Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Torbjørn Bratvold

Near Real-time Hyperspectral Image Classification for In-orbit Decissionmaking HYPSO-1

Near Real-time Hyperspectral Image Classification

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Sivert Bakken

August 2022

NTNU
Norwegian University of Science and Technology

Torbjørn Bratvold

Near Real-time Hyperspectral Image Classification for In-orbit Decissionmaking HYPSO-1

Near Real-time Hyperspectral Image Classification

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Sivert Bakken
August 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

NTNU
Norwegian University of
Science and Technology

Near Real-time Hyperspectral Image Classification for In-orbit Decissionmaking HYPSO-1

Torbjørn Bratvold

August 1, 2022

Abstract

With the launch of the HYPSO-1 hyperspectral imaging research small satellite in February of 2022. Improvements in processing technology has made on-board processing of larger amount of data feasible on such small computational platforms. For the HYPSO-1 satellite today the biggest bottleneck is how much data that can be sent over the radio-link. This work aims to create a novel machine learning model for near real-time classification of hyperspectral images. The model takes advantage of the way the HYPSO-1 captures image cubes one spatial line at a time to achieve near real-time classification. This way larger autonomy can be given to the satellite to decide when image captures should be initiated and which images should be prioritised for downlink.

An exprimentational approach lead to the building and training of a model capable of achieving 95.0% accuracy when classifying land, clouds and water on a chosen self labelled raw HYPSO-1 capture with no pre-processing or radiometric calibration. The model experiments with using graphical convolutional layers instead of the more conventional convolutional neural network. The model was built using the Python framework PyTorch and PyTorch Geometric. The model was also tested with the standard hyperspectral datasets Indian pines and SalinasA to gauge a more general performance of the model.

Sammendrag

Småsatellitten HYPSO-1 ble skutt opp i februar 2022 for å brukes som en hypespektral avbildningsplattform for havforskning. Utviklingen innen prosesseringsteknologi har gjort at også småsatellitter kan gjøre ombord prosessering av større mengder data. Den største flaskehalsen for HYPSO-1 er i dag mengden data som den klarer å overføre over radiolinken. Denne oppgaven har som mål å utvikle en ny maskinlæringsmodel for å oppnå nær sanntid klassifisering av hyperspektrale bilder. For å oppnå dette utnytter modellen at HYPSO-1 tar bilder ved å skanne de romlige linjene hver for seg. Målet er at modellen skal kunne gjøre det mulig å gi satellitten større selvstendighet til å bestemme når billedtakning skal gjennomføres og hvilke bilder som burde prioriteres for nedlink til bakkestasjonen.

En eksperimentell tilnærming har ført til en modell i stand til å oppnå 95.0% nøyaktighet på klassifisering av land, skyer og vann på et utvalgt bilde tatt av HYPSO-1. Det utvalgte bildet inneholder hyperspektral rådata og har hverken gjennomgått pre-prosessering, eller radiometrisk kalibrering for å oppnå denne nøyaktigheten. Modellen eksperimenterer med å bruke et graph convolutional network i stedet for det mer konvensjonelle convolutional neural networket. Modellen ble laget i Python rammeverket PYTorch og PyTorch Geometric. Modellen ble også testet mot de hyperspktrale standard datasettene Indian Pines og SalinasA for å få mer innsikt i ytelsen til modellen på mer generelt grunnlag

Preface

The work of this thesis was conducted during the spring of 2022 and concludes the five year Master's degree programme Electronics Systems Design and Innovation, with a spe-cialization within communication and signal processing. The Master's thesis is submitted to the Departmentof Electronic Systems at the Norwegian University of Science and Technology.

I would like to thank my supervisor Milica Orlandic for giving me the needed motivation for finishing this thesis. I would like to thank my co-supervisor Sivert Bakken for all his help and support. I would like to thank my friends for giving me needed breaks from the work. Especially I want to thank Simen and Simen for giving me some final input and helping me proofread.

Lastly I would like to thank the amazing people in the Orbit NTNU and HYPSO team. You have taught me so much, and without you I would never have dreamt of working with space-technology and satellites.

Contents

Figures

Tables

Code Listings

Acronyms

CNN Convelutional Neural Network. 15, 17, 18, 37

FPGA Field Programmable Gate Array. ix, 9, 10, 13, 15, 39

GCN Graph Convolutional Network. xiii, xiv, 17, 18, 24, 25, 37, 38

GNN Graph Neural Network. 15, 18, 19

GPU graphics processing unit. 9, 10

HDL hardware descriptive language. 9

HS Hyperspectral. 14

HSI Hyperspectral imaging. xiii, 7–9, 14, 19, 20, 40, 41, 43

HYPSO HYPer-spectral Smallsat for ocean Observatio. 8, 9

NTNU Norwegian University of Science and Technology. 8

Glossary

HYPSO-1 The first operational nanosatellite built by NTNU launched in February 2022. x, xv, 8, 9, 13, 32, 37, 39–41, 43

in-situ The physcical place were an event happens. 1

nanosatellites Satellite between 1 and 10 kg. 1, 9

Chapter 1

Introduction

1.1 Motivation

The domain of satellite imaging has revolutionized the field of earth observations [1]. With the advent of frequent and global coverage for remote sensing devices given by satellite platforms we can now tell more about the globe and its condition than ever before. The miniaturization of electronics has enabled smaller satellites to be used in big research campaigns [1]. In addition to traditional monolithic satellites, small CubeSats in the order of nanosatellites can contribute with valuable data whilst being cheaper and faster to develop, thus enabling innovative technology to be tested [2].

While earth observation is today performed by many different sensors such as radars or lasers, the most well-known form of earth observation is probably optical imaging. nanosatellites have optical limitations due to their small size, making trade-offs of capture configurations important [2]. In order to guide these trade-offs, it is desirable to obtain information about the data in-situ. By processing information about the data as it is happening, information can be converted to knowledge in real-time instead of using valuable resources to downlink it and process it on ground.

Neural networks is today the state-of-the-art within image classification, and

the prospect of implementing machinelearning models on-board satellites is promising for achieving better on-board decision making. While conventional microprocessors for the most part runs instructions in sequency and are thus inefficient for running neural networks, the inclusion of field programmable hardware on smallsatellites opens the door for parallelisation and implementation of on-board neural networks.

By introducing on-board image classification the we open the door for giving larger autonomy to satellites so that they can make independent decisions based on the data they collect, thus improving operational performance and lowering the need for human input during operations.

The questions this thesis wants to answer are:

- **1:** *Can we build a machine learning model capable of classifying an image line as it is captured in near real-time?*
- **2:** *How can on-board and near real-time hyperspectral image classification improve satellite operations?*

Chapter 2

Background and theory

The background and theory section contains the concepts needed for understanding of hyperspectral imaging and graph neural networks. It outlines the HYPSO-1 mission and how near real-time classification has the potential to improve satellite operations.

2.1 Hyperspectral imaging

2.1.1 Excitation of atoms

In quantum mechanics an excited state of an atom molecule or nucelus is described as a state were the atom contains more energy than the absolute minimum that the atom can contain, this minimum is called the ground state of the atom. When an atom interacts with other particles or electromagnetic waves (photones) it will be brought from its current energy state to a state of higher energy. This can be from the ground state or a previously excited state [3].

To illustrate excitation we will consider the hydrogen atom which is one of the simplest nuclear systems, figure 2.1. The hydrogen starts out in the stable ground state with the minimal possible energy level, ***E1***. The hydrogen then interacts with an incoming photon exciting the electron to a higher energy state***E2*** by absorbing the energy of the photon. Further energy states are denoted as ***E3***, ***E4***, ***E5*** ,etc. The

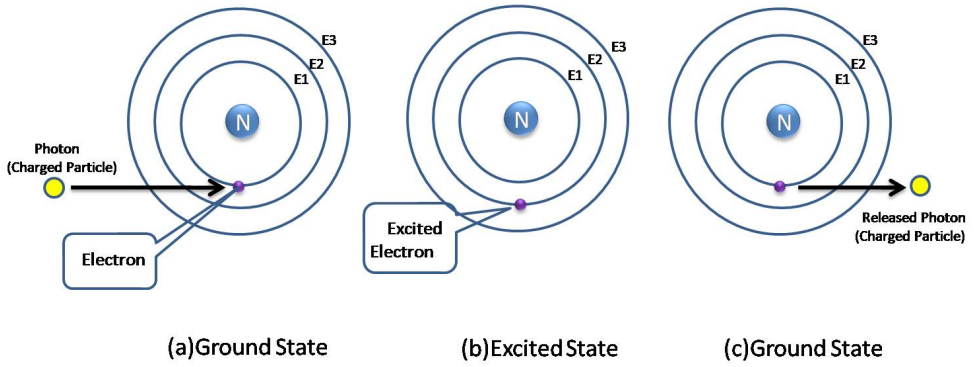


Figure 2.1: Hydrogen excitation. Figure obtained from [5]

atom will persist in this excited state for only a short time before returning to the stable ground state, ***E1***. To return to the ground state the atom will release energy in the form of electromagnetic waves. These waves can sometimes be observed as visible light detectable by the human eye. If too much energy is absorbed, the electron can become unbound to the nuclear core creating an ionized atom [4].

The following sections on energy levels and their equations are taken from [4]. The energy required to reach an excited state is defined as the difference between between the energy of excited state and the ground state. This energy is usually measured in electron volt (eV) where

$$1eV = 1.6 \times 10^{(-19)}J \tag{2.1}$$

In the case of the hydrogen atom the ground state energy $E1 = -13.6eV$ and the first excited state $E2 = -3.4eV$. The energy E needed to excite a hydrogen atom from E1 to E2 is thus given by equation 2.2 [4]

$$E = E2 - E1 \tag{2.2}$$

All systems has a distinct set of energy levels meaning that the excitation from one energy level to another will always require the same amount of energy for a specific atom or molecule. In other words when a system de-excites it will release

specific sets of electromagnetic radiation native to that system.

For example, we can consider a hydrogen atom de-exciting from E3 to E2. The released energy E is given by

$$E = E2 - E3 \tag{2.3}$$

Here $E2 = -3.4eV$ and $E3 = -1.51eV$. Inserted in the formula above we get

$$E = -1.51V - (-3.4eV) = -1.89eV = 3.028\ 113\ 838 \times 10^{-19}J \tag{2.4}$$

The frequency of the released electromagnetic radiation can then be calculated by utilising the Planck-Einstein relation [6]

$$f = \frac{h}{E} \tag{2.5}$$

Were f is the radiation frequency, h is Planck's constant, equal to $6.62\ 607\ 015 \times 10^{-34} J\ Hz^{-1}$ and E is the radiation energy.

The radiation wavelength can then be calculated by the equation [6]

$$\lambda = \frac{c}{f} \tag{2.6}$$

Were λ is the radiation wavelength, c is the speed of light, $3.00 \times 10^8 m/s$ and f is the radiation frequency. By rearranging equation 2.5 and 2.6 we get

$$\lambda = \frac{h \times c}{E} \tag{2.7}$$

By inputting the emitted energy between E2 and E3 we get a wavelength of $656nm$ which corresponds to the color dark red in the visible light spectrum.

2.1.2 Spectroscopy

Spectroscopy is a field within chemical science that studies the absorption and emission of light and other radiation by matter. Spectroscopy is made up of many

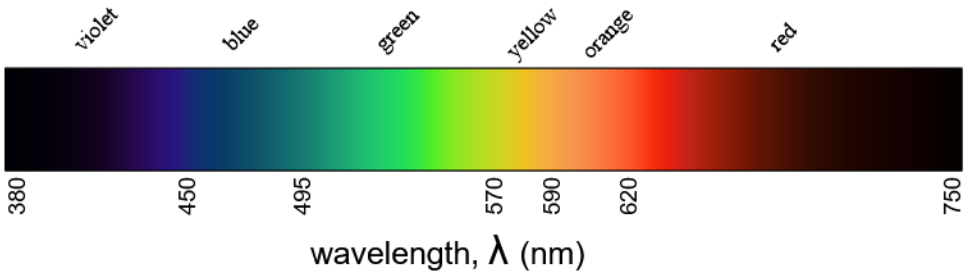


Figure 2.2: Visible light spectrum, taken from [10]

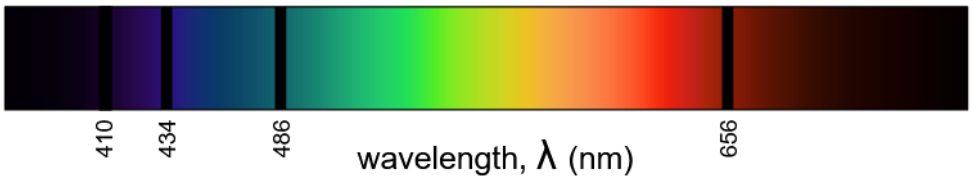


Figure 2.3: Hydrogen Absorption Spectrum, taken from [11]

different methods for studying molecules and atoms. In the world of astrophysics spectroscopy has been used to determine the composition of stars based on their electromagnetic radiation. [7]

Although light visible to the human eye has no define cut-off the spectrum of visible light is generally considered to contain wavelengths between 400nm to 750nm with the distribution as can be seen in figure 2.2 [8]. Before interacting with an atom or molecule, visible light will have the same intensity for all wavelengths as seen in figure 2.2. When visible light interacts with an object some of the energy contained in the light will be absorbed and cause excitation of the objects atoms. By observing the light spectrum after the light has passed through the material we can gain much insight of the composition of the material. In figure 2.3 we see the light spectrum again after passing through a hydrogen atom. In the figure we can see four distinct black lines in the spectrum. The energy in these bands correspond to the excitation energy of the hydrogen and are absorbed by the hydrogen atom. Hence these kinds of spectrum's are called absorption spectrum's. [9]

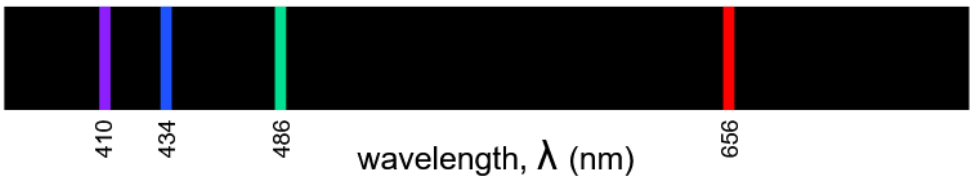


Figure 2.4: Hydrogen Emission Spectrum, taken from [12]

As mentioned in the previous subsection, the absorbed energy levels are the same for all materials of the same instance. By equation 2.7 this means that the corresponding absorbed wavelength will also be consistent with each material. Thus the observed absorption spectrum can be used to identify the material which the light has passed through.

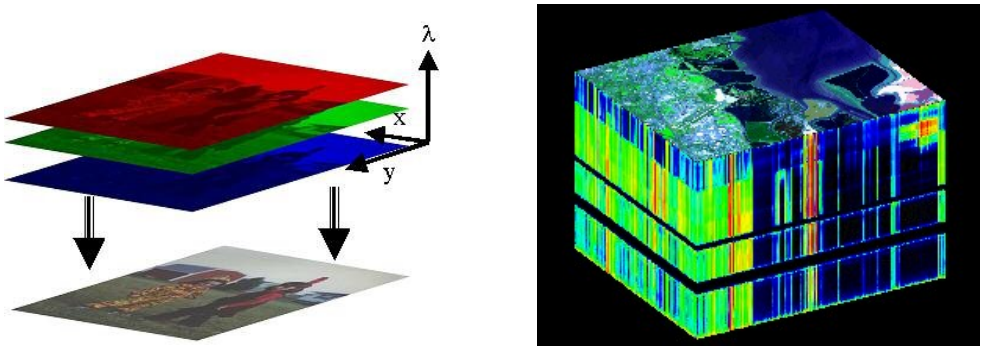
The same can also be achieved by the opposite by looking at the spectrum of the wavelengths emitted by the material after light has passed the material. In 2.4 we see the hydrogen emission spectrum. We see that the lines perfectly matches the dark lines in the hydrogen absorption spectrum in figure 2.3

2.1.3 Hyperspectral images

Section taken from [13]. A standard RGB image has three values for each pixel, namely red, green, and blue (RGB) captured from the visible light spectrum. Hyperspectral imaging (HSI) is a spectral imaging technique that can capture hundreds of wavelength bands for each pixel in the image. HSI can thus extract many times the spectral information than that of an RGB camera. Depending on the image sensor characteristics HSI can also capture wavelengths outside the visible spectrum for additional spectral information

An HSI captures several hundred bands that form an image data cube when stacked together. In figure 2.5a and 2.5b we can see three dimensional representations that illustrates the difference between a traditional RGB image and a hyperspectral image.

Objects have distinctive spectral signatures, which can be captured by an HSI. These signatures act as fingerprints which can be used to identify the composition



(a) RGB Cube, taken from [14] **(b)** HSI Data Cube, taken from [15]

Figure 2.5: Comparison RGB and HSI data

of scanned objects. It is thus possible to extract information about materials that would be all but impossible even with detailed spatial data from a classical RGB image.

2.2 HYPSO mission

The HYPer-spectral Smallsat for ocean Observatio mission (HYPSO) is a series of cubesat missions by the Norwegian University of Science and Technology (NTNU). The satellites are 10x20x30 cm which host a novel hyperspectral pushbroom imaging payload used for earth observation. Its specific mission is to detect and characterize ocean color features such as algae blooms, phytoplankton, river plooms. etc. The first satellite in this series is the HYPSO-1 which was launched in February of 2022 [16].

In regards to the operation of the HYPSO-1 the power budget is less restrictive than the downlink time, thus enabling the satellite to generate more data than what is downlinkable. This means that knowing which captures contain interesting data before downlink as well as finding methods for raising the chance of good captures can greatly improve the efficiency and results of capture campaigns.

2.3 In orbit processing

Advances in technology has lead to better and smaller processing technology, leading to the possibility of arming nanosatellites with processing tasks outside their primary functionality. This opens up for a wide variety for new concepts of operations for satellite platforms. The more data that can be processed in orbit means that less raw data needs to be downloaded from the satellite and one can save power and contact time intended for downloads. This also increases overall system responsiveness as the amount of data needed to be processed and analysed on the ground decreases, which in turn also supports the satellite with more data for autonomous decision making.

In context of the HYPSO mission the HYPSO-1 is envisioned as an early warning system for algae detection along the Norwegian coast. The more data that can be processed rapidly on-board the HYPSO-1 the more responsive the rest of the early warning system will become.

2.3.1 Field Programmable Gate Array (FPGA)

Field Programmable Gate Array (FPGA) is an integrated circuit device containing logical elements organised in a matrix structure (array). These logical gates can be reprogrammed in the field at any time using low level hardware descriptive language (HDL). These devices are particularly useful for static tasks as they can perform calculations in hardware much faster than a software implementation.

The HYPSO-1 satellite contains an on-board FPGA for hardware compression of HSI images and other hardware acceleration tasks. The reprogram-ability of the FPGA provides the flexibility for implementing additional functionality, such as neural networks for image classification.

The basic structure of an FPGA is inherently parallel and is therefor well suited for implementation of neural networks. In contrast to graphics processing units FPGAs are generally smaller, requires far less power and produces less heat. These are qualities very beneficial for space usage because of the limited power output

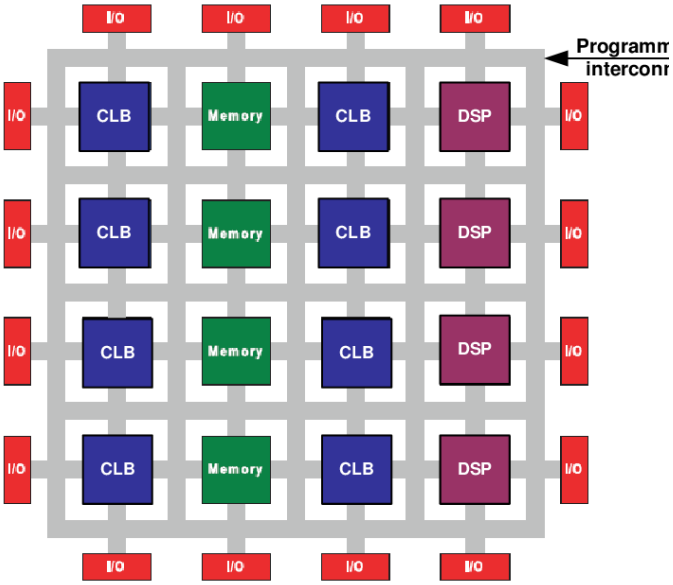


Figure 2.6: Internal structure of Xilinx FPGA [17]

of small-satellites and air cooling not being an option in space. These advantages comes at the drawback of model implementation being more challenging on FPGA than on a GPU, especially for very complex models.

2.4 Classification of Hyperspectral Images

2.4.1 Image segmentation

Image segmentation is an operation in image-processing were features in an image are divided into different regions (segments) in the image. An example of segmentation can be seen in figure 2.7. Here the image has color-coded regions in the image depending on what they contain, people have been colored red, vehicles are blue and vegetation is greenish. Segmentation is an effective method for extracting information from an image.

Figure 2.7: Segmented image, taken from [18]

2.4.2 Pixel based segmentation

In the context of hyperspectral images the standard segmentation method is pixel based segmentation. Here each pixel in an hyperspectral image is assigned to one of a pre-defined set of classes. In figure 2.8 we see the Salinas Scene which is a hyperspectral data-set collected over Salinas Valley in California. The left image 2.8 shows an rgb composite of the hyperspectral data, while the right side contains the segmented classes assigned to the image. The Salinas set contains 16 different classes of crops as well as a "don't care" class (noted in black) which is used as a collection class for pixel types that are not interesting for classification.

2.4.3 Current methods

The amount of standard pre-prepared hyperspectral images data sets are more limited than many other image classification tasks. To name some of the most popular data sets:

- Pavia University
- Indian Pines

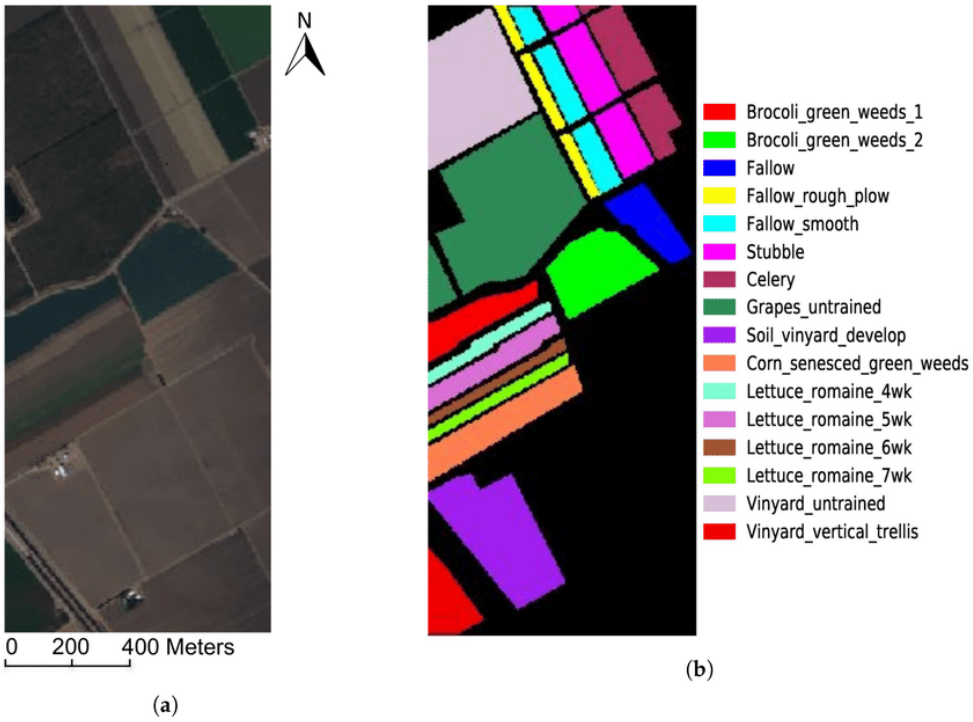


Figure 2.8: Salinas Scene segmented, taken from [19]

- Salinas Scene

One inherent disadvantage of these data sets are that they do not have overlapping classes, as such one can't use one image for training and others for testing. The standard training and testing method is thus separating each individual image into training and testing sections.

The training data is typically extracted as a subset of the whole cube in the form of patches. Considering a HSI cube with dimensions MxNxB (width x height x number of bands) subsets of SxSxB would be extracted for model training and testing.

Typically the data set is split into either 10% or 30% training data and the rest is used for testing. The lack of volume in hyperspectral training data makes it harder to train good models for practical usage, but provides an interesting challenge for innovation in this field.

2.4.4 On-board Classification

Analysis of the data gathered from the HYPSO-1 satellite is the most vital part of utilizing the satellite for practical purposes. As operational effectiveness and the amount of data generated by the satellite increases. The need for more rapid and accurate data-analysis will be needed. An initial step could be utilizing data processing on the ground were the researchers and operators would have direct opportunity for analysis and decision making based on the satellite data. This step will greatly decrease the time needed for in-person decision making as the data processing tools could notify ground personnel of the most promising data. General on-board classification of the data could potentially be performed by current state of the art models available to the public. Such models could potentially be implemented directly on the FPGA of HYPSO-1.

2.4.5 Real-time Image Classification

While on-board classification has the potential for improving analysis and decision making time it will not improve the satellite data acquisition process in itself. As of now the image acquisition is decided by the ground operators with no on-board decision making by the satellite. When the satellite performs image capture it will capture the image line for line while passing overhead. An illustration of this can be seen in figure 2.9

By utilizing the push-broom capturing method of the HYPSO-1 it is possible to increase satellite capture autonomy by implementing near real-time classification as the satellite passes over the Earth. As the satellite captures images line by line it can also be made to classify these lines the moment they are captured. Doing this can allow for several interesting concepts of operations for the satellite. By having the satellite classify while continuously scanning an area of interest it can wait with activating a slew maneuver until algae or other objects of interest has been detected by the classification algorithm. This can allow for more dynamic operations with higher satellite autonomy while conducting captures.

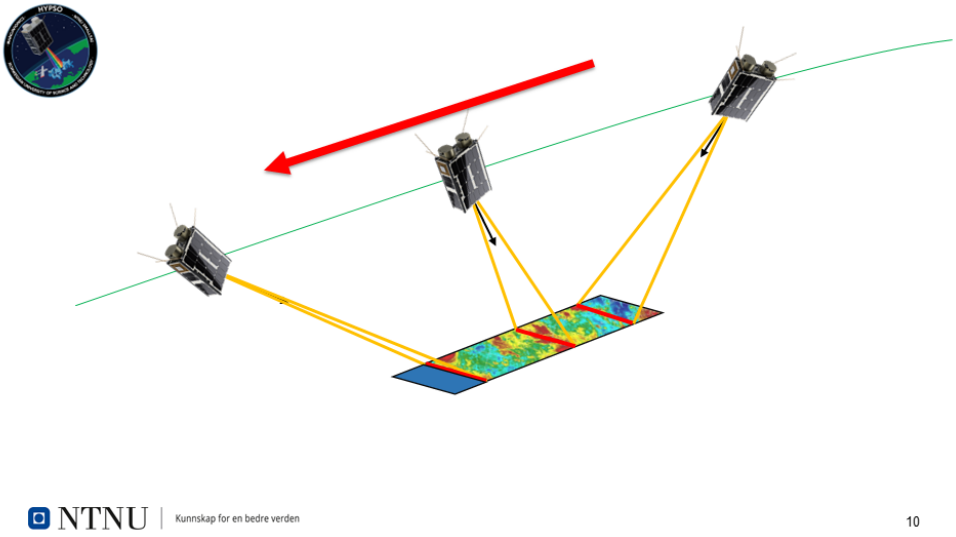
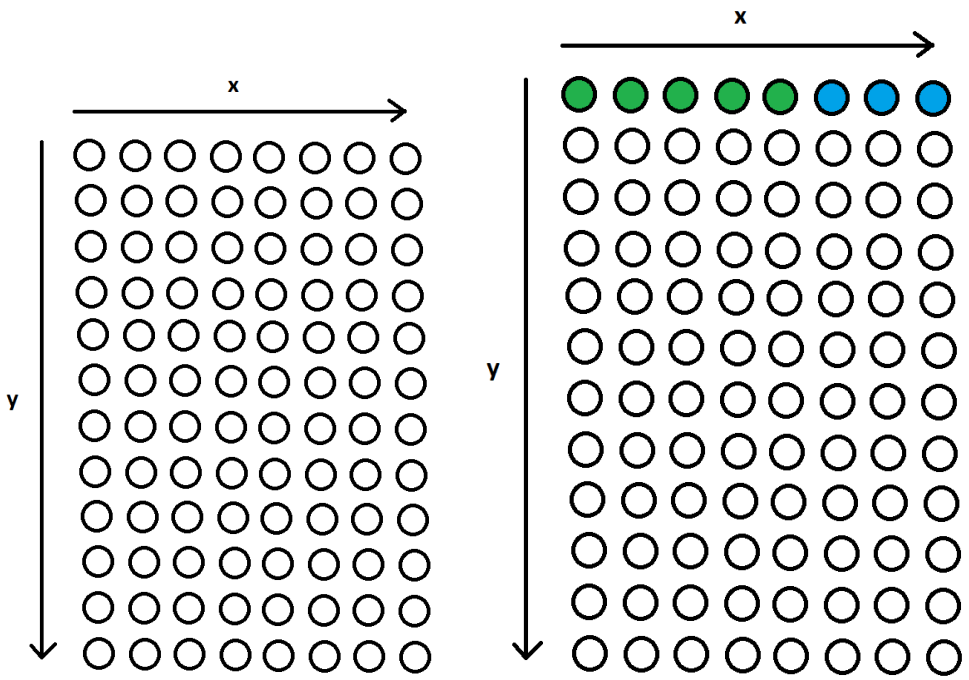


Figure 2.9: The HYPSO-1 satellite performing an image capture

Additionally by continuously classifying, the satellite can avoid capturing areas with cloud cover by performing a capture maneuver first when the camera is clear of the clouds.

Real-time classification is illustrated in 2.10a and 2.10b were we can see a representation of the spatial dimension of a HS image. Here each circle represent a pixel in the HSI cube. With the push broom method the full image is composed by scanning each x dimensions individually while moving down the y dimension. The HS cube can then be classified near real-time by classifying each x dimension individually as seen in figure 2.10b

Performing real-time classification by classifying a single x dimension at a time has the inherent weakness of providing a very limited amount of contextual data for the classification model and is thus expected to perform worse than models with access to a full data cube. Line by line classification is thus not expected to directly compete with general state of the art HSI classification models, but rather be tailored for this specific scenario.



(a) HSI spatial dimension representation, y = scan direction, x = slit size, each circle represent one pixel

(b) HSI spatial dimension representation, y = scan direction, x = slit size, each circle represent one pixel

2.5 Graph Neural Networks

While the most popular and well known artificial neural network for image classification is the Convelutional Neural Network (CNN). In this thesis the use of Graph Neural Network (GNN) are proposed. GNNs generaly uses fewer convolutional layers and thus leads to generally less complex models. This is advantageous as a too complex model might be very hard to convert to run on an FPGA.

2.5.1 Graphs

Graphs are the fundamental part of a GNN. Graphs are a data structure consisting of two components: nodes (verticies) and edges. A graph G can be defined as $G = (V, E)$ were V is a set of nodes and E are the edges between the nodes. These edges can be directional or uni-direct if dependencies between edges goes both

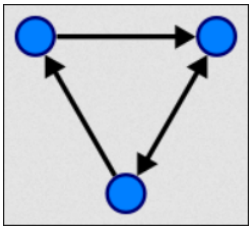


Figure 2.11: Directed Graph, take from [20]

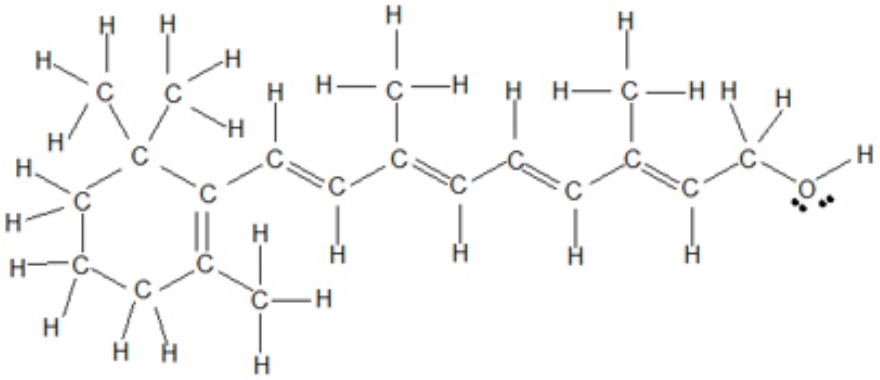


Figure 2.12: Vitamin A, taken from [21]

ways. Figure 2.11 shows a simple graph structure with two directional and one uni-direct edge. [20]

In today's world graphs are all around us, and this simple data structure can be used to describe many complex relationships between objects and people. An example is molecules which can form extremely complex structures e.g Vitamin A in figure 2.12. Each atom can be modelled as nodes while the bonds between them can be represented as edges.

2.5.2 Node embedding

Node embedding refers to mapping the input graph to N-dimensional space. For a computer it is easier to calculate how close nodes are in N-dimensional space than from a graph. The closeness of the embedded nodes can then be calculated by various means [22].

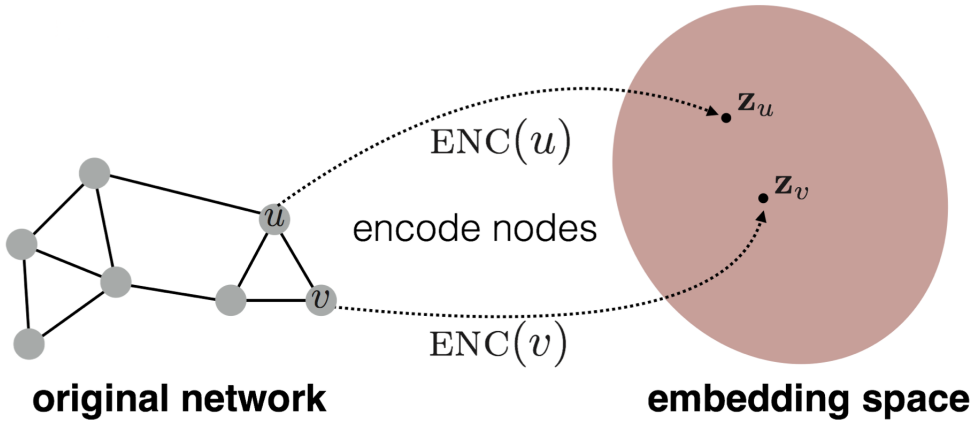


Figure 2.13: Node embedding, taken from [23]

Node Encoding

The following section is taken from [23]. As shown in 2.13 node embedding is done via an Encoder. To illustrate how an encoder works we consider "Shallow" encoding which is the simplest encoding approach. Here the encoder is just an embedding-lookup represented via the equation:

$$ENC(v) = \mathbf{Zv} \tag{2.8}$$

Here the encoder is denoted by ENC which encodes the node v to the N-dimensional space. $\mathbf{Z}$ is a matrix where each column indicates a node embedding, the number of rows in $\mathbf{Z}$ equals the dimension of the embedding. v is the indicator vector which is a vector consisting of all zeroes except in the column indicating the node v, were it is a one. Thus all nodes are assigned an unique embedding vector in "shallow" encoding.

2.5.3 Graph Convolutional Network

In figure 2.14 an illustartion of a Graph Convolutional Network (GCN) is shown. GCNs were introduced as a method for applying neural networks to graph-structured data. The generic structure of GCNs are in many ways similar to the more well known Convelutional Neural Network (CNN)s. They start with graph convolu-

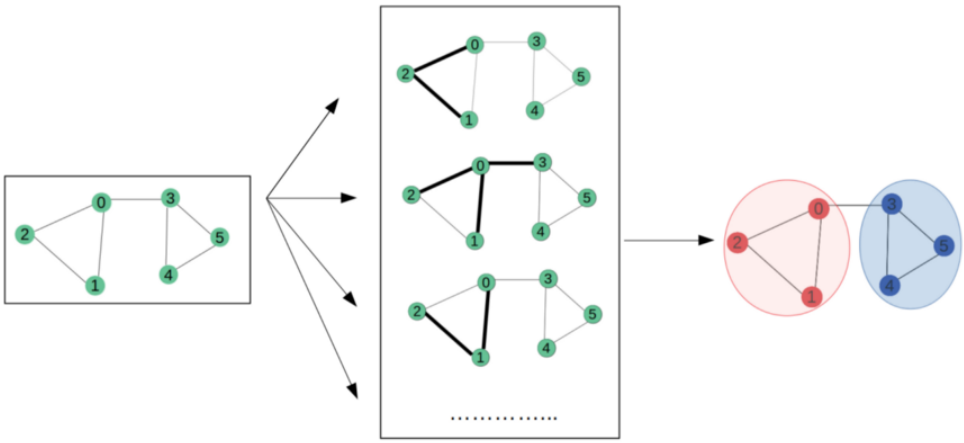


Figure 2.14: Graph Convolutional Network, taken from [24]

tional layers which passes the convulated data to a fully connected network of linear layers which ends with a nonlinear activation layer.

The convolutions in the GCN performs in a similar way to the CNN were data features are learned by inspecting neighboring nodes. While CNNs are specially built to operate on regular (Euclidean) structured data, GNNs is a more generalized version were the number of nodes and connections between them may vary and nodes can be ordered in non-Euclidean ways. [24]

Chapter 3

Implementation

This section starts by outlining the way data has been modelled as graphs for training and testing. It then presents the datasets, attempted approaches that did not improve model performance and the final model implementation.

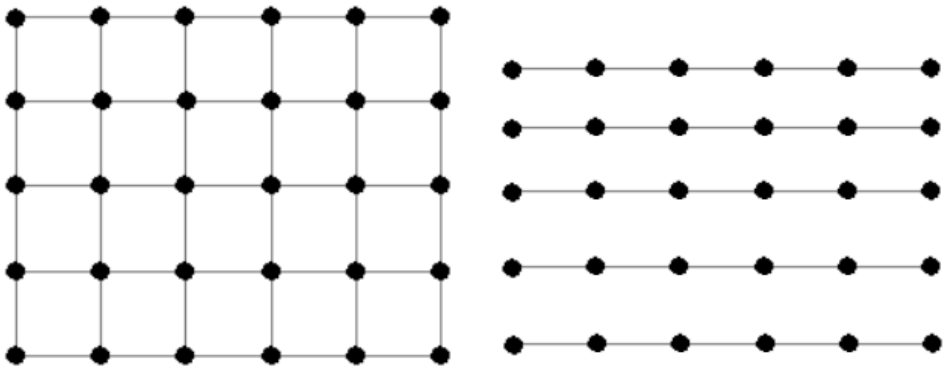
3.1 Representing HSI as graphs

To be able to utilise GNNs for HSI classification the HSI data must first be modeled as a graph structure. As explained in 2.5.1 the graphs contains nodes and edges. Here the nodes represent a certain data point and the edges represent the correlation between these points.

In the way of HSI or any normal RGB image one can represent the image itself as a graph were each pixel is represented as a node and the edges between the nodes corresponds to the correlation between any two pixels. In terms of correlation between nodes in an image this correlation can be seen as how likely the pixel next to a given pixel will be of the same class. This correlation will be the strongest with pixels right next to eachother. In other words if a specific pixel has been classified as grass the pixels right next to it will have a greater chance of also being grass. This probability will shrink the further in the spatial domain you remove yourself from the pixel, as such the edge relation will become weaker

between far away pixels.

The basic representation would thus be showing an image as a grid were each pixel has an edge to the neighbouring pixels



(a) Graph representation of a hyperspectral image

(b) Graph representation of real-time HSI classification

In the case of real-time classification were the spatial dimension is classified line by line the whole image cube will not be available to the model. Thus the edges between each line in the y direction is not implemented. The graph representation will thus bee what is shown in figure 3.1b. Meaning that each data point only share edge with two adjacent nodes.

3.2 Datasets

Four different data-sets have been used for training and testing on the model

- Indian Pines
- Salinas A
- Self labeled HICO dataset
- Self labeled HYPSO-1 dataset

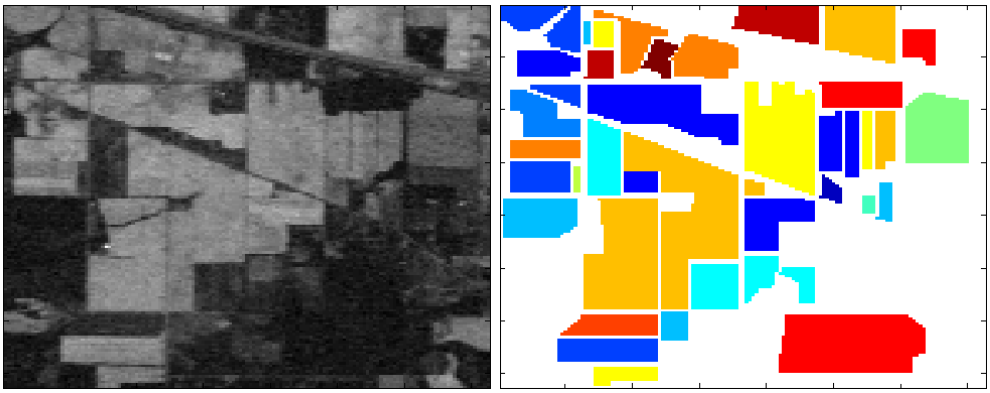
A general overview of the datasets can be seen in table 3.1.

Dataset	**Indian Pines**	**SalinasA**	**HICO**	**HYPSO-1**
Capture type	Reflectance	Reflectance	Reflectance	Reflectance
Spectral range	400-2500 nm	400-2500 nm	404-896 nm	387-801 nm
Spatial dimensions (pixels)	145 x 145	86 x 83	506 x 1996	684 x 956
Number of bands	200	224	128	120
Number of labelled classes	17	6	3	3

Table 3.1: Dataset overview

3.2.1 Indian Pines

This dataset is gathered from North-western Indiana by an AVRIS sensor. This data is gathered aerially with a spatial dimension of 145x145 pixels and 200 reflectance bands. The bands vary from 400 to 2500 nm. The datasaet originaly has 220 bands, but they have been reduced to 200 by removing bands covering water absorption. The data set consists of 16 different classes and can be seen in 3.2a and 4.6 [25].

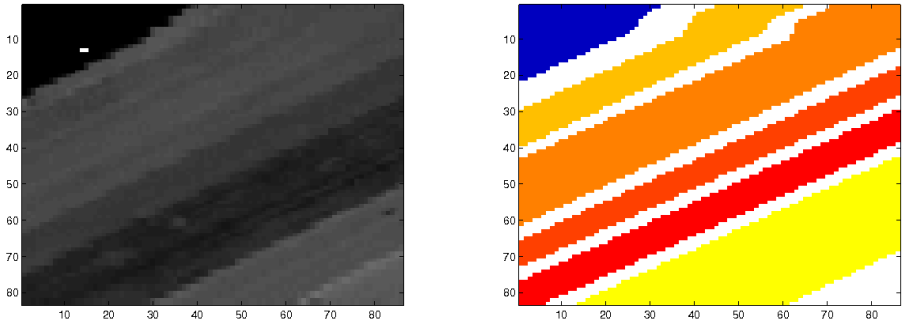


(a) Sample band of Indians Pines dataset, taken from [25]

(b) Indian Pines Ground truth, taken from [25]

3.2.2 Salinas A

The Selinas A data set is a subset of the Salinas scene. It was collected by a 224 band AVIRIS sensor. The scene contains different vegetables soil and vineyards. The spatial dimensions are 86x83 and consists of six classes figure 3.3a and 3.3b [25].



(a) Sample band of Salinas A dataset, taken from [25]

(b) Indian Pines Ground truth, taken from [25]

3.2.3 HICO dataset

Taken from [26]. This dataset is gathered from the HICO sensor mounted on the International Space Station. It consists of 128 different bands ranging from 352 to 1080 nm. It consists of 512 x 2000 pixels. The data was self labeled semi supervised and consists of three classes (land, water and clouds). The bands in the range 1-9 and 87-128 were removed as they are considered less accurate. As such wavelengths between 404-896nm are present in the data. The dimension has also been reduced to 506 x 1996 because the sensors viewing slit was visible in parts of the image, figure 3.4 [27].

3.2.4 HYPSO-1 dataset

This dataset is gathered from the HYPSO-1 satellite, figure 3.5. It consists of 956 x 684 pixels and 120 spectral bands. The data was self labeled semi supervised

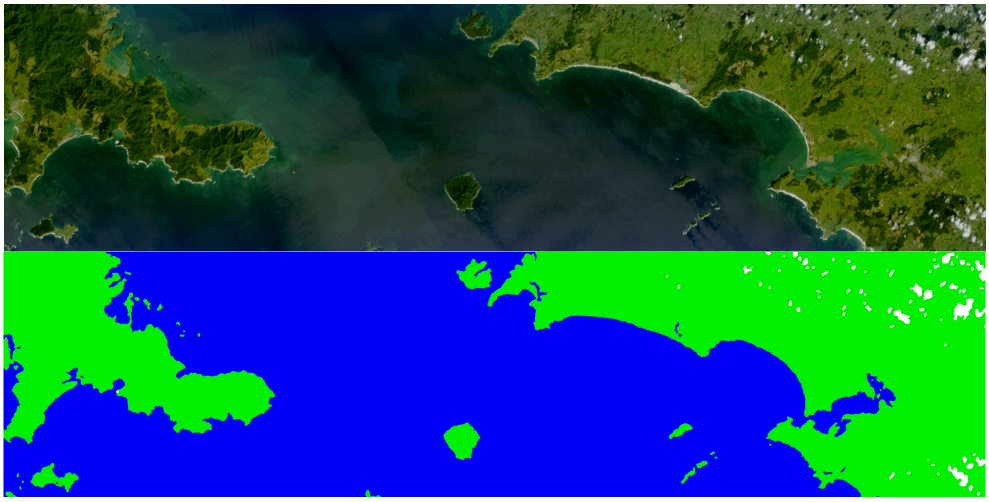


Figure 3.4: Comparison of quasi true and generated ground truth of HICO image H2011216003423, taken from [13]

and consists of three classes (land, water and clouds). The data is raw from the satellite without radiometric calibration. The scene contains the Norwegian lake Mjosa.

3.3 Building and training models

The models were built and trained using the machine-learning framework Pytorch. In addition to the PytorchGeometric library which integrates support for GNNs. This section will talk about how the models were trained and constructed as well as the training data construction and augmentation.

3.3.1 Training and test data

The training data consists of a subset of the whole dataset. The training data was built by extracting spatial lines along the x axis from the datasets.

The training data could then be divided further into smaller subsets by utilising a self built function make_training_dataset() the function allows to specify the length of the subset and the amount of overlapping pixels between each sub-

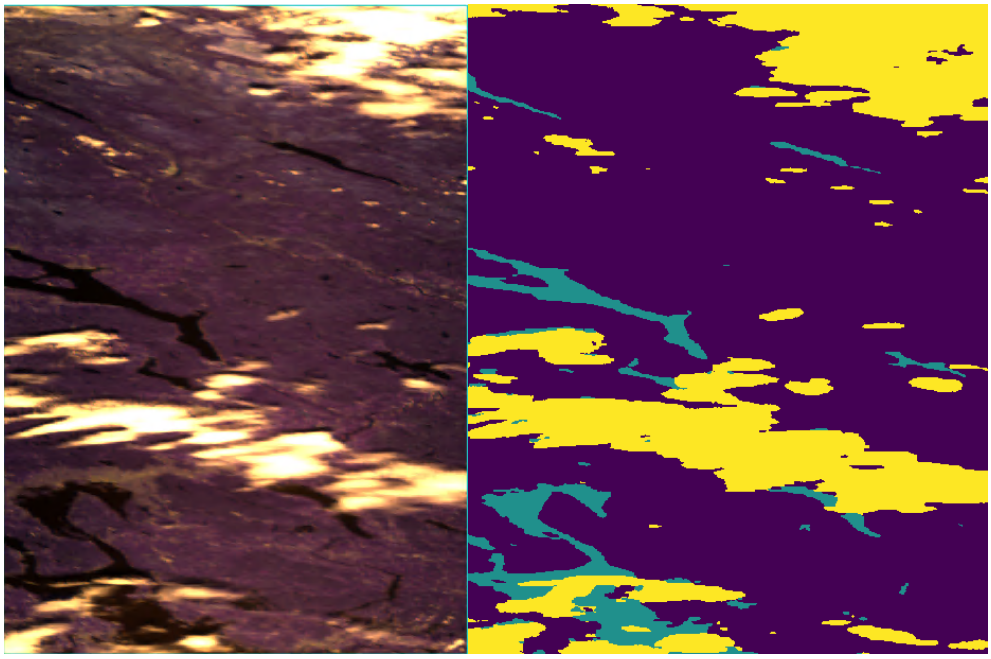


Figure 3.5: Comparison of RGB composit and ground truth of HYPSO-1 image 20220623_CaptureDL_00_mjosaT09_42

set. Finally the edge connections between the nodes were were established. The amount of edge connections between pixels could also be specified.

3.3.2 Pre-processing

Principal component analysis (PCA) [28] was used as preprocessing to lower the amount of image bands and increase the accuracy of the model prediction.

3.3.3 Graph convolutional layers

The first part of the models are the Graph Convolutional Layers. The trained modules utilize a Graph Convolutional Network (GCN). The mathematical definition can be seen in 3.1, which is taken from [29].

$$x_i^{(k)} = \sum_{j \in N(i) \cup \{i\}} \frac{1}{\sqrt{deg(i)} \cdot \sqrt{deg(j)}} \cdot (W^T \cdot x_j^{k-1}) + b \tag{3.1}$$

i and j represent neighbouring node features. Here W is the weight matrix, which is used for transforming the node features. The features are then normalised by their degree (the degree of a node is the number of edges it has to other nodes in the network. b is the bias vector.

3.3.4 Approaches

During the work on this thesis many different network structures and data augmentations were tried. This subsection will go through the different failed approaches that were tried.

Deep GCN

Regular convolutional networks receives great boosts in accuracy when employing deeper networks. As such experimenting with deeper graph convolutional networks were attempted. Here stacking five to six convolutional layers with descending spectral resolution was attempted. The results from stacking multiple GCN performed worse than when using only a few layers. This is in line with [29] the findings in the initial paper. Implementing batch normalisation between the layers improved performance, but the deep GCN still performed subpar in comparison to a more shallow version.

Residual Network

Another successful network structure is the ResNet and its variants [30]. One of the main challenges when training very deep networks is the vanishing gradient problem [31]. By utilising skip connections the network combats the vanishing gradient problem by passing information through the layers. In figure 3.6 we see the original resnet block implementation and in figure 3.7 we can see the corresponding GCN implementation. Expriments with residual blocks did not lead to any improvement in network performance an did thus not find its way into the final implementation.

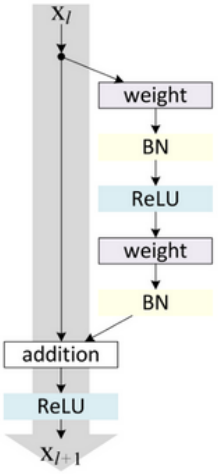


Figure 3.6: ResNet residual block, taken from [30]

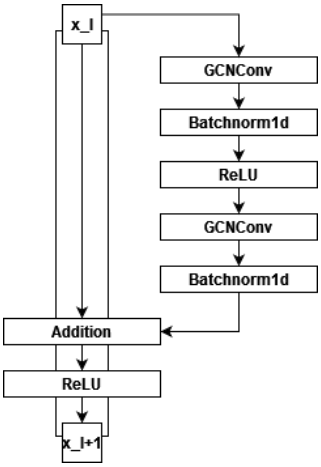


Figure 3.7: GCN Resnet implementation

3.3.5 Data Augmentation

When training data is limited, data augmentation is an effective method for enlarging training data in sparse datasets. Since training data is extracted as a subsample lines of the whole image cube augmentation methods are more limited than in classical RGB images. Augmentation attempts was thus limited to two of the more standard augmentation methods flipping and noise.

Flipping was implemented by flipping the data around the y axis ref. Gaussian noise was added randomly to each pixel.

Neither of the augmentation methods provided any improved results for the model. This was somewhat unexpected because basic image transforms usually yield better results for basic image classification tasks. In the case of using GCN this most likely stems from the fact that the GCN works in non-euclidean space and thus is not affected by spatial rotations.

3.3.6 Final implementation

The final model can be seen in figure 3.8. The input dimensions are a HSI line with dimensions **MxN**. The input first pases through a Factor analysis which reduces

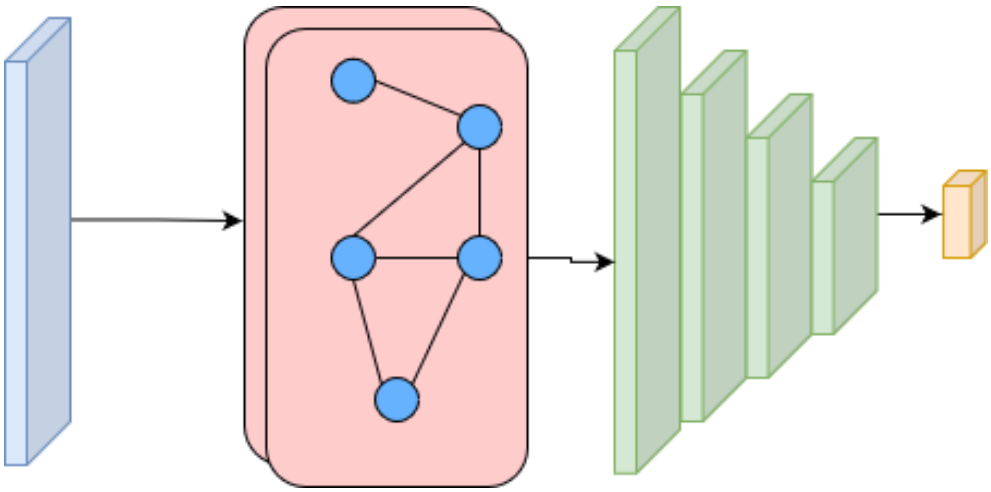


Figure 3.8: The final implemented model

the amount of bands from **N** to **B**. The output is then fed to a series of two graph convoloutional layers with batch normalisation [32], ReLU activation and dropout layers. No dimensional reduction is performed by the graph convolutional layers. The output is then applied to a series of linear layers with batch normalisation and ReLU activation. The output layer is linear with a logarithmic softmax activation. The Adam optimiser is used as the model optimiser.

Chapter 4

Results

This chapter presents how the model was trained, tunable parameters, the final training results and the factors which impacted the results the most. A model prediction as well as confusion matrices are presented for each dataset. The chapter is split in two main sections between the two standard datasets Indian Pines and SalinasAm, and the self labeled HICO and HYPSO-1 datasets. The reason for this being the large difference in training data and class complexity.

4.1 Training

This section presents how the final model was trained and the tunable training parameters. The parameters were tuned to achieve the maximal accuracy. All training of the model was done with 30% training data and 70% test data.

4.1.1 Tunable model and data parameters

This subsection lists the final model and data parameters that were tuned during training.

Model parameters

- Learning rate: **0.01**

- Weight decay: **5e-4**
- Dropout ratio: **0.5**
- Batch size: **15**
- Momentum: **0.9**

Data parameters

- Shuffle data when training: **Yes**
- Amount of pixels in input lines: **30**
- Max distance between edge connected pixels: **2**
- Pixel overlap between input lines: **2**

4.2 Indina Pines and SalinasA

The spectral resolution of the images were down scaled to 100 bands through the pre-processing PCA before being sent through the model. The model was trained for 400 epochs on each image. The model was trained 10 times for each dataset to get the average accuracy and standard deviation. The models were trained multiple times to assess which model features affected the model the most. The final results can be seen in table 4.1.

	SalinasA	**Indian pines**
	Average accuracy	Average accuracy
Final model	89.04%+/-0.66	75.64%+/-0.72
Only one edge between nodes	88.58%+/-0.37	72.61%+/-1.94
No dropout layers	87.19%+/- 1.03	71.23%+/- 0.67
No batch normalisation	88.32%+/- 0.57	75.6%+/- 0.45
Learning rate of 0.01	89.08%+/- 0.47	73.14%+/- 0.20
No graph convolutional layers	86.00%+/- 0.37	73.05%+/- 0.88

Table 4.1: Testing results from SalinasA and Indian pines datasets

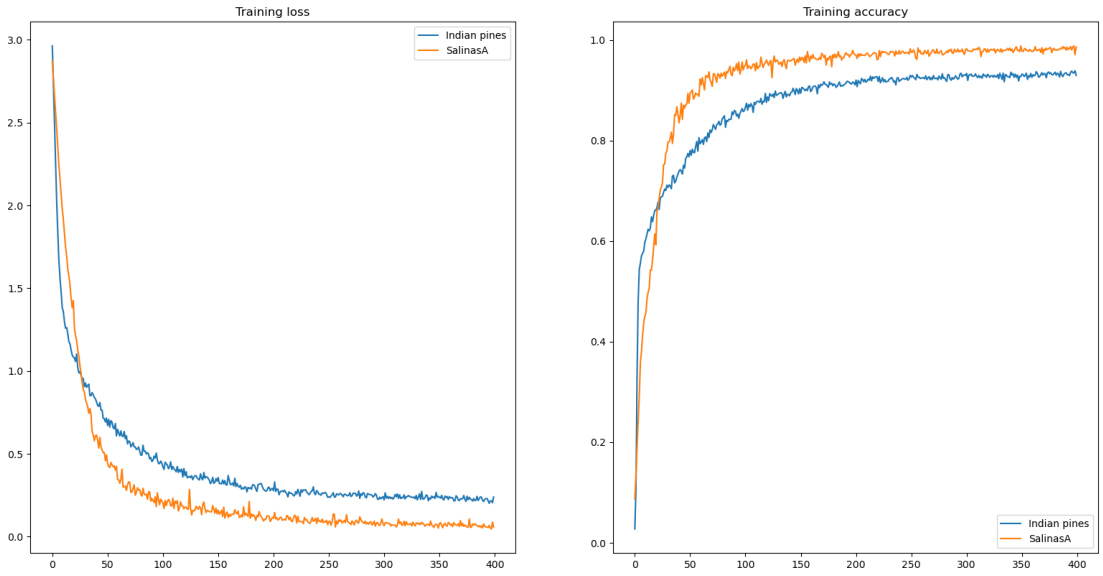


Figure 4.1: Training loss and accuracy, SalinasA and Indian pines

The final model has an average accuracy of 89.04% on the Salinas dataset and 75.64% on the Indian pines dataset. The model predicted images can be seen in figure 4.2 and 4.3. Figure 4.4 and 4.7 shows sample confuse matrices from the testing the model. The SalinasA matrix goes from 0 to 16 because it is a subset of a larger dataset with 16 classes.

When predicting the Indian pines dataset the confusion matrix shows no correct guesses for class 7 and 9, it should here be noted that these classes have under 10 samples each.

4.3 HICO and HYPSO-1

Testing was done both with and without pre-processing with PCA. The model was trained for only 10 epochs on each image because of the large datasize. The model war trained 10 times for each image to get average accuracy and standardeviation.

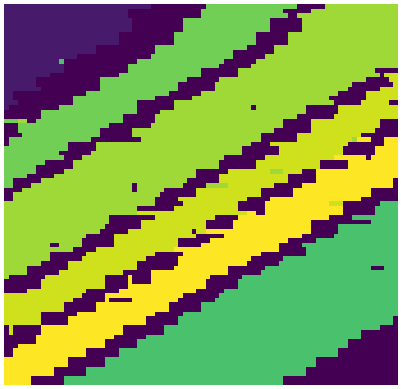


Figure 4.2: SalinasA model prediction

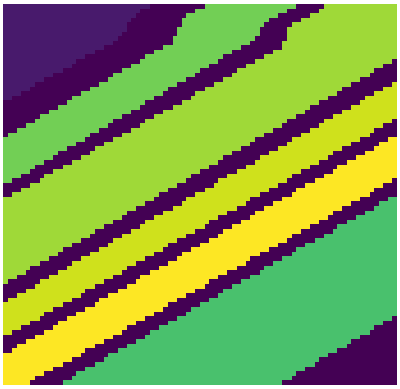


Figure 4.3: SalinasA model prediction

	Mjosa	**HICO**
	Average accuracy	Average accuracy
Final model	97.87%+/- 0.65	99.37%+/- 0.08
No PCA	95.0%+/- 0.14	98.97%+/- 0.003

Table 4.2: Testing results from the HICO and HYPSO-1 dataset

The final results can be seen in table 4.2.

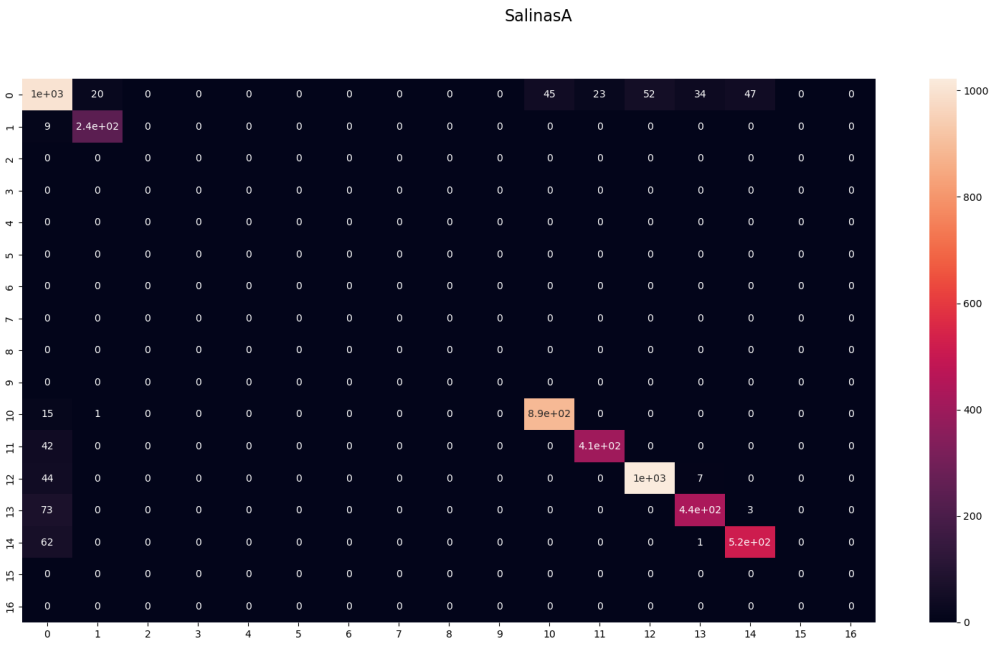


Figure 4.4: Confuse matrix SalinasA

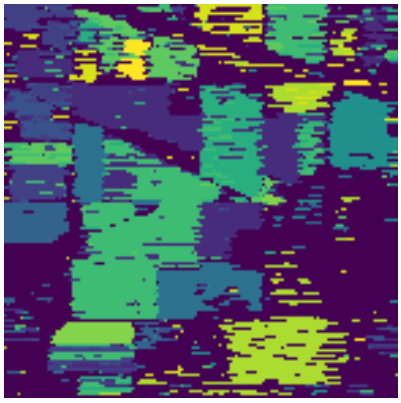


Figure 4.5: Indian pines prediction

Figure 4.6: Indian pines ground truth

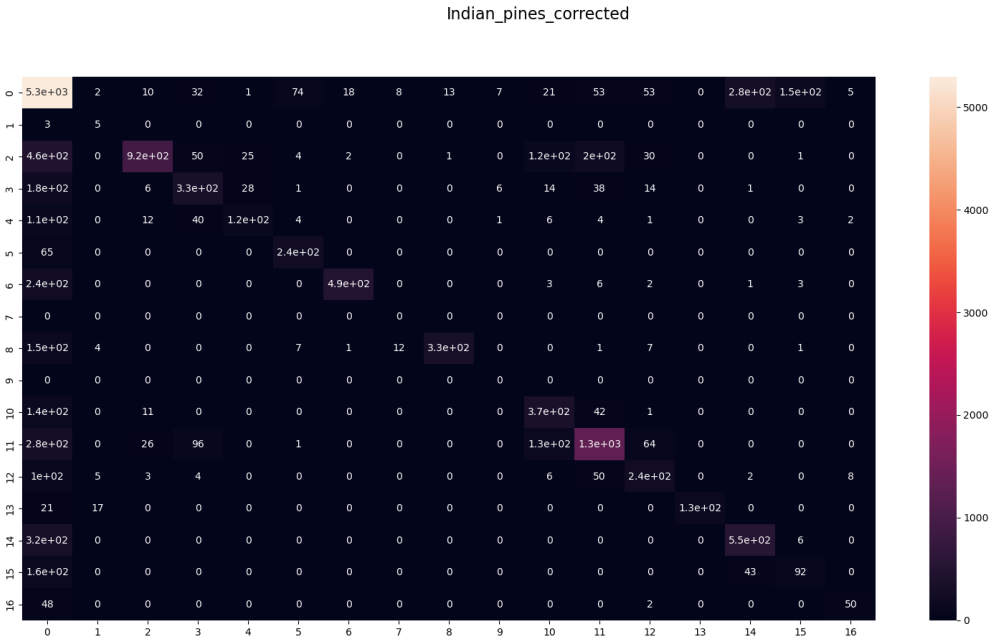


Figure 4.7: Confuse matrix Indian Pines prediction

Figure 4.8: HICO prediction

Figure 4.9: HICO ground truth

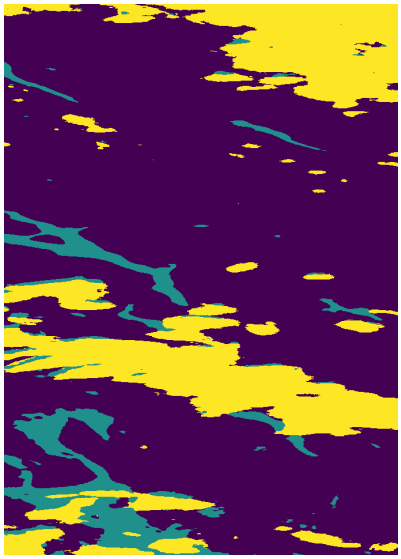


Figure 4.10: Mjosa prediction

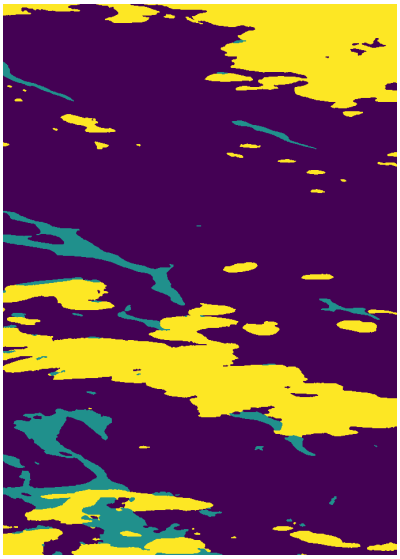


Figure 4.11: Mjosa ground truth

Chapter 5

Discussion

This section will focus on discussion of the model performance and suitability for increasing operational autonomy of the HYPSO-1 satellite. The chapter is divided between the two standard datasets Indian Pines and SalinasA and the two satellite datasets HYPSO-1 and HICO.

5.1 Model performance on Indian Pines and SalinasA dataset

State of the art models predicting on the Indian Pines and SalinasA datasets can reach high 90%s accuracy's [33]. Compared to the models performance on the Indian Pines dataset with a top accuracy of 75.6% it is clear that the model cannot compete with state of the art. This was expected as the model can only utilize spatial information along a single axis. The use of GCN has also lead to an overall less complex model than what would be implemented with a conventional CNN. As such the model will be evaluated based on general its standalone performance and suitability for its intended use instead of a direct comparison with state-of-the art models.

The performance on the SalinasA dataset is good with 89.08% average accuracy while the accuracy on the Indian Pines set is on the weaker side with 75.64%

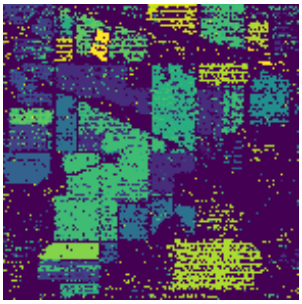


Figure 5.1: Indian Pines prediction with no GCN layers

average accuracy. The Indian Pines dataset is more complex with 16 vs 10 classes and the classes in the SalinasA dataset is more continuous than in the Indian Pines.

From the testing results we can see how the learning-rate impacts the training on each dataset. With a learning rate of 0.01 the Indian Pines average accuracy drops by 2.5% while the SalinasA performance actualy increases by 0.04%. The reason for this most likely comes from the mentioned complexity of the Indian Pines dataset. As such the model will benefit from a lower learning rate for being able to fine-tune the difference between them. This low learning rate is a double-edged sword. On the one hand you will better be able to differentiate between classes on the other hand it can lead to over-fitting the model. This is evident from the results were dropout layers were removed, here the Indian Pines average accuracy fell by 4.41% and the SalinasA average accuracy fell by 1.85%. Here the Indian Pines accuracy was actually worse than without any GCN layers.

5.1.1 Possible model improvements

From the predicted image on the Indian Pines dataset in figure 4.5 there are very distinct horizontal lines of misclassification. These lines can also be found in the SalinasA prediction 4.2, but they are not as prevalent.

In figure 5.1 the model has been trained without the GCN layers. Here the prediction is more noisy with misclassfied pixels scattered around the image. On the positive this means that the final model is actually using the edge connections and correlating samples along the inputted spatial line. On the flip-side the drawback

is that the model is also causing multiple wrong classifications in a row.

Interestingly it can be observed that the majority of misclassified lines are one to two pixels in width along the vertical axis. This suggests that if only an extra one to two lines of spatial information is introduced to the model when predicting, it can potentially give strong improvements to the model prediction.

The first potential improvement is inputting another 1-2 adjacent lines with additional edge connections. This is a relatively simple way to adjust the model and could potentially give very good improvements to accuracy. While the model would no longer scan line by line it could still be usable for near real-time classification.

Using a Recurrent neural-network layer is another possible way to introduce more information for model prediction. This would work by introducing more information over the time domain, rather than in the spatial domain. When predicting a line such a model would take into account the previously predicted pixels. This approach would allow the model to still only process a single line at a time. The drawback of using RNN is that the added complexity of the implementation and processing speed which could make it less suitable for directly porting to FPGA.

5.2 Model performance on HICO and HYPSO-1 dataset

While testing models on standard datasets such as Indian Pines and SalinasA is a good way for benchmarking and comparing state-of-the art models, it is more interesting to see how the model performs on actual satellite data. The self labeled data differs from the standard datasets in three major ways:

- Over 30 times as many pixels in each dataset
- Only three classes
- No class 0 in the datasets

Additionally the HYPSO-1 dataset has no radiometric calibration and is raw

data from the HSI sensor. The results from the model is thus directly transferable to how the model would perform in-orbit.

As can be seen from table 4.2 the test results on the satellite data the results are close to 100%. Noticeably the results from the HYPSO-1 dataset is a few % lower than the HICO data. The most likely reasons for this is either the lack of data calibration or inaccuracy in the labelling. In the project thesis [13] the process of the semi supervised labelling of hyper spectral images is discussed. With this method much of the labelling process is left to a computer and by side by side comparison there can be seen that some pixels can be argued to belong to another class. While fully hand labelling the data might give a bit better results this is not feasible as hand labelling such a large amount of data is extremely time consuming and could take days.

When comparing the predicted image with the RGB representation it can be argued that the trained model is actually has a better classification of the image than the generated groundtruth. Were some smaller bodies of water not present in the groundtruth are present in the predicted image.

In comparison with the prediction of the Indian Pines dataset the predicted HYPSO-1 and HICO images are much smoother and does not contain the mis-classified horizontal lines. Fewer classes might be the reason for this as a similar trend could be seen with the SalinasA prediction. It should also be accounted for the high amount of training data which allows for more accurate predictions.

5.3 Practical model usage

Currently the amount of hyperspectral images that can be acquired from the HYPSO-1 satellite is limited by the downlink capacity. The HYPSO-1 uses image compression which compresses the image cubes down to approximately 80Mb each. This compression allows for the downlink of 5 6 image cubes per day [34]. As such the ability to ascertain and optimise the quality of image captures before downlinking is necessary to get the most out of the limited downlink capacity.

From the results it is clear that classifying line by line for near real-time classification is feasible for segmentation tasks. The predicted images provide smooth class segmentation's which makes it easy for a human operator or a computer to check the initial features of the image, either by the amount of cloud cover or the amount of water bodies present in the capture.

5.3.1 Segmentation for downlink decision making

Giving a preview of the HSI in the form of a basic three class segmentation as in this thesis is in it self not inherently useful, since the operator can downlink an RGB composite of the capture in a similar time frame as a segmented preview. This basic segmentation task is more interesting as a tool for autonomous downlink decisions by the satellite. The segmentation task gives an accurate estimations of the ratio between clouds, water and land in the image. Class thresholds can thus be set by the operator for which images should be discarded and which should be prioritised for downlink. An image full of clouds or an image with no major bodies of water will be less interesting for the detection of algae.

The unpublished paper [34] proposes using the rate of cube compression to estimate if the capture contains large amounts of clouds or over/under exposure. The weakness of this approach is that it can only give an estimate of the general image quality and can not give the satellite or operator any insight in what information the capture actually contains. Utilising on-board segmentation can thus give an even better estimate of the image quality than the compression rate, but at the cost of some extra on-board processing.

While the above mentioned methods can only serve to de-prioritise the downlink of certain captures. By training the model to recognise algae or more complex/segmented waters such images could be prioritised for downlink and processing on the ground.

Currently there exists no additional labelled HYPSO-1 data outside the one presented in this thesis, new data must thus be labelled for the model to recognise

such features. While this will be a more advanced classification task, the results on the Indian Pines and SalinasA datasets indicates that this should be within the models abilities with sufficient data-grounds.

5.3.2 Real time segmentation for capture initiating

The near real-time classification functionality of the model allows for further exploiting the advantages outlined in the previous section. The current performance of the model allows for extended satellite autonomy during image captures. If cloud cover is present over the capture target area. The satellite can attempt to avoid capturing the clouds by not initiating capture before a desired amount of land or water is in view of the satellite.

If new training data enables the detection of algae waters the satellite can continuously scan the environment below, either initiating capture maneuver as algae are detected or alert the ground operators of the coordinates of the detected algae.

Chapter 6

Conclusion

This thesis serves as a starting point for an on-board implementation of a near real-time classification algorithm for hypespectral images for the HYPSO-1 satellite.

A baseline algorithm has been developed that was able to achieve an average accuracy of 95.0% on a self labelled HYPSO-1 HSI dataset with three classes. The dataset consisted of raw HSI data directly from the satellite with no calibration or pre-processing. The results on standard datasets such as Indian pines and Salinas A were 75.64% and 89.04% respectively. The model is thus not suitable to compete with state-of-the-art models, but its simple structure and support for near real-time classification show promise for rapid on-board deployment and in-orbit decision making that can improve captures and allow for higher automisation of which HSI captures that should be prioritised for downlink. Further the near real-time classification functionality of the model opens up the possibility of avoiding capturing during clouds or continuously scanning the terrain bellow for algae.

6.1 Further work

Further work will consist of taking the necessary steps to train and implement the model on-board the HYPSO-1 satellite. Work must be done to transfer the model to FPGA and how the model should best be integrated in the on-board pipeline.

More HYPSO-1 data must be labelled to give the model sufficient dataground for consistent predictions over a wide variety of captures and the possibility of predicting new classes.

The proposed model alterations in 5.1.1 should be explored in an attempt to improve the average accuracy of the model.

Bibliography

[1] P. Mhangara, 'The emerging role of cubesats for earth observation applications in south africa photogrammetric engineering remote sensing,' *Photogrammetric Engineering and Remote Sensing*, vol. 86, pp. 333–340, Jun. 2020. DOI: 10.14358/PERS.86.6.333.

[2] M. H. Azami, N. Örger, V. Schulz, T. Oshiro and M. Cho, 'Earth observation mission of a 6u cubesat with a 5-meter resolution for wildfire image classification using convolution neural network approach,' *Remote Sensing*, vol. 14, p. 1874, Apr. 2022. DOI: 10.3390/rs14081874.

[3] J. Linder, 'Eksitere (fysikk),' Aug. 2020. [Online]. Available: https://snl.no/eksitere_-_fysikk.

[4] S. D. S. Survey, 'Energy levels of electrons,' [Online]. Available: https://skyserver.sdss.org/dr1/en/proj/advanced/spectraltypes/energylevels.asp.

[5] 'Hydrogen excitation.' (Jun. 2022), [Online]. Available: https://commons.wikimedia.org/wiki/File:Atom_excitation.jpg.

[6] I. Wikipedia, 'Planck relation,' Jan. 2022. [Online]. Available: https://en.wikipedia.org/wiki/Planck_relation.

[7] D. A. Smale, 'Spectra and what they can tell us,' Aug. 2013. [Online]. Available: https://imagine.gsfc.nasa.gov/science/toolbox/spectra1.html.

[8] J. Skaar, 'Lys,' Nov. 2020. [Online]. Available: https://snl.no/lys.

[9] R. K. Jostein and H. Trygve, 'Lys,' Dec. 2020. [Online]. Available: https://snl.no/lys.

[10] 'Visible ligth spectrum.' (Jun. 2022), [Online]. Available: http://ch301.cm.utexas.edu/atomic/#H-atom/line-spectra.html.

[11] 'Hydrogen absorption spectrum.' (Jun. 2022), [Online]. Available: http://ch301.cm.utexas.edu/atomic/#H-atom/line-spectra.html.

[12] 'Hydrogen emission spectrum.' (Jun. 2022), [Online]. Available: http://ch301.cm.utexas.edu/atomic/#H-atom/line-spectra.html.

[13] T. Bratvold, 'Exploration of machine learning techniques for classification in hypespectral images,' Dec. 2021.

[14] 'Rgb cube.' (Dec. 2021), [Online]. Available: https://www.researchgate.net/figure/Data-cube-representation-of-the-RGB-components-of-a-digital-image_fig2_252986131.

[15] 'Hsi data cube.' (Dec. 2021), [Online]. Available: https://www.esa.int/ESA_Multimedia/Images/2014/04/Hyperspectral_image_data_cube.

[16] H. J. Kramer, 'Hypso (hyperspectral smallsat for ocean observation),' Jul. 2022. [Online]. Available: https://directory.eoportal.org/web/eoportal/satellite-missions/h/hypso.

[17] 'Internal structure of xilinx fpga.' (Jun. 2022), [Online]. Available: https://www.researchgate.net/figure/1-Internal-structure-of-Xilinx-FPGA-3_fig1_290929451.

[18] 'Image segmentation.' (Jun. 2022), [Online]. Available: https://medium.com/syncedreview/facebook-pointrend-rendering-image-segmentation-f3936d50e7f1.

[19] 'Salinas dataset.' (Jul. 2022), [Online]. Available: https://www.researchgate.net/figure/Salinas-dataset-a-false-color-composite-image-bands-29-19-and-9-b-ground-truth_fig7_330747101.

[20] A. Menzli, 'Graph neural network and some of gnn applications: Everything you need to know,' Jul. 2022. [Online]. Available: https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications.

[21] 'Vitamin-a.' (Jul. 2022), [Online]. Available: https://chem.libretexts.org/Bookshelves/Organic_Chemistry/Supplemental_Modules_(Organic_Chemistry)/Fundamentals/Structure_of_Organic_Molecules.

[22] A. Filipovic, 'Introduction to node embedding,' Dec. 2021. [Online]. Available: https://memgraph.com/blog/introduction-to-node-embedding.

[23] S. Stanford, 'Node representation learning,' Jul. 2022. [Online]. Available: https://snap-stanford.github.io/cs224w-notes/machine-learning-with-networks/node-representation-learning.

[24] I. Mayachita, 'Understanding graph convolutional networks for node classification,' Jun. 2020. [Online]. Available: https://towardsdatascience.com/understanding-graph-convolutional-networks-for-node-classification-a2bfdb7aba7b.

[25] 'Hyperspectral remote sensing scenes.' (), [Online]. Available: https://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes.

[26] G. C. Feldman. 'Sensor and data characteristics.' (Dec. 2021), [Online]. Available: https://oceancolor.gsfc.nasa.gov/hico/instrument/dataset-characteristics/.

[27] G. C. Feldman. 'Spectral angle mapper.' (Dec. 2021), [Online]. Available: https://oceandata.sci.gsfc.nasa.gov/directaccess/HICO/L1/.

[28] I. Wikipedia, 'Principal component analysis,' Jul. 2022. [Online]. Available: https://en.wikipedia.org/wiki/Principal_component_analysis.

[29] T. N. Kipf and M. Welling, 'Semi-supervised classification with graph convolutional networks,' Feb. 2017. [Online]. Available: https://arxiv.org/pdf/1609.02907.pdf.

[30] V. Feng, 'An overview of resnet and its variants,' Jul. 2017. [Online]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035.

[31] C.-F. Wang, 'The vanishing gradient problem,' Jan. 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[32] J. Brownlee, 'A gentle introduction to batch normalization for deep neural networks,' Dec. 2019. [Online]. Available: https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.

[33] R. Stojnic. 'Paperswithcode.' (Dec. 2021), [Online]. Available: https://paperswithcode.com/task/hyperspectral-image-classification.

[34] B. S. Birkeland R. and G. J. L, 'Prioritizing hyperspectral image downlink by compression ratio,' Jul. 2022.

Appendix A

Model code

A.1 main.py

```
from cProfile import label
from operator import mod
from select import select
from statistics import mode

from this import d
import time
from scipy import rand
import torch
import scipy.io
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from torch import batch_norm, double, nn
from torch.utils.data import TensorDataset
from torch_geometric.data import Data
from torch.nn import BatchNorm1d
import torch.nn.functional as F
from torch_geometric.loader import DataLoader, DenseDataLoader
from torch_geometric.nn import GCNConv, ChebConv, GraphNorm, MessageNorm,
from torch_geometric.data import InMemoryDataset, download_url
from tqdm import tqdm
import random
```

```
from sklearn.decomposition import FactorAnalysis
from sklearn.decomposition import PCA
from tqdm import tqdm
from models import DGC, GCN

import torchvision
from confusion import *
from train import *

class AddGaussianNoise(object):
    def __init__(self, mean=0., std=1.):
        self.std = std
        self.mean = mean

    def __call__(self, tensor):
        return tensor + torch.randn(tensor.size()) * self.std + self.mean

    def __repr__(self):
        return self.__class__.__name__ + '(mean={0}, std={1})'.format(self.mean, self.std)


def applyPCA(X, numComponents=100):
     newX = np.reshape(X, (-1, X.shape[2]))
     pca = PCA(n_components=numComponents, whiten=True)
     newX = pca.fit_transform(newX)
     newX = np.reshape(newX, (X.shape[0],X.shape[1], numComponents))
     return newX, pca


dataset_list = [ "H2011216003423_3"]

loss_list_salinas = []
accuracy_list_salinas = []
val_loss_list_salinas = []
val_acc_list_salinas = []
final_salinas_accuracy = 0

loss_list_indian = []
accuracy_list_indian = []
val_loss_list_indian = []
```

```
val_acc_list_indian = []
final_indian_accuracy = 0

for dataset in dataset_list:
    std_list = []
    print("Making dataset")

    data, targets = load_dataset(dataset)



    class AddGaussianNoise(object):
        def __init__(self, mean=0, std=1.):
            self.std = std
            self.mean = mean

        def __call__(self, tensor):
            return tensor + torch.randn(tensor.size()) * self.std + self.mean

        def __repr__(self):
            return self.__class__.__name__ + '(mean={12.69}, std={1})'.format(self.mean, self.std)




    #data, pca = applyPCA(data)

    print(np.amax(data))
    print(np.amin(data))

    print(data.shape)



    transform = torchvision.transforms.Compose([AddGaussianNoise()])

    augment = False

    train_data, train_targets, val_data, val_targets = split_training_data(data, targets, 0.3)
```

```
train_data_fliped = np.flip(train_data, 1)
train_targets_fliped = np.flip(train_targets, 1)

if augment:
    train_data = torch.FloatTensor(train_data)

    train_data_augmented = transform(train_data)

    train_data = torch.cat((train_data, train_data_augmented))
    train_data = train_data.numpy()

    train_targets = np.concatenate((train_targets, train_targets))




"""train_data_avg = avrage_pixels(train_data, train_targets)
train_data = np.concatenate((train_data, train_data_avg))
train_targets = np.concatenate((train_targets, train_targets))"""



train_data = np.concatenate((train_data, train_data_fliped))
train_targets = np.concatenate((train_targets, train_targets_fliped))

train_data = make_training_data(train_data, train_targets, 30, 2)






val_data = make_test_data(val_data, val_targets)

train_dataloader = DataLoader(train_data, batch_size=15, shuffle=True)

"""train_dataloader.transform = torchvision.transforms.Compose([
    AddGaussianNoise(0., 1.)])"""

val_dataloader = DataLoader(val_data)
#plt.imshow(targets)
```

```
    #plt.show()

    print("Initiating training")
    #dataloader_iterator = iter(train_dataloader)

    #data, target = next(dataloader_iterator)

    #convert to graph
    #loader = DataLoader(datalist, shuffle=False)



    m = 0.8




    number_of_loops = 10
    temp = 0

    loss_list = []

    for loop in range(number_of_loops):
        ################################
        #Training section
        model_name = "dc"

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

        if model_name == "dgc":
            model = DGC().to(device)
        else:
            model = GCN().to(device)
            #lr = 0.0005
        optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=5e-4)
        loss = 100
        best_loss = 100
        times_no_new_best_loss = 0



        number_of_epochs = 25
```

```
for i in tqdm(range(number_of_epochs)):
    avg_train_loss, avg_train_accuracy = train_loop(train_dataloader, model, optimizer, device)
    #val_loss, val_accuracy = val_loop(val_dataloader, model, device)

    avg_train_loss = avg_train_loss.data.cpu().numpy()
    if dataset == "Indian_pines_corrected":
        loss_list_indian.append(avg_train_loss)
        accuracy_list_indian.append(avg_train_accuracy)
        #val_loss_list_indian.append(val_loss)
        #val_acc_list_indian.append(val_accuracy)
    elif dataset == "SalinasA":
        loss_list_salinas.append(avg_train_loss)
        accuracy_list_salinas.append(avg_train_accuracy)
        #val_loss_list_salinas.append(val_loss)
        #val_acc_list_salinas.append(val_accuracy)
    #print(avg_loss)

    if avg_train_loss < best_loss:
        times_no_new_best_loss = 0
        best_loss = avg_train_loss
        #print("Saving model")
        torch.save(model.state_dict(), "best_model")
    else:
        times_no_new_best_loss += 1
    #if times_no_new_best_loss == 60:
    #   break
################################
print(best_loss)
"""plt.plot(loss_list)
plt.show()
loss_list = []"""
if model_name == "dgc":
    model = DGC().to(device)
else:
    model = GCN().to(device)


accuracy, predicted_image = test_loop(val_dataloader, model, device)
class_numbers = ['1' ,'2', '3']
confusion_matrix = get_confusion_matrix(predicted_image, val_targets, class_numbers)
```

```
    plot_confusion_matrix(confusion_matrix, dataset)
    std_list.append(accuracy)
    temp += accuracy

avg_accuracy = temp/number_of_loops

print("Avg test accuracy: ", avg_accuracy)
print("Standard deviation: ", np.std(std_list))
#predicted_image = predicted_image.cpu()

if dataset == "Indian_pines_corrected":
    final_indian_accuracy = accuracy
elif dataset == "SalinasA":
    final_salinas_accuracy = accuracy



test_data = make_test_data(data, targets)



test_dataloader = DataLoader(test_data)

accuracy, predicted_image = test_loop(test_dataloader, model, device)


#f, axarr = plt.subplots(1,2)
colors = ['green', 'blue']


cmap = mpl.colors.ListedColormap(colors)
#plt.imshow(predicted_image, interpolation='none', cmap=cmap)
plt.imsave(dataset + '_prediction.png', predicted_image)

plt.imsave(dataset + '_ground_truth.png', targets)
"""axarr[0].imshow(predicted_image)
axarr[1].imshow(targets)"""

#plt.show()
```

```
f, axarr = plt.subplots(1,2)
axarr[0].plot(loss_list_indian, label='Indian pines')
axarr[0].plot(loss_list_salinas, label='SalinasA')
axarr[0].legend(loc='upper right')
axarr[0].title.set_text('Training loss')
axarr[1].plot(accuracy_list_indian, label='Indian pines')
axarr[1].plot(accuracy_list_salinas, label='SalinasA')

axarr[1].legend(loc='lower right')
axarr[1].title.set_text('Training accuracy')

plt.show()
```

A.2 train.py

```
from audioop import avg
from select import select
from this import d
from tkinter import N

from matplotlib import transforms
from scipy import rand
import torch
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
from torch import batch_norm, double, nn
from torch.utils.data import TensorDataset
from torch_geometric.data import Data
import torch.nn.functional as F
from torch_geometric.nn import GCNConv, BatchNorm, ChebConv, TransformerConv, AGNNConv, TAGConv
from torch_geometric.data import InMemoryDataset, download_url
from tqdm import tqdm

import random
```

```
def load_dataset(name):
    datapath = 'data/' + name + '.mat'
    ground_truth_path = 'data/' + name + '_gt.mat'
    raw_data = scipy.io.loadmat(datapath)


    if name == "Indian_pines_corrected":

        #name = name.lower()
        ground_truth = scipy.io.loadmat("data/Indian_pines_gt.mat")
        dataset = raw_data["indian_pines_corrected"] # use the key for data here
        gt_key = name + '_gt'

        target = ground_truth["indian_pines_gt"] # use the key for target here

        dataset = dataset.astype(int)
        target = target.astype(int)


    elif name == "H2011216003423_3":
        ground_truth = scipy.io.loadmat("data/H2011216003423_3_class.mat")
        dataset = raw_data["H2011216003423"] # use the key for data here
        gt_key = name + '_gt'

        target = ground_truth["H2011216003423_class"] # use the key for target here

        dataset = dataset.astype(int)
        target = target.astype(int).reshape(1996, 506)

        print(target.shape)

    elif name == "mjosa":
        ground_truth = scipy.io.loadmat("data/mjosa_class.mat")
        dataset = raw_data["mjosa"] # use the key for data here
        gt_key = name + '_gt'

        target = ground_truth["mjosa_class"] # use the key for target here
```

```
        dataset = dataset.astype(int)
        print(target.shape)
        target = target.astype(int).reshape(956, 684)

    else:
        #name = name.lower()
        ground_truth = scipy.io.loadmat("data/SalinasA_gt.mat")
        dataset = raw_data["salinasA"] # use the key for data here
        gt_key = name + '_gt'
        target = ground_truth["salinasA_gt"] # use the key for target here

        dataset = dataset.astype(int)
        target = target.astype(int)
        print(target.shape)

    return dataset, target


def avrage_pixels(train_data, train_target):
    for line_number in range(len(train_data)):
        for i in range(len(train_data[line_number])-1):
            pixel_1_t = train_target[line_number][i]
            pixel_2_t = train_target[line_number][i+1]
            if pixel_1_t == pixel_2_t:
                avg_pixel = (train_data[line_number][i] + train_data[line_number][i+1])/2
                train_data[line_number][i] = avg_pixel
                train_data[line_number][i+1] = avg_pixel
                i += 1
    return train_data

def split_training_data(data, targets, training_size, number_of_connected_lines=0):
    number_of_training_samples = np.round(training_size*len(data)).astype(int)
    train_indexes = np.round(np.linspace(0, len(data) - 1, number_of_training_samples)).astype(int)

    if number_of_connected_lines > 0:


        print(train_indexes)

    train_data = []
```

```
    train_targets = []
    val_data = []
    val_targets  = []

    print(train_indexes)

    for index in range(len(data)):
        if index in train_indexes:
            train_data.append(data[index])
            train_targets.append(targets[index])

        else:
            val_data.append(data[index])
            val_targets.append(targets[index])




    return train_data, train_targets, val_data, val_targets


def make_edge_indexes(number_of_nodes):
    edge_index = []


    for node in range(number_of_nodes - 1):



        edge_to_1 = [node, node+1]
        edge_back_1 = [node+1, node]
        edge_index.append(edge_to_1)
        edge_index.append(edge_back_1)
        if node +2 < number_of_nodes:
            edge_to_2 = [node, node+2]
            edge_back_2 = [node+2, node]
            edge_index.append(edge_to_2)
            edge_index.append(edge_back_2)
        """if node +3 < number_of_nodes:
            edge_to_3 = [node, node+3]
```

```
            edge_back_3 = [node+3, node]
            edge_index.append(edge_to_3)
            edge_index.append(edge_back_3)"""









    return torch.tensor(edge_index, dtype=torch.long)


def make_training_data(raw_data, target, nodes_in_data, node_overlap):

    edge_indexes = make_edge_indexes(nodes_in_data)

    datalist = []

    for i in range(len(raw_data)):

        spectra = torch.from_numpy(raw_data[i]).float()
        spectra_target = torch.from_numpy(target[i]).long()# change type to your use case

        n = 0
        break_loop  = False
        while(not break_loop):
            if n == len(spectra):
                break
            while n + nodes_in_data >= len(spectra):
                n -= 1
                break_loop = True

            mini_spectra = spectra[n:n+nodes_in_data]
            mini_spectra_target = spectra_target [n:n+nodes_in_data]
            data = Data(x=mini_spectra, edge_index=edge_indexes.t().contiguous(), y=mini_spectra_target)
            datalist.append(data)

            n += nodes_in_data
            n -= node_overlap
```

```
    return datalist

def make_test_data(raw_data, targets):
    nodes_in_data = len(raw_data[0])
    edge_indexes = make_edge_indexes(nodes_in_data)

    data_list = []

    for i in range(len(raw_data)):

        spectra = torch.from_numpy(raw_data[i]).float()
        spectra_target = torch.from_numpy(targets[i]).long()# change type to your use case

        data = Data(x=spectra, edge_index=edge_indexes.t().contiguous(), y=spectra_target)
        data_list.append(data)

    return data_list


def train_loop(data_loader, model, optimizer, device):
    model.train()
    size = len(data_loader.dataset)
    summed_loss = 0
    summed_acc = 0.0
    correct = 0
    n = 0
    for batch, data in enumerate(data_loader):
        data = data.to(device)

        #pbar.set_description("Loss: %f" % loss)
        out = model(data)
        #out = out.view(4, 100)
        #print(out.size())
        #print(data.y.size())
        #print(data.y.size())
        pred = out.argmax(dim=1)

        correct = (pred == data.y).sum()

        summed_acc += int(correct) /len(data.y)
```

```
        #print("out ", out.size())
        #print("y ",data.y.size())
        loss = F.nll_loss(out, data.y)
        summed_loss += loss
        """if loss < best_loss:
            torch.save(model.state_dict(), "best_model")
            best_loss = loss"""
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        n += 1

    avg_loss = summed_loss/n
    avg_acc = summed_acc / n

    return avg_loss, avg_acc


def val_loop(data_loader, model, device):
    size = len(data_loader.dataset)


    model.eval()
    summed_loss = 0
    summed_acc = 0.0
    correct = 0
    n = 0

    for batch, data in enumerate(data_loader):
        data.to(device)
        out = model(data)
        #pred = pred.view(145, 100)
        #print(pred.size())
        #print(data.y.size())
        #dim_1 = pred.size()[0] * pred.size()[1]
        #pred = pred.view(dim_1,17)
        pred = out.argmax(dim=1)

        correct = (pred == data.y).sum()
```

```
        summed_acc += int(correct) /len(data.y)

        loss = F.nll_loss(out, data.y)
        summed_loss += loss
        n += 1

    avg_loss = summed_loss/n
    avg_acc = summed_acc / n

    return avg_loss, avg_acc









def test_loop(data_loader, model,  device):
    size = len(data_loader.dataset)

    model.load_state_dict(torch.load("best_model"))
    model.eval()
    correct = 0
    predicted_image = []
    summed_loss = 0
    n = 0

    for batch, data in tqdm(enumerate(data_loader)):
        data.to(device)
        pred = model(data)
        #pred = pred.view(145, 100)
        #print(pred.size())
        #print(data.y.size())
        #dim_1 = pred.size()[0] * pred.size()[1]
        #pred = pred.view(dim_1,17)
        loss = F.nll_loss(pred, data.y)
```

```
        summed_loss += loss
        n += 1
        pred = pred.argmax(dim=1)



        correct += (pred == data.y).sum()

        pred = pred.tolist()
        predicted_image.append(pred)

    avg_loss = summed_loss/n
    print("Loss: ", avg_loss)
    print("wrongs: ", len(data.y)*size - int(correct))
    acc = int(correct) / int((len(data.y)*size))
    print(f'Accuracy: {acc:.4f}')

    return acc, predicted_image
```

A.3 models.py

```
from operator import mod
from select import select
from statistics import mode

from this import d
import time
from scipy import rand
import torch
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
from torch import batch_norm, double, nn
from torch.utils.data import TensorDataset
from torch_geometric.data import Data
from torch.nn import BatchNorm1d
import torch.nn.functional as F
from torch_geometric.loader import DataLoader, DenseDataLoader
from torch_geometric.nn import GCNConv, ChebConv, GraphNorm, MessageNorm, TAGConv, GINConv ,GINEConv, ARMACon
```

```
from torch_geometric.data import InMemoryDataset, download_url
from tqdm import tqdm
import random
from sklearn.decomposition import FactorAnalysis
from sklearn.decomposition import PCA

import torchvision

from train import *

def make_conv_layer(inputs, outputs):
    return GCNConv(inputs, outputs)

class DGC(torch.nn.Module):
    def __init__(self):
        super().__init__()




        self.linear1 = nn.Linear(100, 80)
        self.linear1_2 = nn.Linear(40,100)
        self.linear2 = nn.Linear(80, 50)
        self.linear3 = nn.Linear(50, 25)
        self.batch_norm3_1 = BatchNorm(25, eps=1e-5, momentum=0.9)
        self.linear3_2 = nn.Linear(25, 50)
        self.batch_norm3_2 = BatchNorm(50, eps=1e-5, momentum=0.9)
        self.linear4 = nn.Linear(25, 23)
        self.linear5 = nn.Linear(23, 17)

        self.conv1 = make_conv_layer(100, 100)
        self.batch_norm1 = BatchNorm(100, eps=1e-5, momentum=0.9)
        self.conv2 = make_conv_layer(100, 50)
        self.conv2_2 = make_conv_layer(50, 100)
        self.batch_norm2 = BatchNorm(50, eps=1e-5, momentum=0.9)
        self.conv3 = make_conv_layer(100, 50)
        self.conv3_2 = make_conv_layer(50, 100)
        self.batch_norm3 = BatchNorm(50, eps=1e-5, momentum=0.9)
        self.conv4 = make_conv_layer(80, 60)

        self.msg_norm1 = MessageNorm()
```

```
        self.batch_norm3 = BatchNorm(50, eps=1e-5, momentum=0.9)
        self.batch_norm4 = BatchNorm(100, eps=1e-5, momentum=0.9)
        self.batch_norm6 = BatchNorm(40, eps=1e-5, momentum=0.9)
        self.batch_norm5 = BatchNorm(50, eps=1e-5, momentum=0.9)

    def forward(self, data):
        x, edge_index = data.x, data.edge_index

        x = self.conv1(x, edge_index)
        x = self.batch_norm1(x)
        msg = x


        #x = F.dropout(x, training=self.training)

        x = self.conv2(x, edge_index)
        x = self.batch_norm2(x)
        x = F.relu(x)
        x = self.conv2_2(x, edge_index)
        x = self.batch_norm1(x)

        x += msg
        x = F.relu(x)

        x = self.conv3(x, edge_index)
        x = self.batch_norm3(x)
        x = F.relu(x)
        x = self.conv3_2(x, edge_index)
        x = self.batch_norm1(x)

        x += msg
        #x = self.conv4(x, edge_index)
        #x = self.msg_norm1(x, msg)
        #x = self.batch_norm4(x)
        #x = F.relu(x)
        ########################################
```

```
        ########################################
        msg = x
        ###
        x = self.linear1(x)
        x = self.batch_norm6(x)
        x = F.relu(x)
        #x = self.linear1_2(x)
        #x = self.batch_norm4(x)
        ####
       # x += msg
       # x = F.relu(x)

        x = self.linear2(x)
        x = self.batch_norm5(x)
        x = F.relu(x)

        #msg = x
        ####
        x = self.linear3(x)
        x = self.batch_norm3_1(x)
        x = F.relu(x)
       # x = self.linear3_2(x)
       # x = self.batch_norm3_2(x)
        ###

       # x += msg
       # x = F.relu(x)

        x = self.linear4(x)
        x = F.relu(x)

        x = self.linear5(x)



        return F.log_softmax(x, dim=1)


class GCN(torch.nn.Module):
    def __init__(self):
        super().__init__()
```

```
        """super(GCN, self).__init__()"""

        self.inpu_size = 100
        self.num_layers = 1



        self.rnn = nn.RNN(self.inpu_size, self.hidden_size, self.num_layers, batch_first=True)
        self.input_layer_size = 87

        self.linear1 = nn.Linear(self.input_layer_size, 80)
        self.linear1_2 = nn.Linear(80, 80)
        self.linear2 = nn.Linear(80, 50)
        self.linear2_2 = nn.Linear(50, 50)
        self.linear3 = nn.Linear(50, 30)
        self.linear4 = nn.Linear(30, 4)

        self.conv1 = make_conv_layer(self.input_layer_size, self.input_layer_size)
        self.conv2 = make_conv_layer(self.input_layer_size, self.input_layer_size)

        self.conv3 = make_conv_layer(60, 30)

        self.dropout1 = nn.Dropout2d(0.0)

        self.batch_norm1 = BatchNorm1d(self.input_layer_size, eps=1e-5, momentum=0.9)
        self.batch_norm2 = BatchNorm1d(self.input_layer_size, eps=1e-5, momentum=0.9)
        self.batch_norm3 = BatchNorm1d(80, eps=1e-5, momentum=0.9)
        self.batch_norm4 = BatchNorm1d(50, eps=1e-5, momentum=0.9)
        self.batch_norm5 = BatchNorm1d(30, eps=1e-5, momentum=0.9)

    def forward(self, data, test=False):
        x, edge_index = data.x, data.edge_index
        msg = x
       # edge_index = edge_index[0]
        #print(edge_index.size())
        #edge_index = edge_index[0]
        batch_size = 24
        if test:
            batch_size = 1

        hidden = self.init_hidden(batch_size)
```

```
        x = self.conv1(x, edge_index)
        #x = self.msg_norm1(x, msg)
        #x = self.graph_norm1(x)
        x = self.batch_norm1(x)
        x = F.relu(x)
        x = self.dropout1(x)
        #x = F.dropout(x, training=self.training)
        #print(x.size())
        #print(hidden.size())
        x = self.conv2(x, edge_index)
        #x = self.msg_norm1(x, msg)
        #x = self.graph_norm1(x)
        x = self.batch_norm2(x)
        x = F.relu(x)
        x = self.dropout1(x)


        x = self.linear1(x)
        x = self.batch_norm3(x)
        x = F.relu(x)
        x = self.dropout1(x)



        x = self.linear2(x)
        x = self.batch_norm4(x)
        x = F.relu(x)
        x = self.dropout1(x)


        x = self.linear3(x)

        x = self.batch_norm5(x)
        x = F.relu(x)


        x = self.linear4(x)
```

```
        return F.log_softmax(x, dim=1)
```

A.4 confusion.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn


def get_confusion_matrix(labels_predicted, labels_true, classes):
    confusion_matrix = []

    labels_predicted = np.uint8(labels_predicted)
    labels_true = np.uint8(labels_true)

    labels_predicted = labels_predicted.flatten()
    labels_true = labels_true.flatten()

    print(labels_true.shape)
    print(labels_predicted.shape)
    for predicted_class in classes:
        row = []

        for true_class in classes:
            # All occurences of current true_class in labels_true:
            true_indices = np.where(labels_true == np.uint8(true_class))[0]


            # All occurences of current predicted_class in labels_predicted:
            predicted_indices = np.where(labels_predicted == np.uint8(predicted_class))[0]

            # We want to find the number of elements where these two matches:
            num_occurences = len( np.intersect1d(true_indices, predicted_indices) )
            row.append(num_occurences)
```

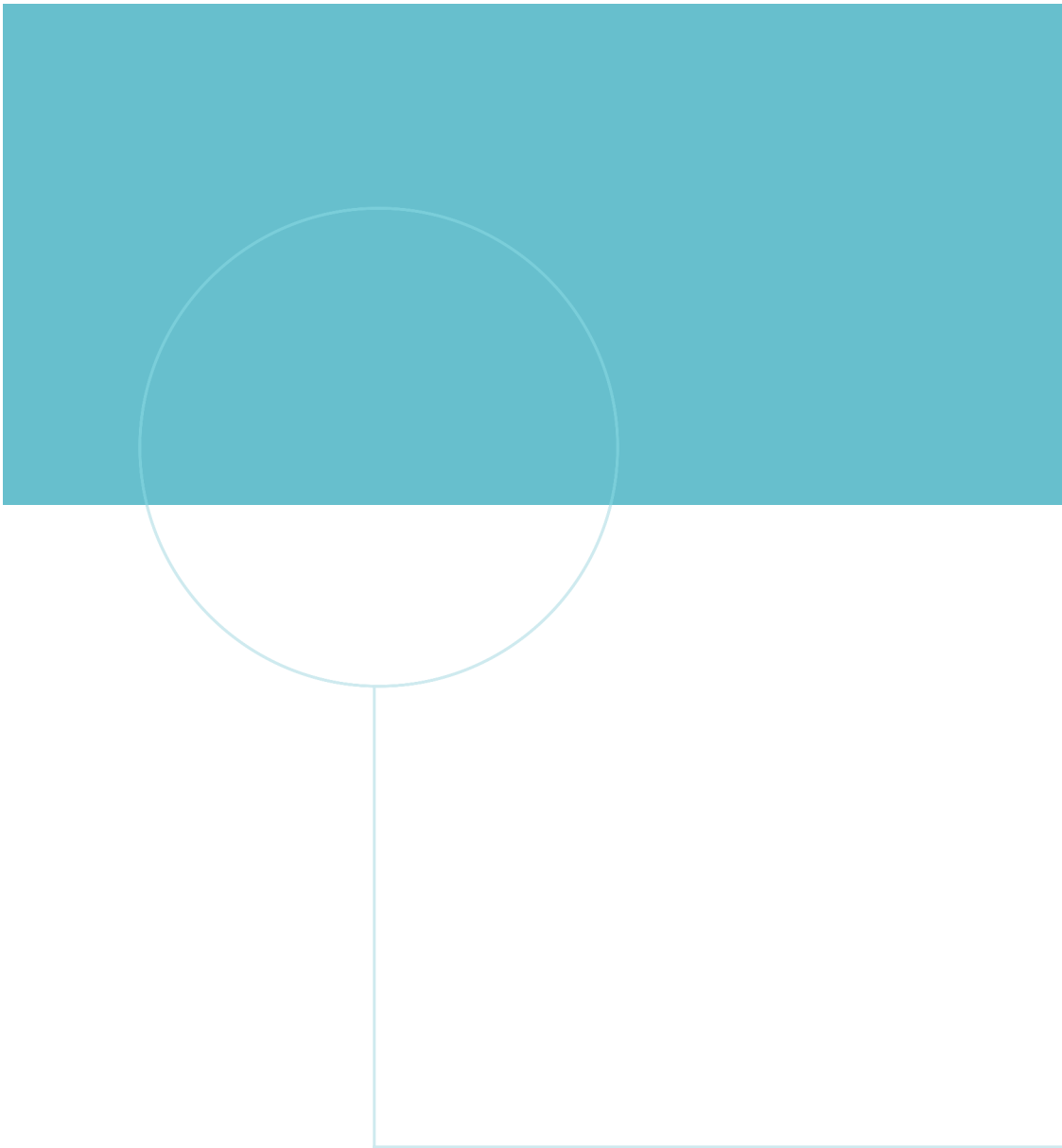
```
        confusion_matrix.append(row)

    return np.array(confusion_matrix)


def plot_confusion_matrix(confusion_matrix, name):
    class_numbers = ['1' ,'2', '3']
    df_cm = pd.DataFrame(confusion_matrix, index = [i for i in class_numbers],
                  columns = [i for i in class_numbers])
    fig = plt.figure(figsize = (10,7))

    fig.suptitle(name, fontsize=16)

    sn.heatmap(df_cm, annot=True)
```



NTNU

Norwegian University of
Science and Technology