoencoder network. *Computers Geosciences*, 86:75–82, 2016.

Appendix

A Hello World Example

```cpp
int main {
    // This is a comment
    std::cout << "Hello World from C++!" << std::endl;
    std::cout << "I am using the default style to print this code in beautiful
    ↪  colors. Since the text is so long I have to include the 'breaklines'
    ↪  option as well" << std::endl;
    return 0;
}
```

⋮

```python
# This is a comment
print('Hello world from Python!')
print('I am using the "rrt" style to print this code in beautiful colors')
```

⋮

```matlab
% This is a comment
disp("Hello World from MATLAB!");
disp("I am using the "tango" style to print this code in beautiful colors");
```