NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

Retrogressive Thaw Slump identification using U-Net with Satellite Image Inputs

Remote Sensing Imagery Segmentation using Deep Learning techniques

Maria Leonor Caetano Soares Furtado

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade NOVA de Lisboa

RETROGRESSIVE THAW SLUMP IDENTIFICATION USING U-NET WITH SATELLITE IMAGE INPUTS

by

Maria Leonor Caetano Soares Furtado

Dissertation presented as partial requirement for obtaining the Master's degree in Advanced Analytics

Advisor: Mauro Castelli
Associate Professor, NOVA University of Lisbon

October 2021

To my family and friends.

Acknowledgements

Acknowledgments are personal text and should be a free expression of the author.

However, without any intention of conditioning the form or content of this text, I would like to add that it usually starts with academic thanks (instructors, etc.); then institutional thanks (Research Center, Department, Faculty, University, FCT / MEC scholarships, etc.) and, finally, the personal ones (friends, family, etc.).

But I insist that there are no fixed rules for this text, and it must, above all, express what the author feels.

"All models are wrong, but some are useful." (George E. P. Box)

Abstract

Global warming has been a topic of discussion for many decades, however its impact on the thaw of permafrost (frozen ground) and vice-versa has not been very well captured or documented in the past. This may be due to most of permafrost being in the Arctic and similarly vast remote areas, which makes data collection difficult and costly.

A partial solution to this problem is the use of Remote Sensing imagery, which isn't novel in documenting the changes in permafrost regions, where it has been widely used for decades. Despite its many benefits, this still required manual inspection of images which could be a slow tedious task for researchers.

Over the last decade, the use of Deep Learning on Remote Sensing imagery has risen in popularity, mainly due to the increased availability and scale of Remote Sensing data. This has been fuelled in the last few years by open-source multi-spectral high spatial resolution data, such as the Sentinel-2 data used in this project.

However, the use of Deep Learning for the particular use case of identifying the thaw of permafrost, addressed in this project, has nonetheless not been widely studied. To address this gap, the semantic segmentation model proposed in this project performs pixel-wise classification on the satellite images for the identification of Retrogressive Thaw Slumps (RTSs), using a U-Net architecture.

By using this technique, this project aims to aid the identification of these landforms in a more automated way and at a larger scale. Since these landforms can be a proxy for the thaw of permafrost, the hope is that this project can help make progress towards the mitigation of the impact of such a powerful geophysical phenomenon.

Keywords: Permafrost; Retrogressive Thaw Slump (RTS); Remote Sensing; Multispectral (MSI); Machine Learning (ML); Deep Learning (DL); Artificial Intelligence (AI);U-NET; Convolutional Neural Network (CNN); Computer Vision (CV) ...

Resumo

Nas últimas décadas o aquecimento global tem sido tópico de discussão. Contudo o seu impacto no derretimento de "permafrost" (solo permanentemente congelado) e vice-versa não tem sido bem registado no passado. Uma das razões que pode ter levado a isto é o facto do "permafrost" se encontrar no Ártico ou em regiões igualmente remotas de difícil acesso, o que faz com que a recolha de dados seja difícil e com demasiados custos.

Uma das soluções para este problema é a recolha de imagens de satélite para analisar as mudanças nas regiões de "permafrost", onde já é usada há muitas décadas. Apesar dos seus inúmeros benefícios, esta técnica requer inspeção e análise detalhada das imagens, e quando é feita manualmente pelos cientistas pode tornar-se monótona e demorada.

Nos últimos anos, o uso de "Deep Learning" para a análise de imagens de satélite tem aumentado, este popularismo deve-se ao aumento da quantidade e disponibilidade das mesmas. Os dados obtidos por esta tecnologia têm sido cada vez mais relevantes devido ao uso de sensores multiespectrais de alta resolução espacial, como aqueles usados neste projeto, provenientes da missão "Sentinel-2".

No entanto, o uso de "Deep Learning" especificamente para a identificação de "permafrost" não tem sido amplamente estudado. Para combater este problema, o modelo de "semantic segmentation" proposto neste projeto, classifica cada pixel nas imagens de satélite para identificar "Retrogressive Thaw Slumps (RTSs)", usando a arquitetura "U-Net".

Com o uso desta técnica, este projeto visa ajudar na identificação destas formas de relevo de uma maneira mais automática e numa escala maior. Como estas formas de relevo são uma boa indicação do derretimento de "permafrost", a esperança é que este projeto possa ajudar na mitigação do impacto deste poderoso fenómeno geofísico.

Palavras-chave: Permafrost; Retrogressive Thaw Slump (RTS); Imagens de satelite; Multispectral Instrument; Machine Learning (ML); Deep Learning (DL); Artificial Intelligence (AI);; U-Net; Redes Neuronais Con- volucionais; Visão computacional ...

Contents

List of Figures xix

List of Tables xxi

Acronyms xxiii

1 Introduction 1
- 1.1 Project Objectives and Research questions . . . 1
- 1.2 The task of identifying Retrogressive Thaw Slump (RTS)s using remote sensing data . . . 2
  - 1.2.1 The relevance of identifying RTSs using Deep Learning (DL) and remote-sensing imagery . . . 3
  - 1.2.2 Challenges and Opportunities . . . 3
- 1.3 Report Structure . . . 4

2 Theoretical Context 5
- 2.1 Permafrost and its degradation . . . 5
  - 2.1.1 Thermokarst landforms . . . 5
  - 2.1.2 Retrogressive Thaw Slumps . . . 6
- 2.2 Artificial Intelligence Applications in Remote Sensing . . . 7
  - 2.2.1 Conventional Machine Learning (ML) applications in Remote Sensing . . . 7
  - 2.2.2 DL applications in Remote Sensing Imagery . . . 9
  - 2.2.3 DL Models in Remote Sensing . . . 10
- 2.3 Convolutional Neural Network (CNN) . . . 11
  - 2.3.1 Main Layers of a CNN . . . 11
  - 2.3.2 Activation Functions . . . 16
  - 2.3.3 Classical CNN Architectures and their evolution . . . 19
- 2.4 The various architectures used in Remote Sensing applications . . . 22
  - 2.4.1 Fully Convolutional Network (FCN) . . . 22

2.4.2 DeepLab models . . . 23
2.4.3 U-Net . . . 23

3 Semantic Segmentation Models - Input data, algorithm and hyperparameters 25
3.1 Input data . . . 25
3.1.1 Labelled data . . . 25
3.1.2 Sentinel-2 data . . . 25
3.1.3 Sentinel-2 data collection . . . 26
3.1.4 Data understanding and preparation . . . 29
3.2 Introduction to semantic segmentation models . . . 31
3.2.1 Pre-processing of data . . . 31
3.2.2 Preparing the data for modelling using Tensorflow . . . 33
3.2.3 DL Learning process . . . 33
3.2.4 Measurement of model performance . . . 43
3.2.5 Transfer Learning . . . 43

4 Modelling Experiments 45
4.1 Choosing the most appropriate classification model . . . 45
4.1.1 Pixel-wise classification model . . . 45
4.1.2 Evaluating the use of transfer learning . . . 47
4.1.3 Pixel-level class imbalance . . . 47
4.1.4 Experiment set up . . . 48
4.2 Preprocessing experiments . . . 49
4.2.1 Size of the input patch . . . 49
4.2.2 Normalisation of input . . . 50
4.2.3 Data Augmentation . . . 51
4.3 Training experiments . . . 52
4.3.1 Network Hyperparameters . . . 52
4.3.2 Optimiser . . . 52
4.3.3 Learning rate . . . 53
4.3.4 Batch size . . . 55
4.3.5 Initialisation . . . 55
4.3.6 Activation function . . . 56
4.3.7 Loss Function . . . 57
4.3.8 Preventing over-fitting . . . 59
4.4 Learnings from conducting experiments . . . 60

5 Training the model on more data 63
5.1 Using the weights of previous model to train on new data . . . 63
5.2 Training the model from scratch on new data . . . 64
5.2.1 Comparison with the GEE dataset model - same parameters . . . 64

5.2.2 Hyperparameter tuning from scratch 65

6 Final model and evaluation **67**
6.1 Final Model 67
6.2 Test Evaluation 67

7 Conclusions and Future Work **71**
7.1 Conclusions 71
7.2 Limitations 72
7.3 Future Work 72

Bibliography **75**

Annexes

List of Figures

2.1 Landscape features and processes info-graphic (Philipp et al., 2021) . . . 6
2.2 Example of an RTS with added legend. (b) and (c) are the ground photo and remote sensing image of an RTS whose central location is 92.912° E, 34.848° N. (L. Huang et al., 2020) . . . 7
2.3 Three-layer back-propagation neural network (Yang et al., 2008) . . . 9
2.4 Study target of DL in Remote Sensing studies (L. Ma et al., 2019) . . . 9
2.5 General Framework of Remote Sensing Image Classification Based on DL (Li et al., 2018) . . . 10
2.6 DL models used in Remote Sensing studies (L. Ma et al., 2019) . . . 11
2.7 The main layers of a CNN (Hiep and Joo, 2018) . . . 12
2.8 The movement of the filter in a 2D CNN layer (Ferretti et al., 2020) . . . 13
2.9 The Convolution Operation in an n-channel input CNN (Panchbhaiyye and Ogunfunmi, 2020) . . . 13
2.10 Different types of pooling with 2x2 kernel and stride 2 . . . 16
2.11 Softmax activation function (Nabiyev and Malekzadeh, 2021) . . . 17
2.12 Tahn and Sigmoid activation functions (Fathi and Maleki Shoja, 2018) . . . 18
2.13 Rectified Linear activation functions (Shenoy, 2019) . . . 19
2.14 Architecture of LeNet-5 (Lecun et al., 1998) . . . 20
2.15 Architecture of AlexNet (Krizhevsky et al., 2012) . . . 20
2.16 U-Net Architecture, blue boxes are feature maps and white boxes are copied feature maps (Ronneberger et al., 2015) . . . 23

3.1 Google Earth Engine (GEE) Console User Interface (UI) showing a snippet of the bespoke JavaScript script . . . 26
3.2 Example of the Red, Green, Blue (RGB) channels normalised using z-score 28
3.3 Example of a RTS mask . . . 28
3.4 Distribution of GEE data RTS positive pixels . . . 29
3.5 Distribution of JPEG 2000 (JP2) data RTS positive pixels across datasets . 30
3.6 NN hyperparameter optimization cycle (Stock et al., 2020) . . . 34

3.7 Overview and relationship among the existing loss functions. (J. Ma et al., 2021) . . . 35
3.8 Convergence of a 30-layer CNN in He et al.'s paper (He et al., 2015c) . . . 40
3.9 (a) Regular 2-layer Neural Networks (NN) (b) Example of NN after dropout applied. Crossed neurons have been turned off. (Shanmugamani, 2018) . 41

4.1 Example of convolution block($x = 16$, $z = 0.1$) . . . 46
4.2 Example of transpose convolution block($x = 16$, $z = 0.1$) . . . 46
4.3 U-Net architecture used in this project . . . 47
4.4 Input patch size Dice Coefficient comparison . . . 49
4.5 Normalisation method Dice Coefficient comparison . . . 51
4.6 Augmentation method Dice Coefficient comparison . . . 52
4.7 Optimiser Loss comparison . . . 53
4.8 Learning rate Dice Coefficient comparison . . . 54
4.9 Initialisation method Dice Coefficient comparison . . . 56
4.10 Activation function Loss comparison . . . 57
4.11 Loss function Dice Coefficient comparison . . . 58
4.12 Early Stopping (ES) usage Loss comparison . . . 59
4.13 Removal of Dropout Loss comparison . . . 60

5.1 Retraining with frozen weights Dice Coefficient comparison . . . 63
5.2 Retraining from scratch Loss comparison . . . 64
5.3 Subset of hyperparameter experiments with new data Validation dice coefficient . . . 66

6.1 Dice Score comparison . . . 67
6.2 Loss comparison . . . 67
6.3 Dice Score vs. Intersection over Union (IoU) Score Test images scatter plot 68
6.4 Test set predicted vs. ground truth high score examples . . . 69
6.5 Test set predicted vs. ground truth medium score examples . . . 69
6.6 Test set predicted vs. ground truth low score examples . . . 70
6.7 Test set predicted vs. ground medium score example . . . 70

List of Tables

3.1 Distribution of GEE data for scene classification 29
3.2 Train/ Validation/ Test split of samples for GEE data 30
3.3 Train/Validation/Test split of samples for JP2 data 30

4.1 Input patch size comparison of Dice Coefficient and Loss 50
4.2 Normalisation method comparison of Dice Coefficient and Loss 50
4.3 Optimiser comparison of Dice Coefficient and Loss 53
4.4 Learning rate comparison of Dice Coefficient and Loss 54
4.5 Batch size comparison of Dice Coefficient and Loss 55
4.6 Initialisation method comparison of Dice Coefficient and Loss 56
4.7 Activation function comparison of Dice Coefficient and Loss 57
4.8 Loss function comparison of Dice Coefficient and Loss 58

5.1 Retraining on new data comparison of Dice coefficient and Loss 65
5.2 Hyperparameter combinations comparison of Dice Coefficient and Loss . 66

6.1 Dice score test image summary 68

Acronyms

a.k.a.	also known as
Adam	Adaptive Momentum Estimation
AE	Autoencoders
ANN	Artificial Neural Network
API	Application Programming Interface
AWI	Alfred Wegener Institute
BGD	Batch Gradient Descent
BPNN	Back-Propagation Neural Networks
CE	Cross Entropy
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRF	Conditional Random Field
CRS	Coordinate Reference System
CV	Computer Vision
DBN	Deep Belief Networks
DL	Deep Learning
DSM	Digital Surface Model
ELU	Exponential Linear Unit
EPSG	European Petroleum Search Group
ES	Early Stopping
FCN	Fully Convolutional Network
GAN	Generative Adversarial Network

GCP Google Cloud Platform
GCS Google Cloud Storage
GEE Google Earth Engine
GHGs Green House Gases
GPU Graphic Processing Unit

IoU Intersection over Union

JP2 JPEG 2000

LULC Land Use and Land Cover

ML Machine Learning

NAdam Nesterov-accelerated Adaptive Momentum Estimation
NAG Nesterov Accelerated Gradient
NiN Network in Network
NIR Near Infra-Red
NN Neural Networks

PRD Permafrost Region Disturbances

RAM Random Access Memory
ReLU Rectified Linear Unit
RF Random Forest
RGB Red, Green, Blue
RMSProp Root Mean Square Propagation
RNN Recurrent Neural Networks
ROI Region of Interest
RTS Retrogressive Thaw Slump

SEI Stockholm Environment Institute
SELU Scaled Exponential Linear Unit
SGD Stochastic Gradient Descent
SVM Support Vector Machines
SWIR Short Wave Infra-Red

UI User Interface

VNIR Visible Near Infra-Red

w.r.t. with respect to

1

Introduction

Permafrost is found mainly in the Arctic region, where it covers a quarter of the Northern Hemisphere's land (Olthof et al., 2015). The carbon content stored in the frozen ground is thought to be double that in the atmosphere (chuur E. McGuire, 2015). As this frozen ground thaws due to global warming and other climate-change-driven events, the stocks of carbon get released into the atmosphere. This release of Green House Gases (GHGs) is estimated to cause extra global costs of climate change impacts of dozens of trillions of dollars in the next two to three decades (Calel R., 2020).

The thaw of permafrost also known as (a.k.a.) permafrost degradation can not only lead to damage to roads and other man-made infrastructures but the release of said trapped GHGs could also contribute further to global warming and even more permafrost degradation (Murton, 2021), as it warms and thaws at a faster rate, creating a vicious cycle of global warming that is threatening life on Earth.

Although little can be done to prevent permafrost degradation, the ability to identify it in a semi-automated way using DL and Remote Sensing imagery can go a long way in assessing and forecasting permafrost thaw related landscape changes, hopefully feeding into carbon budget estimations and the development of mitigating responses (Osterkamp and Jorgenson, 2009a).

DL in remote sensing has been used in a wide range of applications, many of which have been captured in a recent review study, including but not limited to Land Use and Land Cover (LULC) classification, image fusion, image registration, object detection, scene classification and image segmentation (L. Ma et al., 2019).

This project aims to combine remote sensing multispectral imagery data with the advancements of DL frameworks to construct an automated way of helping to identify signs of the patterns and processes associated with permafrost degradation.

1.1 Project Objectives and Research questions

This project has the primary aim of using remote sensing multispectral imagery to build a DL neural network that allows the identification of retrogressive thaw slumps

in an attempt to assess the thawing of permafrost in the Arctic.

To achieve this goal, this project aims to answer the following questions:

1. How to deal effectively with the challenge of extracting and pre-processing multi-spectral imagery?
2. How well can different CNN meta-architectures perform on remote sensing imagery?
3. What is the impact of changing hyperparameters and input parameters in the network?
4. Is it possible to achieve satisfactory results using only 10-meter resolution imagery, or is higher resolution needed?

1.2 The task of identifying RTSs using remote sensing data

The use of remote sensing imagery to analyse and monitor RTSs has been widespread for many decades for many use cases in remote regions where RTSs are more commonly found. Due to this remote location access to these areas is not only logistically complicated but also expensive, this may have contributed to the popularity of use of remote sensing imagery due to its reduced cost and ease of data collection.

Up until the last 5 years, most techniques used to identify RTSs were either manual visual inspection, traditional statistical methods or conventional ML techniques such as Linear Regression, Random Forest amongst others.

RTSs were often identified manually, either by simply identifying RTSs visually with tools such as GEE Timelapse (Lewkowicz and Way, 2019) or by combining remote sensing imagery with other data such as field observations and historical weather data to investigate the growth of RTSs (Kokelj et al., 2015). This is a lengthy laborious process, taking precious research time away from specialists, leading to a desire to automate this process as much as possible.

In a step towards this, Nitze et al. (Nitze et al., 2018) take remote sensing data and apply Linear Regression and Random Forest techniques to classify RTSs and other types of Permafrost Region Disturbances (PRD).

Futhermore, since 2018, Huang et al. (L. Huang et al., 2018) (L. Huang et al., 2020) (L. Huang et al., 2021) makes use of DL techniques to identify RTSs, making unprecedented contributions to the field of identification of RTSs by using DeepLabv3+ to assign a label to each pixel in satellite images taken by the Planet CubeSat constellation at a regional level.

The availability of labelled data from the Arctic, provided by researchers during a project with both the Alfred Wegener Institute (AWI) and Stockholm Environment Institute (SEI), was the missing element, proving the feasibility of using DL for the identification of RTSs in this region.

The main goal of this project is to build a DLmodel that identifies thaw slump locations and shapes in the Arctic region through pixel-wise classification of remote sensing images. This project aims to make a significant contribution to this field by attempting to fill some knowledge gaps in RTS identification and provide a basis for future carbon emissions estimations.

To this author's knowledge, there are no state-of-the-art segmentation models focused on the use of satellite images to identify RTSs in the Arctic.

1.2.1 The relevance of identifying RTSs using DL and remote-sensing imagery

Nitze et al. (Nitze et al., 2018) highlight the uncertainty of the scale of permafrost degradation at a rapid pace is a result of most PRD not being documented due to their under-representation in the remote sensing studies. They also expect permafrost to degrade faster than the current projections by models that don't consider PRD-driven thaw. These models don't take into account the full extent of increased carbon emissions from permafrost thaw, which can further contribute to global warming, a.k.a. permafrost carbon feedback (Schaefer et al., 2014).

In (Osterkamp and Jorgenson, 2009b) the need for more research targeted at identifying and monitoring PRD for the benefit of monitoring GHGs, and the estimation of the consequences of its monitoring on global warming and the future of life on earth, is well-argued.

Despite the extensive literature on using remote sensing imagery and DL in the field of LULC, it is very limited when it comes to the identification of permafrost disturbances. A recent review study on remote sensing for permafrost-related analysis (Philipp et al., 2021) found that only a handful amongst 325 articles in the last two decades actually used DL for permafrost-related analysis, and only one by Huang et al. (L. Huang et al., 2020) refers to RTSs in particular.

By developing a project through Accenture with researchers from both the AWI and SEI the feasibility of using DL for the identification of RTSs the relevance of this project became clear.

This project aims to make a significant contribution to this field by attempting to fill some knowledge gaps in RTSs identification and provide a basis for future carbon emissions estimations.

1.2.2 Challenges and Opportunities

One of the biggest challenges ahead of this project will be the satellite data extraction in an automated way, this is due to the unreliability of the Application Programming Interface (API)s available to perform this task. Another challenge could be the identification of small RTSs given that the smallest pixel resolution available in this project

represents a 10x10 meter area, and some thaw slumps can be as small as that if not smaller.

There is an opportunity of using techniques used in other widely researched remote sensing applications, for example LULC use cases, that could provide some guidance in designing and training the DL model introduced in this project.

1.3 Report Structure

Chapter 2 will consist of the theoretical context of this project. Chapter 3 will describe the chosen methodology, including the chosen data and algorithms. Chapter 4 will outline the experiments conducted in the optimisation of the model of choice. Chapter 5 will assess the behaviour of the model when trained on more data. Chapter 6 will be dedicated to the final model and its evaluation on unseen data. Chapter 7 will summarise the conclusions of this project, its limitations, and suggestions for future work.

2

Theoretical Context

All the necessary scientific concepts and ideas will be discussed in this section to provide the reader with a theoretical context of the task at hand.

2.1 Permafrost and its degradation

Permafrost is characterised as frozen ground that has a temperature colder than 0 degrees C continuously for 2 or more years (Everdingen and International Permafrost Association (USA), 1998).

As the Arctic has warmed twice as fast as the globe on average, a.k.a. Arctic amplification (Cohen J., 2014) the consequences of global warming are disproportionally felt in the Arctic.

In the Northern Hemisphere, huge amounts of soil organic carbon are trapped in permafrost soils. Any change in boundary conditions that causes the ground to warm will result in the thaw of permafrost. These abrupt thaw processes will expose previously frozen soil organic matter, causing it to undergo microbial degradation, and release GHGs e.g. methane and carbon dioxide as a result.

The thaw of permafrost (degradation) can not only lead to damage to roads and other man-made infrastructures but also lead to the release of said trapped GHGs which consequently could contribute further to global warming and even more permafrost degradation (Murton, 2021) as it warms and thaws at a faster rate, creating a vicious cycle of global warming.

This could further exacerbate Arctic amplification, therefore it is imperative to monitor and predict the volume of GHGs released into the atmosphere due to the abrupt thaw of permafrost.

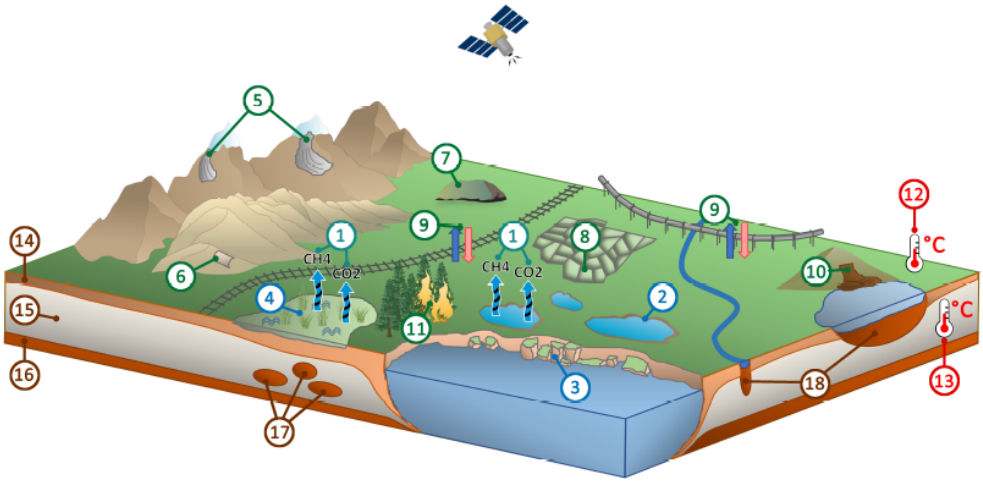
2.1.1 Thermokarst landforms

The extent of permafrost and its degradation is difficult to measure and historically requires extensive and costly investigation. A way to attempt this is to look for evidence

of the formation of thermokarst landforms such as those depicted in Figure 2.1 which cause visible geological changes in close range of abrupt thaw locations.

Thermokarst is the sinking of the ground's surface due to thawing of the ground (permafrost degradation). There are many examples of thermokarst landforms, amongst the most common and fast-changing are lakes and ponds (Rowland et al., 2010).

In Figure 2.1, identified by label number 10, we can also see another relevant example, RTS, this paper will focus on the identification of this thermokarst feature in an attempt to capture some extent of permafrost degradation in the Arctic. The next sub-section will describe RTSs further.



Atmospheric Features and Processes:
1: *Greenhouse gas emissions*

Surface Water Features and Processes:
2: Surface water extent dynamics
3: *Coastal erosion*
4: Bog/fen development

Surface Land Features and Processes:
5: Rock glaciers
6: Landslides
7: Pingo development
8: Patterned ground
9: *Frost heave and thaw settlement*
10: Thaw slumps
11: *Wild fires*

Thermal Features and Processes:
12: *Land surface temperature dynamics*
13: *Ground temperature dynamics*

Subsurface Features and Processes:
14: Active layer thickness dynamics
15: Permafrost
16: Unfrozen ground
17: Cryopeg
18: Talik

Figure 2.1: Landscape features and processes info-graphic (Philipp et al., 2021)

2.1.2 Retrogressive Thaw Slumps

RTSs can usually be described as horseshoe-shaped landslides caused by the thawing of ice-rich permafrost, which causes erosion mostly at its steep head scarp. RTS usually occur on sloped terrain so that the thawed material can flow downslope, usually into nearby water features such as lakes and rivers. (Osterkamp and Jorgenson, 2005)

This means that RTSs have particular parts that can ease their visual identification through the mentioned head scarp, the headwall that forms after the slump, the slump floor itself (Lantuit and Pollard, 2008) which is also described as slump scar zone. These have been labeled in Figure 2.2 for the reader's benefit.

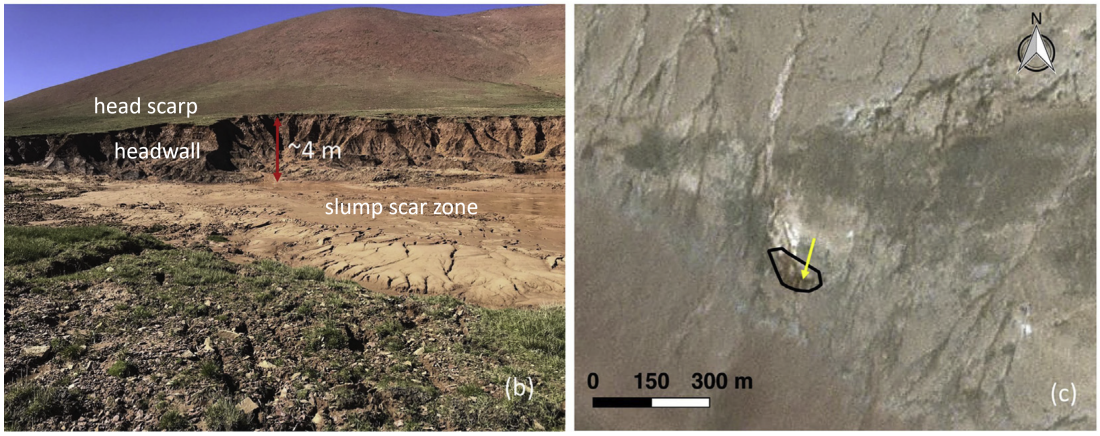


Figure 2.2: Example of an RTS with added legend. (b) and (c) are the ground photo and remote sensing image of an RTS whose central location is 92.912° E, 34.848° N. (L. Huang et al., 2020)

As some sections of the slump scar stabilise they become vegetated (Kokelj et al., 2015) and thus harder to identify using only visual Red (R), Green(G) and Blue (B) channels, this is where remote sensing's other "sensor"bands can identify other features not seen with the "naked eye"(L. Huang et al., 2020)

2.2 Artificial Intelligence Applications in Remote Sensing

The advancements in Artificial Intelligence have made it possible to identify and classify particular features in images and like in this case, even remote sensing imagery, this section will introduce the techniques used in this field.

Given the lack of research found on the specific task of identifying RTSs using remote sensing imagery, a more expansive approach on looking at DL and ML techniques using remote sensing imagery was taken.

2.2.1 Conventional ML applications in Remote Sensing

Before the rise in popularity of DL applications in Remote Sensing imagery, more shallow structures that process features extracted from the sattelite images,such as Random Forest (RF), were used. This type of ML require features to be extracted by experts and flattened to enable the algorithm to use them, this can be time consuming and can be overcome by the use of DL, which does the feature map extraction as part of the algorith as will be seen in section 2.3.1.

In a study (Bramhe et al., 2018) on extracting built-up areas from Sentinel-2 remote sensing imagery, DL techniques were compared against the most widely used conventional ML methods in remote sensing classification to show the benefits of usng DL algorithms, both Gaussian Support Vector Machines (SVM) and Back-Propagation Neural Networks (BPNN) were evaluated.

SVM

According to a 2010 review (Mountrakis et al., 2011) of over 100 studies on the use of SVM in remote sensing, this model's popularity in the field came from its ability to generalise well even when there is limited training data, which is quite common in remote sensing problems. Their major limitation comes in the form of parameter assignment issues, which affect the quality of the results.

RF

RF has always been a popular method, as it is an ensemble classifier that makes use of multiple decision trees combined with random subsampling of the training data and variables.

According to a review on the use of RF in remote sensing, (Belgiu and Drăguţ, 2016) RF became a popular classifier in remote sensing due to the high accuracy of its classifications. It also seems to successfully handle the high dimensionality and multicollinearity of remote sensing data, as well as, being fast and not prone to overfitting. Its main limitation is its sensitivity to the sampling design.

BPNN

The traditional BPNN is a popular ML algorithm, according to a review (Yuan et al., 2020) in environmental remote sensing it has been used extensively for research in this field. Yuan et al. reports improvement in accuracy against traditional regression methods across a number of papers by using BPNN, but highlights a couple of limitations: the slow convergence of the algorithm and how much it is affected by weight initialisation being susceptible to getting stuck in local minimums.

This traditional Neural Network forms the back-bone of many DL models, so it is important to understand its inner works.

A BPNN consists of one or more hidden layers between an input and an output layer, each layer having many neurons/nodes that are fully connected to the next layer as it can be seen from Figure 2.3. It is trained via forward and backward propagation, that is: starting with forward propagation, the input is propagated through the hidden layers in the network until it reaches the output layer where the error between the predicted and actual values is calculated.

Straight after the error is calculated, it is propagated backward to update the neuron weights of each layer to minimise the error.

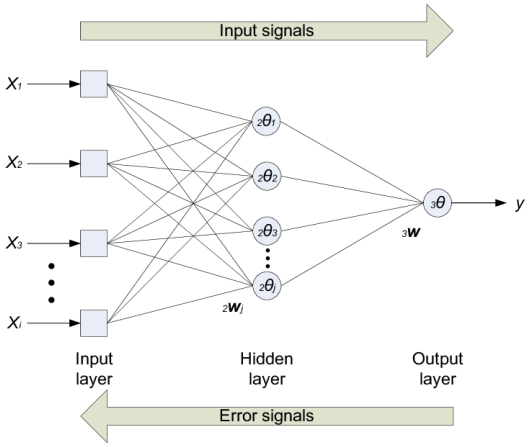


Figure 2.3: Three-layer back-propagation neural network (Yang et al., 2008)

2.2.2 DL applications in Remote Sensing Imagery

Looking at the field of DL with Remote Sensing, most studies have been focused on the field of LULC classification, as suggested by a DL in Remote sensing application review (L. Ma et al., 2019) in Figure 2.4. Object detection, Scene Classification and Segmentation (a.k.a. pixel-wise classification) are also techniques to classify images or areas of these images.

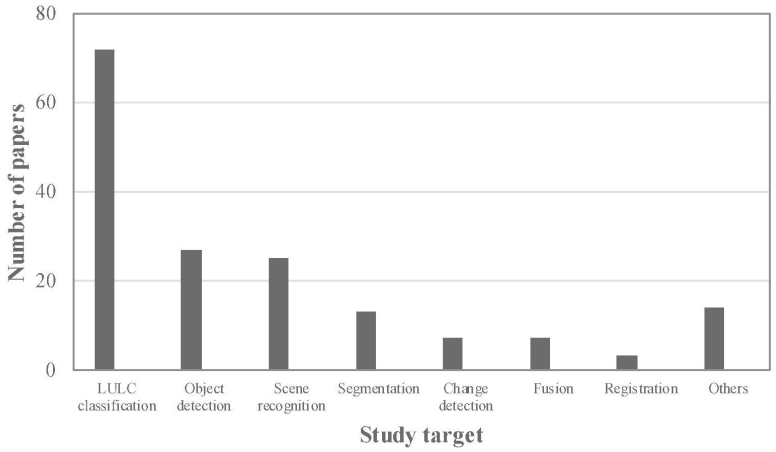


Figure 2.4: Study target of DL in Remote Sensing studies (L. Ma et al., 2019)

In traditional natural images, image classification is the task of assigning a label to a whole input image, either by outputting a class or a probability of tasks that describe said image. However, this is not the case in remote sensing, where the concept is broader, referring to either pixel-wise or "scene"classification, as shown in Figure 2.5.

Pixel-wise classification The classification of each pixel, similar to the concept of image segmentation, as it is known when performed in natural images. Each pixel in a remote sensing image could represent, as an example, a 10 by 10 meter area, which would usually be associated with a natural image area.

So by classifying each pixel in a remote sensing image, we are classifying a 10x10 meter area, each class for example represented by the different colours at the top right of Figure 2.5.

Scene classification The automatic assignment of a semantic label to a scene, represented by the different coloured "containers"at the bottom right of Figure 2.5. A scene being a local image patch, usually manually extracted from large-scale satellite images that contain classes (e.g. forest areas, residential area)(Li et al., 2018).

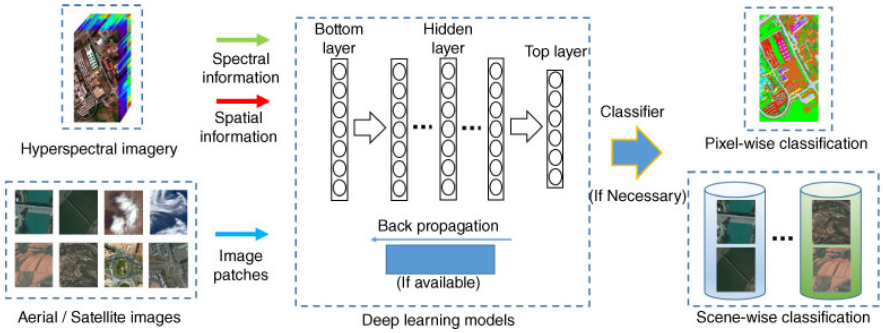


Figure 2.5: General Framework of Remote Sensing Image Classification Based on DL (Li et al., 2018)

Object detection In remote sensing images, it is used to find out if a satellite image has one or more objects belonging to a given class and find the position of each object predicted in the image.(Cheng and Han, 2016)

2.2.3 DL Models in Remote Sensing

The same review (L. Ma et al., 2019) also suggests that the most used DL model in Remote Sensing imagery is CNN as indicated in Figure 2.6.

The models in Figure 2.6 can be classified into two types of DL models: CNN, Recurrent Neural Networks (RNN), and FCN are supervised models, whereas Autoencoders (AE), Deep Belief Networks (DBN), Generative Adversarial Network (GAN) models are unsupervised models.

Supervised models require labeled data to learn from the ground truth, whereas unsupervised ones don't. For example, Alvarez et al. (Alvarez et al., 2020) use GAN to solve the binary change detection problems in remote sensing by exploiting the discriminator likelihood to generate the distribution of unchanged samples.

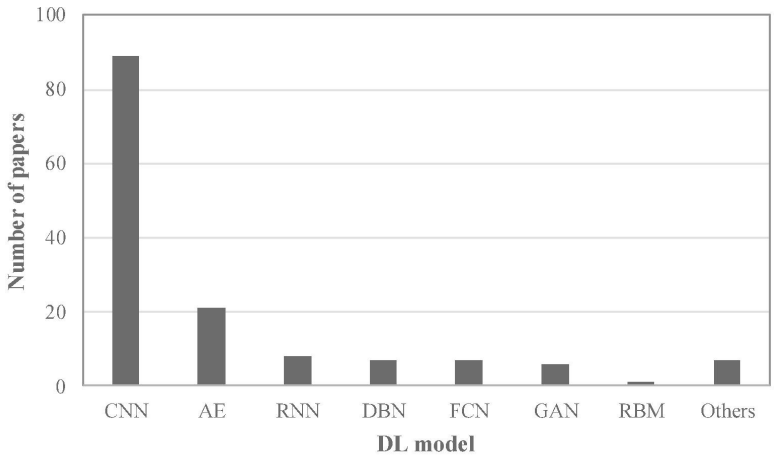


Figure 2.6: DL models used in Remote Sensing studies (L. Ma et al., 2019)

As labeled data has been collected for this project, its focus will be on a supervised DL Model. CNN has been chosen as its the most popular supervised techniques, section 2.3 will expand on this.

2.3 CNN

CNNs are a type of Artificial Neural Network (ANN) that have been praised for their contribution to the field of computer vision. CNNs have been very successful and efficient in image classification, object detection, and many other computer vision applications (Goodfellow et al., 2016).

In general, a CNN takes an input image, extracts low-level features, and hierarchically builds on this to extract more abstract features, so that it is able to extract features that are common for all outputs. This concept has its origin in the biology of the visual cortex, which has small regions of cells that "light up"to specific characteristics of the visual field until the entire visual field is processed and categorised.

2.3.1 Main Layers of a CNN

Like any other Neural Network, a CNN is composed of an input layer, an output layer, and several hidden layers connecting them, as shown in Figure 2.7.

The input layer usually consists of a representation of the input image to be analysed, and the output layer of the probabilities for each class. The hidden layers usually consist of convolutional, non-linearity, pooling, and fully connected layers, which are the building blocks for most CNNs and therefore will be described in detail below.

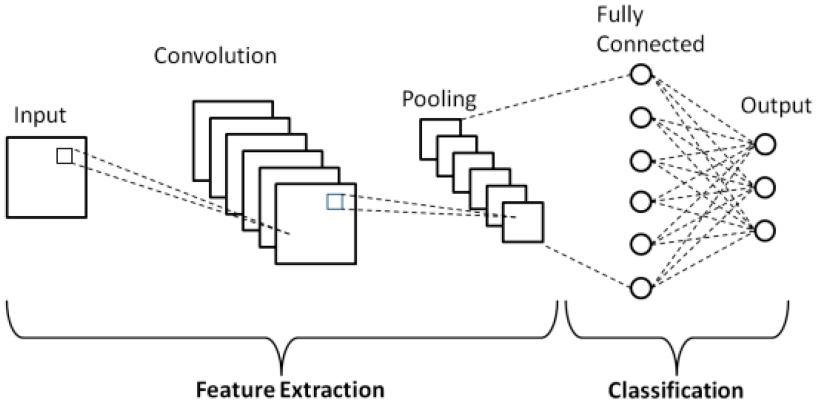


Figure 2.7: The main layers of a CNN (Hiep and Joo, 2018)

2.3.1.1 Input Layer

Each image is represented as a 3D matrix with dimensions of width (W), height (H), and depth (D), respectively. In the field of natural images the third dimension, depth, may take the value of 3 for RGB channels of images, or one for grey colour images. However, in the field of remote sensing imagery this dimension can have N channels each associated with readings from a different sensor. For example Sentinel-2 images have 13 channels, this will be explored in detail in section 3.1.2.

2.3.1.2 Convolutional Layer

The objective of the Convolution Operation, after which the layer has been named, is to extract features from an image. It is the key differentiator from a Dense layer, as it enables the ability to capture dependencies between pixels through the application of filters. It learns local patterns rather than global patterns. In abstract mathematical terms, it takes the matrix of the image and the matrix of a filter/kernel and merges the information in both. By making the filter smaller than the input dimension, sparse interaction can be achieved, reducing the memory requirements and improving the model's efficiency.

It also means that neurons are constrained to using the same set of weights for getting the output, a.k.a. shared parameters. This parameter sharing gives the Convolution Operator the property of translation equivariance, meaning that if we translate the input the output will also be translated, giving CNNs the ability to learn features regardless of their position.

A Convolutional layer is a layer that applies different convolution operations to the data, making it the most essential building block of a CNN, and its most computationally expensive one as well. To fully understand this process, it will be broken down below.

The filter/kernel, introduced above has dimensions (K) smaller than the input

image for height and width but the same third dimension, the number of channels (n). The filter moves along the width and height of the input image, with a certain stride (s) value, performing matrix multiplication between the filter and the same dimensional portion of the image over which the filter is passing (depicted by the shaded area in Figure 2.8) until it traverses the entire image.

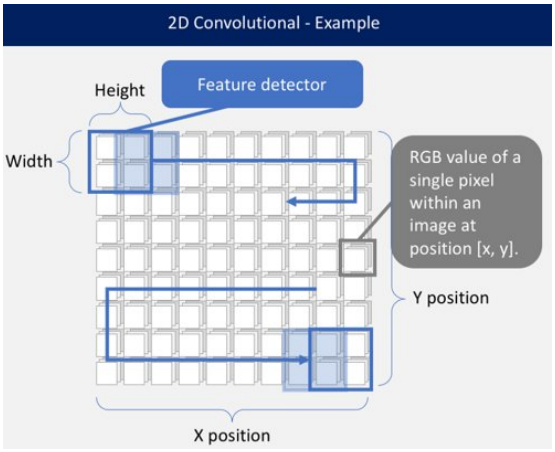


Figure 2.8: The movement of the filter in a 2D CNN layer (Ferretti et al., 2020)

In the case of images with n-channels, the matrix multiplication is performed between the kernel and input channel stacks as depicted in Figure 2.9, all the results are then summed with the bias to give a one-depth channel convoluted feature map with depth equal to the number of filters (m).

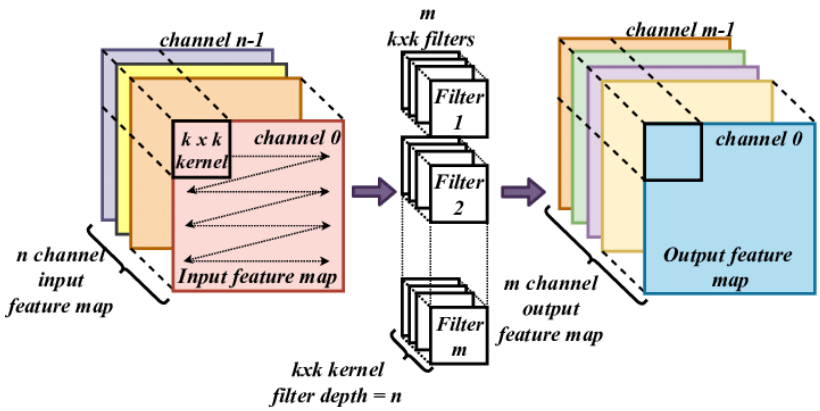


Figure 2.9: The Convolution Operation in an n-channel input CNN (Panchbhaiyye and Ogunfunmi, 2020)

The size of this feature map will be affected by several parameters, some of which have been mentioned above already:

- **Filter dimensions** (K) The dimensions of the filters used in the layer. For example, k x k x n in Figure 2.9.

- **Stride (s)** The number of pixels the filter shifts over the input matrix. When the stride is 1, the filters are moved 1 pixel at a time. The larger the stride, the smaller the feature map.
- **Number of filters (m)** The number of filters used in the convolutional layer. It is said that the higher the number of features the more image features get extracted and the better the CNN gets at recognising patterns in unseen images.
- **Zero Padding (p)** Sometimes the filter dimensions don't fit the image input dimensions perfectly. A strategy commonly used to maintain the input matrix dimensions and avoid the loss of information is to pad said matrix with 0s around the borders.

The output convolution dimensions, assuming a symmetric CNN architecture and image to be square that is height equals width, so both dimensions have the same values, can be calculated as follows (Dumoulin and Visin, 2018):

$$dim_{out} = \frac{dim_{in} + 2p - K}{s} + 1 \tag{2.1}$$

If either the architecture or the input size is asymmetric, 2.1 can be used to calculate the feature map dimensions separately for height and depth.

When it comes to depth as shown in Figure 2.9, the output depth dimension is always equal to the number of filters m:

$$D_{out} = m \tag{2.2}$$

A CNN will usually have several convolutional layers to learn spatially hierarchical patterns. The first convolutional layer will learn local simple patterns such as lines and edges, then a second convolutional layer will learn features made up of the features learned in the first layer to learn more complex patterns.

As we go deeper into the network, the filters also begin to be more responsive to a larger region of the pixel space, enabling it to learn larger patterns. The process will repeat itself for subsequent layers, the number of layers will depend on the complexity of the input image's features.

2.3.1.3 Non-Linearity Layer

Non-linearity is a key feature of NN that enables the modelling of outputs that can't be produced from linear combinations of the inputs. A non-linearity layer is nothing more than an activation function that takes the feature map generated by the convolutional layer and creates an activation map as the output.

The different activation functions are described in section 2.3.2 in detail. Most of the recent CNN meta-architectures use Rectified Linear Unit (ReLU) (or its derivatives, such as leaky ReLUs) due to their efficiency and robustness to noise (He et al., 2015b).

2.3.1.4 Pooling Layer

The Pooling Layer, a.k.a. subsampling or downsampling layer, usually follows a Convolutional Layer or a Non-Linearity Layer if these are separate. Its main function is to reduce the dimensionality and subsequently the number of parameters in the network, whilst retaining the most important information of the activation map. This can reduce the training time/ computational power required and increase training efficiency through extracting the dominant translation-invariant features.

Just like the convolutional layer above, a window slides through each feature map applying the pooling operation, so spatial neighbourhood dimensions/ pooling kernel size (K_p) and stride (s) are important parameters that need to be defined beforehand. For example, in an overlapping pooling layer $K_p > s$, one can calculate the feature map output dimensions following a pooling layer as follows (Dumoulin and Visin, 2018) using equations 2.3 and 2.4:

$$dim_{out} = \frac{dim_{in} - K_p}{s} + 1 \tag{2.3}$$

When it comes to depth, the output depth dimension is always equal to the input dimension depth:

$$depth_{out} = depth_{in} \tag{2.4}$$

To achieve this, several spatial pooling types can be applied interchangeably, the most common ones are described below. Figure 2.10 depicts an example of a non-overlapping pooling layer, where $K_p = s$:

- **Max Pooling** It returns the maximum value from the window of the image covered by the pooling kernel. It discards noisy activations, thus performing denoising as well as dimensionality reduction. This could help avoid overfitting. In practice, this type of pooling shows the best performance (Goodfellow et al., 2016).

- **Average Pooling** It returns the average of all the values from the window of the image covered by the pooling kernel, only performing dimensionality reduction.

2.3.1.5 Fully Connected Layer

Finally, once the outputs of all the layers above successfully represent the high-level features of the input image, this output is flattened into a 1D matrix that can be fed to the Fully Connected Layers. The first few layers of this type learn non-linear combinations of these features to identify which of these features most strongly correlate with the output classes.

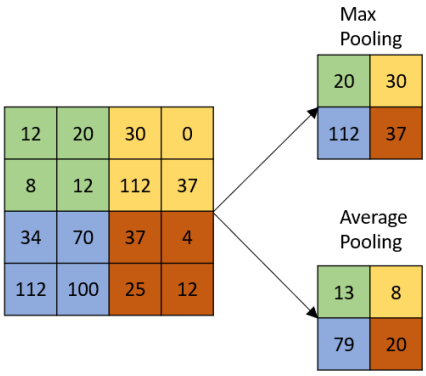


Figure 2.10: Different types of pooling with 2x2 kernel and stride 2

In a scene classification problem, to identify the most likely output class, the last of these fully connected layers usually has a softmax (multi-class problem) or sigmoid (binary class problem) activation function to output a 1xN matrix of probabilities, with each element in this matrix corresponding to the probability of an object belonging to a specific class and all of these probabilities summing to one.

In a pixel-wise classification problem, the final layer could be used to perform a pixel-wise prediction and will output a WxH matrix of probabilities, with each element in this matrix corresponding to the probability of an object belonging to a specific class. More detail on how the necessary WxH dimensions are upsampled is given in the following layer.

2.3.1.6 Fractionally-strided/ Transposed Convolutional Layer

For example, in an architecture that allows for end-to-end pixel-wise classification, such as FCN), a fractionally-strided layer (Long et al., 2015) is needed to link the network's outputs back to the pixels using learnable parameters. It is used to upsample the feature map within the network back to the image size dimensions as a segmentation map to enable end-to-end learning by backpropagating the pixel-wise loss. It is often misleadingly referred to as deconvolutional layer, due mainly to its use in a study (Zeiler and Fergus, 2013) that visualises convolutional networks.

In a pixel-wise classification problem, this could also be used as the last layer, like in Deconvnet (Noh et al., 2015) instead of feature maps this Transposed Convolutional Layer generates pixel-wise class probabilities corresponding to the size of the input images, the said WxH matrix described above.

2.3.2 Activation Functions

There are several activation functions used for different purposes, those commonly used in hidden layers are the Tahn, Sigmoid, and ReLU activation functions. In a CNN, it is usual to have an activation function following every convolution layer

to introduce non-linearity, the recommendation in modern NN is the use of ReLU activation functions(Goodfellow et al., 2016).

We will start by introducing some activation functions commonly used in the output layer - softmax and sigmoid.

2.3.2.1 Softmax

Anytime we want to represent a probability distribution over a discrete variable with K possible values we use the softmax function shown in Equation 2.5. It can be thought of as a generalisation of the Sigmoid function shown in Equation 2.6, which is used to represent it over a binary variable instead (Goodfellow et al., 2016).

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{2.5}$$

In NN, the softmax activation function is commonly used for multi-class classification, being an output layer that predicts the multinomial probability distribution described above.

As it can be seen in Figure 2.11, the softmax activation will output one value for each node in the output layer, this outputted vector of probabilities that sum to 1, is interpreted as the probability of membership for each class.

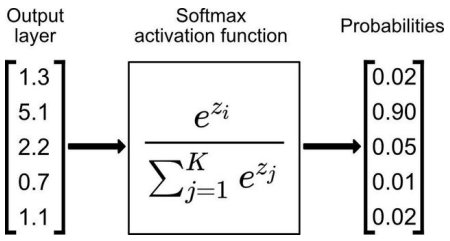


Figure 2.11: Softmax activation function (Nabiyev and Malekzadeh, 2021)

2.3.2.2 Sigmoid

Although the Sigmoid activation function was popular in the 90s as a non-linear layer to normalise the output of each neuron in hidden layers, it lost popularity (Goodfellow et al., 2016) to both the activation functions that follow.

These days, since it outputs values between 0 and 1, the sigmoid activation function is usually used in the output layer for single label/ multi-label binary classification problems.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

2.3.2.3 Tanh

The hyperbolic tangent activation function a.k.a. Tanh function was used as the default activation function for hidden layers in the late 90s to 2010s after it showed signs of typically performing better than the logistic sigmoid activation function (Goodfellow et al., 2016). Both of these activation functions are depicted in Figure 2.12.

As shown in Equation 2.7, it takes as input any real value and outputs values between -1 and 1.

$$Tanh(x) = \frac{1 - e^{-\alpha x}}{1 + e^{-\alpha x}} \tag{2.7}$$

Despite their ability to learn complex mapping functions, both the Sigmoid and Tahn activation functions have a known limitation, they saturate for extreme values of z, only being strongly sensitive to their input when z is near 0, which can be seen from Figure 2.12.

This problem is known as vanishing gradients, it makes it very difficult to know how the parameters should change to improve the cost function (Goodfellow et al., 2016) and consequently for Deep Neural Networks to learn effectively.

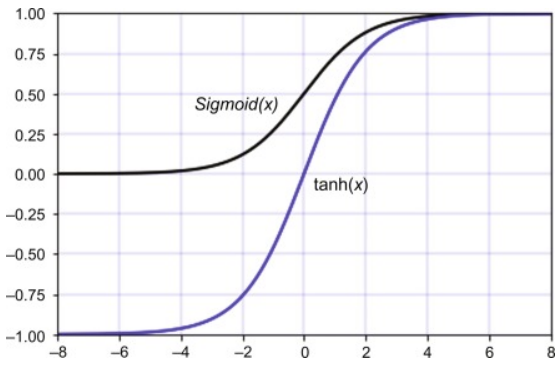


Figure 2.12: Tahn and Sigmoid activation functions (Fathi and Maleki Shoja, 2018)

2.3.2.4 ReLU

ReLUs were introduced to address the problem of vanishing gradients and quickly replaced the Sigmoid and Tahn activation functions by providing performance improvements, Deep CNNs with ReLUs trained several times faster than the same ones with Tahn units(Goodfellow et al., 2016).

The rectified linear activation function simply returns the input value provided if this value is greater than 0, or the value 0 if the input is 0 or less.

$$Relu(z) = max(0, z) \tag{2.8}$$

In this sense, it can be said that because ReLUs are nearly linear, they preserve properties that make linear models easy to optimise and generalise well (Goodfellow et al., 2016).

ReLUs do not come without limitations, large updates to the weights could mean that the input to the activation function is always negative, therefore the activation value will always be 0, this is known as the dying ReLU problem. This means the gradient is 0, so the unit will never activate, the weights will not be adjusted, so like the vanishing gradient problem the learning will be slow with constant 0 gradients (Maas et al., 2013).

This can be corrected by several variations of the ReLU, for example Leaky and Parametric ReLUs that change the slope to the left of $x < 0$. Either by a fixed parameter in Leaky ReLU as shown in Figure 2.13 or by a parameter that is learned by backpropagation using weights and biases in Parametric ReLU.

There are more complex examples like Exponential Linear Unit (ELU)s and Threshold ReLU functions that are known to improve accuracy compared to ReLUs.

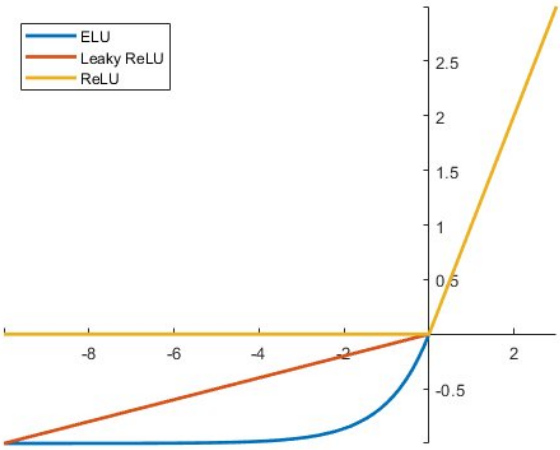


Figure 2.13: Rectified Linear activation functions (Shenoy, 2019)

2.3.3 Classical CNN Architectures and their evolution

An architecture is the arrangement of the layers introduced thus far into set patterns. The first CNN architecture was introduced by Lecun in the early 90s, followed by some other influential architectures that have built on each other and evolved since. This section describes these and their contributions.

2.3.3.1 LeNet

In 1989, LeCun et al. proposed the first multilayered CNN successfully trained via backpropagation. After a decade of improvement iterations, LeNet-5 (Lecun et

al., 1998) was the famous architecture depicted below that started the use of CNNs for Optical Character Recognition (OCR) tasks, but it didn't perform well in other computer vision problems.

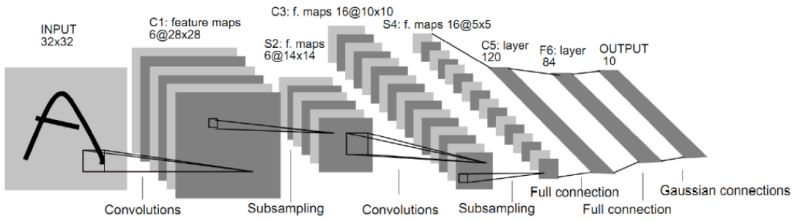


Figure 2.14: Architecture of LeNet-5 (Lecun et al., 1998)

2.3.3.2 AlexNet

In 2012, Krizhevsky et al.(Krizhevsky et al., 2012) took LeNet's work as inspiration to create a much deeper CNN and implemented a few novel contributions which are still essential to the success of CNNs to this day:

- **ReLUs** The benefits of these have been highlighted in section 2.3.2, contributing to faster training times than the Sigmoid activation function used in LeNet's implementation.
- **Dropout** The use of Dropout layers as a regularisation method helped avoid the problem of overfitting
- **Data Augmentation** Using artificial data augmentation techniques to increase the size and variety of the training dataset by translating and reflecting existing images improved the performance of the model
- **Training on a Graphic Processing Unit (GPU)** Using a couple of GPU to train AlexNet allowed for faster training on great amounts of bigger images which set a milestone for the success of CNNs, the delineation of responsibilities between the 2 GPUs is shown in Figure 2.15.

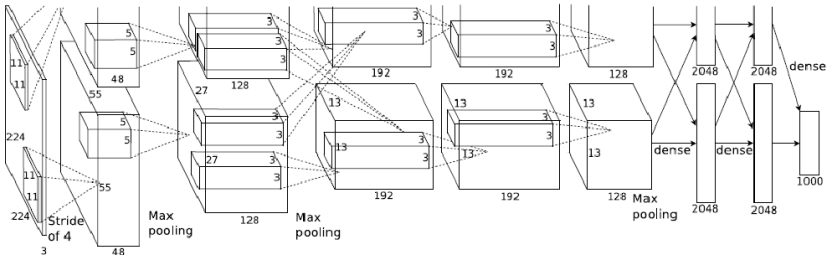


Figure 2.15: Architecture of AlexNet (Krizhevsky et al., 2012)

2.3.3.3 VGG

VGGNet (Simonyan and Zisserman, 2014) won the ImageNet Challenge 2014. This network is built on the simple principles of the two conventional CNNs introduced above, creating a deeper network (19 weight layers) with the idea of using blocks of an increasing number of filters. These repeated structures of a sequence of convolutional layers followed by a max-pooling layer for downsampling, create deeper networks with more non-linearities captured. Several other factors contributed to its success:

- **Filters with smaller dimensions** By using filters with a smaller receptive field of 3x3 and maintaining the number of filters in each of the convolutional layers in the same block, in a sequence of layers, more non-linearities can be captured using fewer parameters.
- **Increasing number of filters** By roughly doubling the number of filters in each block, more complex features can be captured
- **Scale jittering** Uses scale jittering as one of the data augmentation techniques.

2.3.3.4 Other Evolutions

Several meaningful contributions were made by other architectures, such as:

- **Network in Network (NiN)** A NiN block(Lin et al., 2014) applies a fully-connected layer to each pixel, they consist of a convolutional layer and multiple 1x1 convolutional layers, this design has influenced many other CNN designs.
- **GoogleNet** Also working with blocks, each of its "Inception"(Szegedy et al., 2015) blocks has 4 paths, extracting information in parallel through convolutional layers of different filter dimensions and max-pooling layers. It also makes use of the 1x1 convolution introduced in NiN to reduce channel dimensionality per pixel. This makes it a very efficient network architecture with a low computational cost.
- **Batch normalisation)** Introduced in 2015, this method (Ioffe and Szegedy, 2015) makes normalisation part of the model architecture by performing it for each training mini-batch. It performs reparametrisation of the model, reducing the need to coordinate updates across many layers (Goodfellow et al., 2016). It not only allows the use of much higher learning rates and more relaxed initialisation, but it also acts as a regularisation technique, as will be seen in section 3.2.3.5.
- **ResNet** This architecture (He et al., 2015a) challenges the convention of optimising the original unreferenced functions by optimising the residual mappings instead. This way the authors manage to create residual networks with a depth of 152 layers, which is 8 times deeper than the VGG net whilst keeping a lower complexity, making them more accurate and easier to optimise.

- **DenseNet** This architecture (G. Huang et al., 2017) builds on the concept of the ResNet but instead of adding inputs and outputs together in its cross-layer connections, it concatenates them instead. It also uses transition layers (1x1 convolution) to keep the dimensions under control.

2.4 The various architectures used in Remote Sensing applications

In the field of Remote Sensing applications, the goal is usually pixel-wise classification, which can be thought of as natural image segmentation. For this, the main state-of-the-art CNN-based segmentation models used in remote sensing have been introduced.

2.4.1 FCN

Long et al. (Long et al., 2015) proposed FCN to address the task of semantic segmentation. Having taken as base models some Classical CNN architectures introduced in Section 2.3.3, these were transformed from classifiers to FCN by swapping the fully connected layers with convolutional layers and dropping the final classifier layer.

Maggiori et al. (Maggiori et al., 2016) celebrates the improvements of using FCN over patch-based CNN for pixel-wise labelling in Remote Sensing imagery.

To achieve pixel-wise classification, at each of the coarse output locations a 1x1 convolution with a depth dimension of m, where m is the number of classes, is used to predict scores for each class, followed by the Fractionally-strided layer introduced in section 2.3.1 to bilinearly upsample the outputs to pixel-wise outputs.

The authors (Long et al., 2015) note that even though bilinearly upsampling was used in FCN, the convolution filter in such a layer need not be fixed to this but can be learned, emphasising that a stack of said layers and activation functions can even learn nonlinear upsampling.

Many of the architectures introduced below are extensions of the FCN architecture (Long et al., 2015), bringing new evolutions to the field of Semantic Segmentation namely Deeplab models (Chen et al., 2016), Conditional Random Field (CRF)-RNN (Zheng et al., 2015), ParseNet (Liu et al., 2015), U-NET(Ronneberger et al., 2015) and dilated convolutions (Yu and Koltun, 2016).

Despite this, FCN is still being used widely in remote sensing applications, it has been used together with a Digital Surface Model (DSM) on remote sensing imagery (Sun and Wang, 2018), for multi-label remote sensing image retrieval by extracting a segmentation map (Shao et al., 2020), for the classification of multi-source remote sensing data with Fusion-FCN (Xu et al., 2018) amongst others.

2.4.2 DeepLab models

Huang et al. (L. Huang et al., 2020) (L. Huang et al., 2021) uses DeepLabv3+ (Chen et al., 2018), the latest version of DeepLab, to classify each pixel that has an RTS present and to quantify their evolution, making it extremely relevant to this project.

The DeepLab models raise output resolution by using the "atrous" convolution instead of the classic one, doing the same with the density of the labels predicted for each class, Chen et al.(Chen et al., 2016) also use CRF to adjust region boundaries in post-processing and capture small details.

2.4.3 U-Net

U-Net (Ronneberger et al., 2015) was originally presented to solve medical imaging segmentation problems, since it identifies global image context while maintaining spatial accuracy. As shown in Figure 2.16, this is achieved by a combination of encoder, stacked convolution (conv 3x3, ReLU arrow), and max-pooling layers (max pool 2x2 arrow) that capture the context through the feature map and the decoder to enable that specific localization through transposed convolutions (up-conv 2x2 arrow). It introduced the bottom-up/top-down architecture with skip connections (copy and crop arrow) to reach the final result.

Yi et al.(Yi et al., 2019) use U-Net as a base for DeepResUnet, a residual learning adaptation for complex building segmentation using remote sensing imagery. Another application is the use of U-Net as the base for cloud detection with Cloud-AttU (Guo et al., 2020), which incorporates an attention mechanism to complete the cloud detection task in remote sensing images.

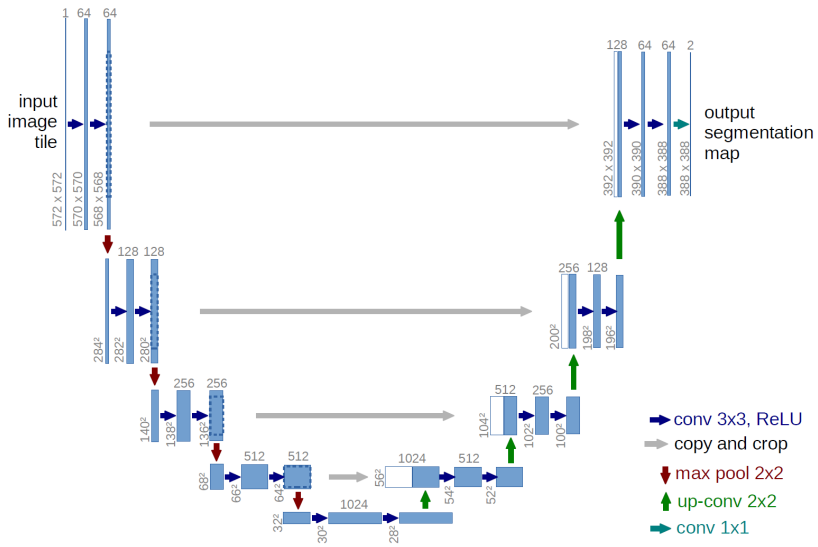


Figure 2.16: U-Net Architecture, blue boxes are feature maps and white boxes are copied feature maps (Ronneberger et al., 2015)

3

Semantic Segmentation Models - Input data, algorithm and hyperparameters

3.1 Input data

The input data is one of the essential aspects of the model, characteristics such as image dimensions (in terms of pixels WxH), the number of bands (D) and even the number of images used for training, validation and test can influence the speed and performance of the chosen model.

3.1.1 Labelled data

To be able to train a supervised DL model, the availability of labelled data is essential in the process of training the model, this ground truth is the key in calculating the error and updating the weights of the model, until its loss and performance metric show good performance.

Thaw Slump Shape files The ground truth RTS polygons were very kindly provided by Dr. Ingmar Nitze and the AWI. These were collected across pre-defined sites in different locations at different timestamps and georeferenced for consistency and ease of use.

Two different types of shape files containing RTS polygons were provided:

- **Merged thaw slumps first iteration** Delineated by students and awaiting validation from Dr. Nitze, this type contains a total of 8,005 features.
- **Merged thaw slumps validated** Delineated by students and validated by Dr. Nitze, this type contains a total of 1,203 features.

3.1.2 Sentinel-2 data

As mentioned in Section 1, the primary data source for this project will involve the use of Sentinel-2 satellite imagery. There are two product types available to users,

Level 1-C Top Of Atmosphere reflectances and level 2-A Bottom Of Atmosphere reflectance images, both in cartographic geometry (Agency, n.d.).

The latter is the one used in this project, as it is pre-processed to minimise atmospheric effects, leading to better normalisation across different regions. There are 13 spectral bands available, from the Visible Near Infra-Red (VNIR) and Near Infra-Red (NIR) all the way to the Short Wave Infra-Red (SWIR) (Agency, n.d.).

There are four 10 m resolution bands: the three RGB bands - Blue (≈493 nm), Green (≈560 nm), and Red (665 nm) plus a NIR (≈833 nm) band which is very important for vegetation detection. These are the ones initially used in this project.

In addition, there are six 20 m bands: 4 narrow bands in the VNIR vegetation red edge spectral-domain (≈704 nm,≈740 nm, ≈783 nm, and ≈865 nm) and 2 wider SWIR bands (≈1610 nm and ≈2190 nm) commonly used for snow/ice/cloud detection, or vegetation moisture assessments. Finally, two 60 m bands mainly focused on cloud screening and atmospheric correction (≈443 nm for aerosols and ≈945 nm for water vapour) and cirrus detection (≈1374 nm).

3.1.3 Sentinel-2 data collection

GEE and JavaScript

GEE provides the ability of extracting satellite data using bespoke JavaScript queries through a Console UI (shown in Figure 3.1, the Sentinel-2 data is available in the GEE image collection with ID *COPERNICUS/S2*.

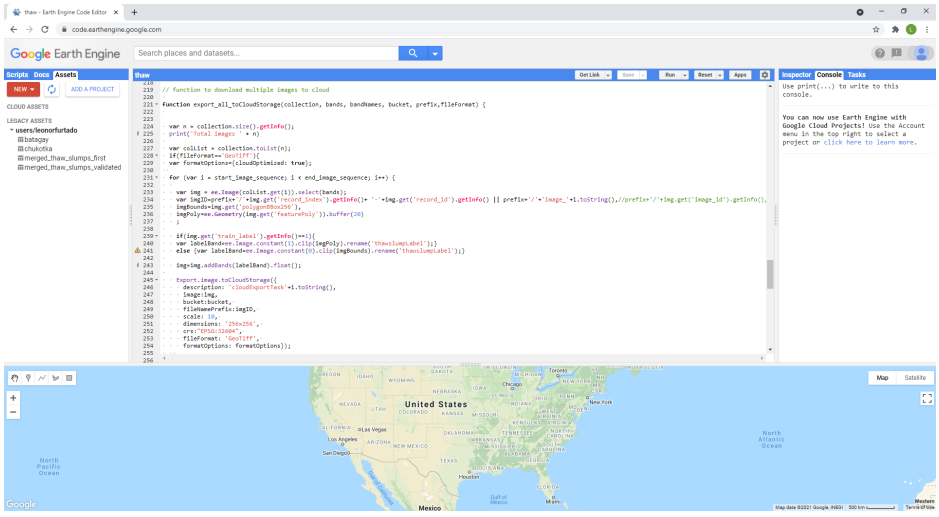


Figure 3.1: GEE Console UI showing a snippet of the bespoke JavaScript script

A bespoke JavaScript script was created that extracts satellite data for a fixed area and stores it in Google Cloud Storage (GCS), this method of data collection will hereby be referred to as *GEE data*. This is an area of 2.56 km^2, so that the input size is close

to 256x256 pixel representation, since for the highest resolution band one pixel represents a 10x10 m area.

For positive label, when a (retrogressive) thaw slump is present in the area the data is extracted so that the RTS polygons presented in section 3.1.1 are at the centre of the image for the extraction. For negative labels, control images are extracted based on 2.56 km^2 squares centred around random points, that are checked for intersection with RTS polygons and any overlapping random features are removed.

This method has the limitation of introducing bias towards positive pixels always being in the centre of the image. This is a type of selection bias, as the subsample of images the model is being trained on does not represent the overall population of images, leading to poor generalisation of the model. For example, if the positive RTS pixels are anywhere else but the centre of the image, the model would display poor performance

All images have the 10m resolution bands (RGB, NIR channels) plus thawslumpLabel.These correspond to the original Sentinel 2A bands B4, B3, B2 (RGB),NIR (B8) plus an additional thaw slump mask.This RTS mask indicates pixels within thaw slump polygons, plus a 20 m (approximately 2 pixels) edge buffer, with value 1 and all other pixels set as 0 in other parts of the image (in control images this band is all 0s).The reason why only these 4 bands were take was due to the extra challenge of resampling other 20 meter or 60 meter resolution bands to 20 meter or vice-versa, given the limited timeframe of this project, doing this would have introduced extra complexity to the problem.

Each satellite image has been collected to match the polygon image date as closely as possible, within a +/− 2-week window. This range is wide as the images extracted exclude any images with cirrus and clouds. This dataset will be hereby referred to as *GEE data*. Files are in GeoTiff format with float32 pixel values that have not been scaled, so are in raw format at this stage, these values returned by the GEE API that can reach values of 10,000 or more in some instances. This is the case as even though reflectances are typically percentage values between 0 and 100 (or 0-1), these are often scaled by a certain factor (e.g. $1e^4$) to be able to use integer values instead to save disk space.

The Coordinate Reference System (CRS), which associates numerical coordinates with a position on the Earth's surface, is defined by organisations such as the European Petroleum Search Group (EPSG).

The projections are identified through an authority:code format, where the authority is for example "EPSG" and the code is a unique number for a given CRS. For example, EPSG:32604 was used for this project's data extraction for RTSs located in Herschel Island.

This method of data collection came with its limitations, the GEE UI would crash for more than 30 images extracted at any one time and would freeze while extracting those images, taking hours at a time. This method wasn't automatable, which explains

why the number of images collected is so low compared to the number of features in the polygons provided as ground truth.

GCS and JP2 data

To address both the bias and automation limitations of the GEE method, downloading images directly from GCS was explored latter in the project.

Sentinel-2 data is made free and available on Google Cloud Platform (GCP) by the European Commission and the European Space Agency(ESA) as part of the Google Public Cloud Data program. The data is made available in a GCS public bucket (*gs://gcp-public-data-sentinel-2*) in the JP2 format, it is stored as tiles of approximately 100 km^2, according to the Sentinel-2 tiling grid (based on the Military grid reference system) at a particular point in time, referred to as a *granule* (Cloud, n.d.).

This method involved downloading granules that contained sites used for the collection of the merged thaw slumps validated labelled data presented in section 3.1.1 at the time the labels were delineated +/- 1 month as a filter to guaranty low (0 – 10%) cloud coverage, was applied.

The polygons were reprojected from EPSG:4326 to the respective CRS associated with each of the extracted tiles, data was then consolidated in a data cube containing the same 10 m resolution bands (RGB, NIR channels) shown in Figure 3.2 after normalisation, plus the thaw slump mask shown in Figure 3.3. Dta was split using sliding windows of 64x64 pixels,after this any empty tiles, ones without any positive RTS pixels and ones with unexpected reflectance patterns were discarded. This pre-processed input data was kindly provided by Adam Wickenden as part of Accenture's collaboration with SEI and AWI and will be hereby referred to as *JP2 data*.

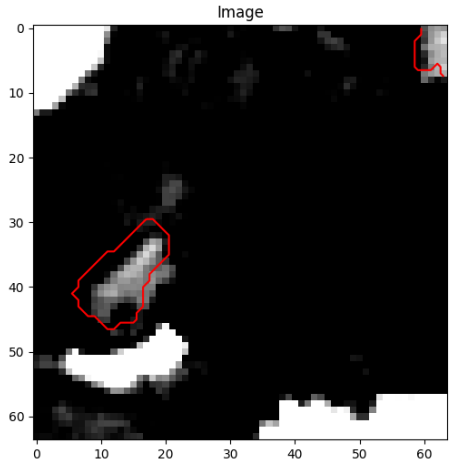


Figure 3.2: Example of the RGB channels normalised using z-score

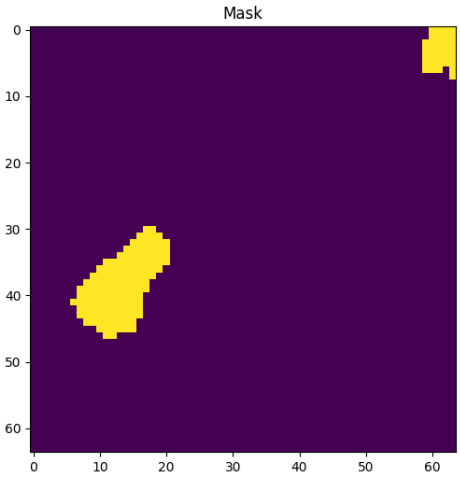


Figure 3.3: Example of a RTS mask

3.1.4 Data understanding and preparation

3.1.4.1 Scene classification labelled data

The original *GEE data* collection method extracted a total of 313 scenes containing RTS positive samples, and 311 control negative samples containing only background pixels. These were split into train and validation datasets with a 80/20 proportion respectively. A stratified method was used to ensure class representation and avoidance of class imbalance in any one dataset. These datasets were used in the training of the initial scene classification model introduced:

Dataset	Thaw Slump	Control
Train	250	249
Validation	63	62
Total	313	311

Table 3.1: Distribution of GEE data for scene classification

3.1.4.2 Pixel-wise classification labelled data

In terms of pixel distribution in each image, there is an imbalance for semantic segmentation models, that are given proportionally more negative background pixels than positive RTS pixels. Figure 3.4 depicts the distribution of pixels with RTS positive label in the *GEE data*, as it can be seen from the figure most images have less than 500 pixels of the positive class which is less than 1% of a 256×256 pixel image.

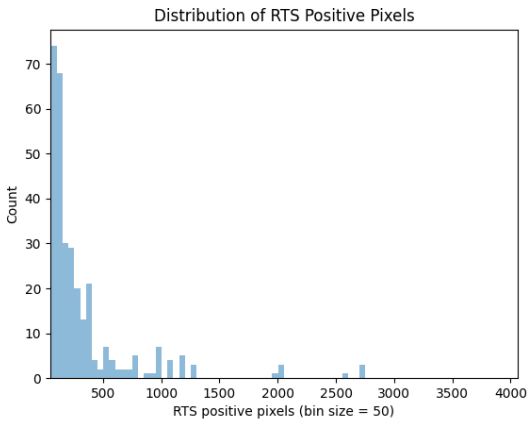


Figure 3.4: Distribution of GEE data RTS positive pixels

To train the segmentation models, only the 313 images in the *GEE data* containing positive RTS pixels were used for training, to avoid further imbalances in the model towards background negative pixels, for training the data was split as follows:

Dataset	Thaw Slump	Percentage of total
Train	250	80%
Validation	47	≈ 15%
Test	16	≈ 5%
Total	313	100%

Table 3.2: Train/ Validation/ Test split of samples for GEE data

Later in the project the *JP2 data* was introduced with a total of 585 64x64 slices containing positive RTS pixels that can be used for the supervised DL model it's distribution is as follows:

Dataset	Thaw Slump	Percentage of total
Train	468	80%
Validation	90	≈ 15%
Test	27	≈ 5%
Total	585	100%

Table 3.3: Train/Validation/Test split of samples for JP2 data

Figure 3.5 depicts the distribution of pixels with RTS positive label in the *JP2 data* across different datasets, to attempt a stratified sampling method the random seed was changed until the distribution looked similar across the datasets.

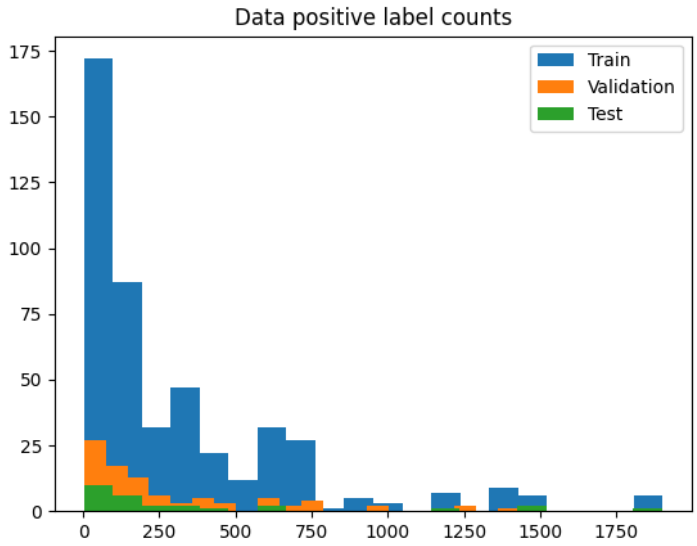


Figure 3.5: Distribution of JP2 data RTS positive pixels across datasets

3.2 Introduction to semantic segmentation models

Given the literature on remote sensing DL problems, along with the labelled RTS data available in a mask format, where each pixel is labelled, it makes sense to make the focus of this project the semantic segmentation networks based on CNN models that were introduced in section 2.4.

These different architectures will be evaluated in experiments, as well as the architecture parameters, that is, Kernel sizes, stride, number of layers, number of filters etc. which nowadays are also considered "hyperparameters"that feed into the learning process described in section 3.2.3, since these were covered in section 2.3.1.

To evaluate the success of the task at end in pixel-wise classification models in remote sensing, the ground truth is imputed as a mask of values, 1 for thaw slump and 0 for background. The other input is the satellite image itself. To bring the best out of the model architecture, this data needs to be pre-processed to be turned into the input that the network is expecting, usually in tensor form with values between 0 and 1.

The model architecture itself also plays an important part in the success of the model, its complexity/depth as well as the architecture parameters described above need to be adequate for the problem at hand.

In addition to this, the network hyperparameters also need to be defined before the training can start. After this is done, the training process can start where the model parameters i.e. weights are optimised so that the predictions of RTS or no-RTS performed by the model match the mask ground truth as closely as possible.

3.2.1 Pre-processing of data

3.2.1.1 Resizing images

In segmentation models, the spatial size of the input tensor has an impact on the performance of the model. With lower dimension sizes a more complex architecture could be required (higher number of filters), however with greater dimensions more training samples are likely to be needed.

So it is important to have input image dimensions (W x H) that are adequate for the problem at hand, especially when the RTS only covers a small area (a few pixels, since each pixel represents a 10m x 10m area.

There are a few ways of increasing the dimensions of an image through different interpolation strategies, some of them are:

- **Bilinear interpolation** mentioned before as one of the segmentation model upsampling strategies, it takes the weighted average of the 4 nearest pixels for the new pixel value
- **Bicubic interpolation** similar to bilinear, but instead uses the 4x4 neighbourhood i.e. 16 pixels to calculate the new value, again weighted more on the closest

pixels.

- **Nearest neighbour** simply returns the value of the nearest pixel according to the interpolation coordinates.

3.2.1.2 Normalisation techniques

Unlike natural images that have a known value between 0 and 255, remote sensing imagery depends highly on the scale of the reflectance values returned by the GEE API varying widely and reaching values of over $10,000$ in some instances, therefore the normalisation of input becomes quite important.

To feed input values into the NN it is better to normalise them between $[0,1]$, this is particularly challenging in remote sensing imagery as the spectral data has a big variation of reflectance values across bands and images.

A naïve approach of simply converting any pixel $x > 10,000 = 10,000$ and then dividing by $10,000$ was the first method attempted, given that most band's reflectance values go up to $10,000$ with a few outliers above $10,000$.

There have been a couple of strategies presented in a few remote sensing papers that could be possible solutions.

In other remote-sensing imagery normalisation the authors normalise the values per spectral band (Benedetti et al., 2018) using in the interval $[0,1]$, others use the z-score to approximate the distribution of pixel values of the input to a normal distribution (Zhong et al., 2017).

They are both converted to spectral band-wise normalisation using the below definitions (Belenguer-Plomer et al., 2021):

$$interval[0,1](x) = \frac{x}{max(b)} \tag{3.1}$$

$$z-score(x) = \frac{x-\mu(b)}{\sigma(b)} \tag{3.2}$$

where x is any given pixel and b is any given spectral band of the image.

3.2.1.3 Data Augmentation

Due to the biased data collection method described in section 3.1.3, making the RTS positive pixels the centre of the training patch, it is essential to apply data augmentation techniques that will vary this feature of the training set, so that the model is not biased towards finding positive labelled pixels only at the centre of the image provided.

For that purpose and also the general purpose of introducing noise in the model to make it more robust and less subject to overfitting due to the small sample size, the following data augmentation techniques are presented:

- **Random horizontal or vertical flipping** Randomly flips the image and mask horizontally or vertically.
- **Cropping** randomly crops the image and mask
- **Scaling** randomly rescales the image by shrinking it or stretching it keeping the same ratio, in this case a square image.
- **Blurring** different satellites resolution could introduce blurring, so it is important to introduce that to make the model more robust.
- **Shifting** shifts the coordinates of every pixel along the image axis, important to avoid that centred Region of Interest (ROI) bias.

3.2.2 Preparing the data for modelling using Tensorflow

A bespoke script was created to code a data generator compatible with Tensorflow models there are a few steps in the model:

1. Read GeoTiff file and convert it to tensor format
2. Select the m bands for X and thaw slump label/mask for y
3. Image normalisation technique applied
4. Generates batches of data for training

3.2.3 DL Learning process

To successfully train any NN, there are a few key "ingredients"that are necessary to ensure any given model with its own architecture and parameters to enable it to accurately perform the task of matching its predictions with the ground-truth. These will be covered in this section:

1. **Loss/cost function** that evaluates how well the model performs on any given task, where the goal is usually to minimise the loss.
2. **Activation function** will define how the activation map is calculated after convolution and fully connected layers. These were covered in detail in section 2.3.2, the ones most widely used in pixel-wise classification architectures in remote sensing are ReLU and its variants.
3. **Optimiser** responsible for updating the model parameters at each iteration to optimise the cost function. To learn the model parameters (a.k.a. weights in a NN) efficiently, essential hyperparameters that estimate the parameters need to be adjusted "manually"as depicted in Figure 3.6:

a) **Batch Size** determines the frequency of updates influencing convergence and generalisation.

b) **Initialiser** a good initialisation can accelerate optimisation and enable convergence.

c) **Regulariser** introduces noise to reduce overfitting.

d) **Learning Rate** influences the optimisation's convergence.

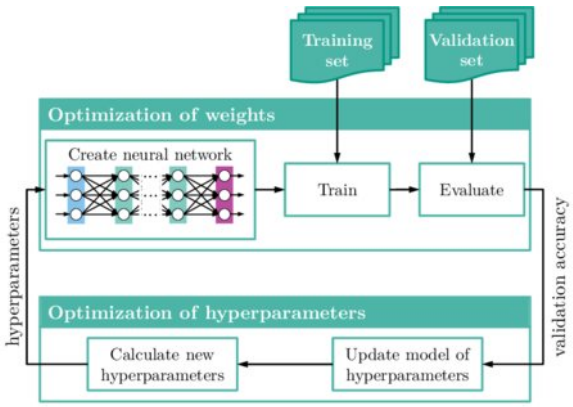


Figure 3.6: NN hyperparameter optimization cycle (Stock et al., 2020)

3.2.3.1 Loss function

The loss function is one of the essential components in the learning process of DL models, since it enables the DL algorithms to learn and optimise the objective through gradient descent. The loss function is responsible for ensuring that the mathematical representation of the objectives is accurate, so that it can assess how well the predicted values match the ground truth.

Ma (J. Ma et al., 2021) breaks down known segmentation loss functions into categories according to different optimisation objectives, which can be seen in Figure 3.7. Some of these will be described in this section.

Binary Cross Entropy (CE)

CE is an example of a distribution-based loss, it is derived from Kullback-Leibler (KL) divergence, which measures the dissimilarity of two distributions(J. Ma et al., 2021). This makes the objective of this category of loss to minimise this dissimilarity.

For this project, the most relevant CE for binary classification tasks is the Binary CE, which can be defined as follows:

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\widehat{y_i}) + (1-y_i)\log(1-\widehat{y_i})] \tag{3.3}$$

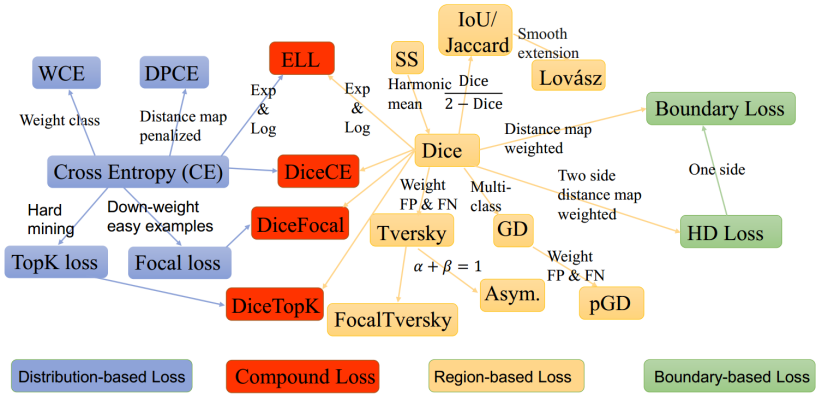


Figure 3.7: Overview and relationship among the existing loss functions. (J. Ma et al., 2021)

As in this project's data positive RTS pixels are always underrepresented compared to background pixels, therefore the DL network can get stuck in local minima, triggering early stopping and be very biased towards the background pixels. It is relevant to mention re-weighted loss functions, where positive pixels get more importance (Ronneberger et al., 2015) as a potential solution.

Focal loss can be highlighted as a potential derivation to deal with extreme pixel class imbalance, as it reduces the relative loss of well-classified pixels by adding a factor to standard CE, *binary focal loss* is implemented as a custom loss function in section 4.3.7.

All the other functions depicted under the distribution-based loss in Figure 3.7 are derivations of CE, therefore will not be covered in detail.

Dice Loss

Dice Loss is the key element of region-based loss functions, which aim to minimise the mismatch between predicted $\widehat{y}$ and ground-truth y regions (J. Ma et al., 2021). It can directly optimise the Dice coefficient, which is defined in section 3.2.4, this was implemented as a custom loss function hereby known as *dice coef loss*.

$$L_{dice} = 1 - \frac{2\sum_i^N \widehat{y}_i y_i}{\sum_i^N \widehat{y}_i^2 + \sum_i^N y_i^2} \tag{3.4}$$

IoU loss

The IoU a.k.a. Jaccard index is used to directly optimise the segmentation class metric, it can be seen as a similarity measure between two sets A and B is defined as

the following:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{3.5}$$

$$0 \leq J(A,B) \leq 1$$

However, the IoU is not differentiable, it can nonetheless be generalised for predicting probabilities, making it differentiable and therefore suitable for optimisation by gradient descent and allow for backpropagation in a NN. This is implemented as a custom loss function and referred to later as *IoU loss*.

$$L_{IoU}(y,\widehat{y}) = 1 - \sum_{i=1}^{n} \frac{y_i \cdot \widehat{y_i}}{y_i + \widehat{y_i} - y_i \cdot \widehat{y_i}} \tag{3.6}$$

It has been argued that IoU loss is better for binary segmentation than those trained with the standard softmax loss(Rahman and Wang, 2016), making it a very relevant loss function for this project.

Compound loss

There are many compound losses which combine and transform some loss functions introduced above. A Compound loss function, inspired by a paper (Vladimir Iglovikov, 2017) that applies semantic segmentation to satellite imagery, appears to also be a feasible solution. In this paper, their compound loss function is defined as a combination of Eq.3.3 and Eq.3.6, this will be referred to as *ce jaccard loss* during implementation:

$$L_{CEIoU} = L_{BCE} - \log(\frac{1}{n}(-L_{IoU} + 1)) \tag{3.7}$$

Another compound loss function used in this project, is a combination of Eq.3.3 and Eq.3.4, which will be referred to hereafter as *CE dice loss*:

$$L_{CEDice} = L_{BCE} + L_{dice} \tag{3.8}$$

3.2.3.2 Optimiser

In Deep Learning, optimisation refers to the process of finding the parameters θ of a NN that attempt to minimise a loss function $J(\theta)$ (Goodfellow et al., 2016), this process usually involves not only using usually a subset of data to calculate each update to the parameters based on an expected value of the cost function, but also taking into account any regulariser, weight initialisation, learning rate and batch size which influence the success of the optimisation strategy.

Most algorithms used for DL use more than 1 but less than all the training examples a.k.a. mini-batch, even though they don't use only one example they are nowadays referred to as just stochastic methods rather than mini-batch stochastic methods. In

a similarly confusing way, even though that Batch Gradient Descent (BGD) implies the processing of the whole training set, the term batch size is usually referring to the mini-batch size, which Goodfellow et al.(Goodfellow et al., 2016) clarifies very well.

Stochastic Gradient Descent (SGD)

SGD and its variants are probably the most popular algorithms in DL as it can converge even when the training dataset gets very large, however it has slower asymptotic convergence then BGD for example.

SGD aims to get an unbiased estimate of the gradient by averaging the gradient in the mini-batch drawn randomly from the training data and follow the gradient downhill (Goodfellow et al., 2016).

The learning rate is an essential parameter in the gradient descent process, it is advised that it gradually decreases during training time, through strategies described in more detail in section 3.2.3.6. It is known that stochastic gradient descent has slower asymptotic convergence than BGD for example.

Momentum

Momentum has been designed to overcome SGD's limitations by accelerating learning, it accumulates an exponentially decaying moving average of previous gradients and keeps moving in their direction (Goodfellow et al., 2016), creating SGD with momentum.

Momentum helps accelerate SGD in the right direction and reduces oscillations (Ruder, 2017), defined by a new momentum hyperparameter α between 0 and 1, determines how quickly this moving average decays.

Common values of α used in practice are 0.5, 0.9, and 0.99, for the semantic segmentation models introduced in section 2.4, the most common momentum parameter value is approximately 0.9 (Sultana et al., 2020)

The next few variations of stochastic gradient descent have adaptive learning rates, so it is recommended not to change the parameters in these, therefore the learning rate section 3.2.3.6 is most relevant for the preceding optimisers.

Root Mean Square Propagation (RMSProp)

RMSProp, is an unpublished optimiser created by Tijemen Tieleman and Geoffrey Hinton in 2012. It divides the learning rate by an exponentially decaying average of squared gradients to discard history from the extreme past so that it can converge rapiDLy, it has been shown to be effective and practical for DL problems (Goodfellow et al., 2016)

Adaptive Momentum Estimation (Adam)

The Adam optimiser (Kingma and Ba, 2017) is a variation of the combination of momentum, which points the model in a better direction and RMSProp, which adapts how far the model goes in the direction to converge quickly.

Unlike the previous algorithms, Adam includes bias correction to both the first-order moments (the momentum term) and the (uncentered) second-order moments (also in RMSProp) making Adam fairly robust to the choice of hyperparameters (Goodfellow et al., 2016).

The authors (Kingma and Ba, 2017) show that Adam works well in practice and compares well with other adaptive learning-method algorithms.

Nesterov-accelerated Adaptive Momentum Estimation (NAdam) Optimiser

NAdam showed dramatic improvements in this paper(Dozat, 2016) the authors argue that it is a no-brainer to incorporate Nesterov momentum into Adam, specially where combining momentum and RMSProp is concerned, this was actually present in Chapter 8 of Goodfellow et al.'s book (Goodfellow et al., 2016).

It was also derived and explained by Ruder(Ruder, 2017), who also emphasizes that Nesterov Accelerated Gradient (NAG) is superior to vanilla momentum, thus combining Adam and NAG seems to make sense theoretically.

Despite this claim, not many other evidences of the use of NAdam where found in practice. Nonetheless, a technical report (Vladimir Iglovikov, 2017) uses NAdam as the optimiser to train a multi-spectral image U-Net to accurately perform semantic segmentation of different classes in satellite imagery, making it quite relevant to this project.

3.2.3.3 (Mini-)Batch size

Batch size is the number of data points used to train a model in each iteration, it is very important to ensure that the algorithm converges, since it determines the frequency of updates of the network.

The larger the batch size, the more accurate cost gradient w.r.t. parameters and faster training speed (Katanforoosh, 2019). Large batch sizes can however negatively impact the generalisation of the network (Keskar et al., 2017).

Common mini-batch sizes range between 50 and 256, but vary for different applications (Ruder, 2017), for the semantic segmentation models introduced in section 2.4 it tends to be between 12 and 20 images, with for example U-Net (Ronneberger et al., 2015) as one of the exceptions being 1 image making it a pure stochastic gradient optimisation process.

3.2.3.4 Initialisation

Some algorithms are very sensitive to weight initialisation and the success of its convergence depends a lot on the chosen initialiser. To prevent gradients from vanishing (weight initialisation too small) or exploding (too large), as a rule of thumb the mean of the activations should be 0 and the variance of the activations should stay consistent across all layers.

For simplicity, let's assume the same initialisation methods are usually applied to both the forward propagation (activations) and backward propagation (cost gradients with respect to (w.r.t.) activations).

Random normal initialiser The weights are randomly initialised usually from a zero-mean normal distribution, however other values of mean and standard deviation can be defined.

It has been used with the default configuration of zero mean and unit standard deviation in remote sensing applications by Kemker et al. (Kemker et al., 2018) in initialisation experiences Deep NN for Semantic Segmentation.

Uniform initialiser In Uniform initialisation, known to work well with the sigmoid activation function (see equation 2.6), the weights belong to a uniform distribution in range (a, b).

"Xavier"(Glorot) initialiser The "Xavier"(Glorot and Bengio, 2010) initialisation, known to work well with the tanh activation function (see equation 2.7), picks the weights of layer l randomly from a normal distribution with mean $\mu = 0$ and variance $\sigma^2 = \frac{1}{n^{[l-1]}}$, where $n^{[l-1]}$ is the number of neurons in layer $l-1$. Biases are initialized with zeros:

$$W^{[l]} \sim \mathcal{N}(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}})$$

$$b^{[l]} = 0$$

There is also a "Xavier"uniform variation, where the weights are chosen from a random uniform distribution, which won't be covered in detail.

"He"(Kaiming) initialiser In an attempt to discover the best initialiser to work well with ReLU (see equation 2.8 like activations, He et al. (He et al., 2015c) adapted "Xavier"initialisation to achieve better performance with rectified units, this proved quite successful as seen in Figure 3.8.

In "He"initialisation, weights of layer l are initialised with a zero-mean Gaussian distribution with a standard deviation $\sigma = \sqrt{\frac{2}{n_l}}$, where n_l is the number of activations of layer l. Biases are also initialized with zeros.

The difference between this "He"and the Xavier initialisation is the $\frac{1}{2}$ that comes from the ReLU activation function.

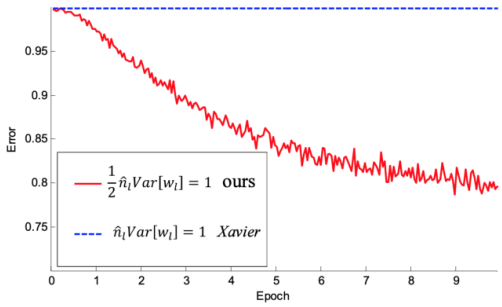


Figure 3.8: Convergence of a 30-layer CNN in He et al.'s paper (He et al., 2015c)

3.2.3.5 Regulariser

When fine-tuning pre-trained models, it is very likely that overfitting will become a problem. To help address this, either $l1$ or $l2$ regularisation can be helpful with early stopping, as well as, the introduction of Dropout or Batch normalisation.

$l1$ and $l2$ regularisation Performing $l2$ regularisation a.k.a. weight decay constrains weigh values towards 0 (but not actually zero) whereas $l1$ regularisation a.k.a. Lasso regression drives the weights to be 0, tends to work well as a feature selection technique.

Regularisers are usually attached to the loss function as a penalty term, $l1$ penalises the absolute value whereas $l2$ penalises the square value of the weights (Shanmugamani, 2018).

$l2$ is more popular, as $l1$ can remove relevant information from high-dimensional data where these are correlated, leading to underperforming models.Most of the segmentation models introduced in section 2.4 use weight decay with values between 0.0001 and 0.0005 (Sultana et al., 2020).

Dropout The idea of Dropout (Srivastava et al., 2014) is to randomly turn off neurons (along with their connections) with some probability p from the NN during training, this is usually applied at each step of forward propagation and weight update step, neurons turned off in one step can become active in the following step.

This can make this method quite computationally expensive and less effective when there are very few labelled training examples, however it does have the advantage of being (Goodfellow et al., 2016).

In Figure 3.9 it can be seen that after dropout is applied in 3.9b the network becomes less complex network subsampled from the original NN in 3.9a.

Batch Normalisation Its main purpose as discussed in section 2.3.1 is to make optimisation better through reparametrisation, this introduces both additive and multiplicative noise.

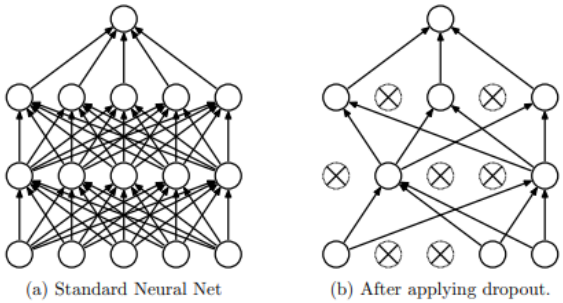


Figure 3.9: (a) Regular 2-layer NN (b) Example of NN after dropout applied. Crossed neurons have been turned off. (Shanmugamani, 2018)

This noise introduced in the hidden units during training can have a regularisation effect, so that there is no need for Dropout (Goodfellow et al., 2016).

Early Stopping The use of early stopping is probably one of the most efficient and easy to implement regularisation methods in deep learning applications(Goodfellow et al., 2016). It involves monitoring the parameters and validation set error, so that the best parameters at the point of the lowest validation error can be returned when the training stops.

The training stops when none of the parameters have improved over the best validation error stored (when monitoring validation loss) for a certain number of iterations (a.k.a. patience parameter).

It is argued that early stopping is advantageous over weight decay as it automatically figures out the correct amount of regularisation while weight decay needs several training experiments to ensure the networks doesn't get trapped in a local minimum.(Goodfellow et al., 2016)

3.2.3.6 Learning Rate

Goodfellow et al. (Goodfellow et al., 2016) suggests that the best way of choosing an initial learning rate is to monitor leaning curves, plots of loss function as a function of learning time, usually epochs.

There are a couple of things to watch out for, big oscillations where the loss increases drastically indicate the learning rate is too high. Small oscillations are okay, specially if the loss function contains stochastic features such as dropout.

If the initial learning rate is too small, the function might be stuck with high loss values, getting stuck and potentially triggering early stopping. As a rule of thumb, it is advised that the initial learning rate is higher than the best performing rate after approximately 100 iterations(Goodfellow et al., 2016).

Linear Decay It is common to decay the learning rate until it makes a couple of hundred passes through the training dataset (until iteration τ)(Goodfellow et al., 2016). Therefore, we define learning rate at iteration k as ϵ_k:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{3.9}$$

where $\alpha = \frac{k}{\tau}$. After iteration τ , it is usual to leave ϵ constant.

Step Decay Step decay drops the learning rate by a factor every few epochs. For example, in the DeepLab model's original paper (Chen et al., 2016) uses a $\epsilon_0 = 0.001$ and an $\epsilon_\tau = 0.01$ in the final classification layer. It drops it 10% by multiplying it by 0.1 at every 2000 iterations.

Exponential Decay As the name implies, an exponential decay function is applied to the initial learning rate ϵ_0:

$$\epsilon_k = \epsilon_0 e^{-\alpha k} \tag{3.10}$$

where α is the exponential decay parameter.

For example, in a paper that introduces the FCN-DenseNer103(Jegou et al., 2017) the authors use $\epsilon_0 = 0.001$ with an exponential decay of 0.995 every epoch.

Polynomial decay policy ("poly") Starting with ParseNet (Liu et al., 2015), where the authors improved the performance of their model by 1.5% with the same iterations (80k), $\epsilon_0 = 1e^{-}9$ and $power = 0.9$, it is proved to converge faster than step decay.

ParseNet inspired the authors of Deeplab model's follow-up paper in 2017 (Chen et al., 2017) to also use "poly" ($power = 0.9$), using the same batch size and training iterations they also achieve better performance (1.17%)than "step", reinforcing the trend.

The creators of the Pyramid Parsing Network (PSPNet) (Zhao et al., 2017), another popular semantic segmentation model, yet again inspired by DeepLab, also use 'poly' learning rate with $\epsilon_0 = 0.01$ and $power = 0.9$. This popular learning rate policy among state-of-the-art semantic segmentation model is defined in Equation 3.11 below:

$$\epsilon_k = \epsilon_0 (1 - \frac{k}{T_k})^{power} \tag{3.11}$$

where T_k is the total number of iterations and $power$ term controls the shape of the learning rate decay. In case of polynomial learning rate policy, T_k is equal to total number of epochs times number of iterations in an epoch (Mishra and Sarawadekar, 2019).

There are many other learning rate policies such as the constant learning rate, manual step learning rate and cosine decay learning rate that are self-explanatory, so will not be covered in detail.

3.2.4 Measurement of model performance

During training, it is usual to monitor certain metrics that measure how well the model is performing, ones typically used in semantic segmentation problems are outlined in this section.

Dice coefficient The dice coefficient (Milletari et al., 2016), ranging between 0 and 1, is the most commonly used segmentation evaluation metric. The Dice coefficient ($Dice$) between 2 binary volumes, which the aim is to maximise, can be written as:

$$Dice = \frac{2\sum_i^N p_i g_i}{\sum_i^N p_i^2 + \sum_i^N g_i^2} \tag{3.12}$$

IoU From the definition of the two metrics, we have that IoU and Dice score are within a factor of 2 of each other $F/2 \leq IoU \leq F$. It has been defined in section 3.2.3.1 with Equation 3.5

In general, like $l2$ penalises the biggest mistakes more than $l1$ regularisation, the IoU metric penalises a single instances of wrong classification more than the Dice score. The IoU metric has a "squaring"effect on the errors in comparison with the Dice score, making it more focused on the worst performance, rather than the average.

3.2.5 Transfer Learning

Due to the small training set, it is advisable to use pre-trained weights on other datasets to accelerate the training process, this is known as transfer learning and it as become a widely used technique in accelerating progress in the field of Computer Vision (CV).

It is the process of initialising a model using the weights of another model pre-trained on a much larger dataset, this usually ensures faster convergence. (Shanmugamani, 2018)

As a rule of thumb, one can either use the pre-trained model as is, or pick and choose which layers to re-train or fine-tune depending on the problem at hand as outlined in table 3.2.5.

Sample size	Dataset Similarity	Dataset Difference
Small Data	Fine-tune output layers	Fine-tune hidden layers
Big Data	Fine-tune whole model	Re-train model

4

Modelling Experiments

In this chapter, experiments will be performed to identify the best pre-processing and training parameters to increase the classification performance of the model of choice. All the below experiments were carried out using this author's personal computer with Intel(R) Core(TM) i7-10750H Central Processing Unit (CPU) @ 2.60GHz, 16GB Random Access Memory (RAM) with a GeForce RTX 2070 with Max-Q Design 8GB GPU

4.1 Choosing the most appropriate classification model

4.1.1 Pixel-wise classification model

After framing the problem as a scene classification problem and getting quite good results, in the search for a more challenging problem and due to the availability of data labels for each pixel form polygons it was decided by the author to frame the problem as a pixel-wise classification problem.

The decision to use a U-Net architecture as baseline came from a lot of the literature on remote sensing images being modified U-Net architectures, below is a description of the U-Net architecture this project implements as a baseline.

In this project a **convolution block**(x, z) consists of:

1. **2D convolutional layer with** x **channels** with 3x3 filter dimensions, stride of 1 and "same"padding so that if you use a stride of 1, the layer's outputs will have the same spatial dimensions as its inputs.
2. **Dropout layer with** z **rate** of input units to drop.
3. **2D convolutional layer with** x **channels** with 3x3 filter dimensions, stride of 1 and "same"padding so that if you use a stride of 1, the layer's outputs will have the same spatial dimensions as its inputs.
4. **2D Maximum Pooling layer** with a 2x2 pooling dimension.

This can be seen as Python Tensorflow Keras code in Figure 4.1

```
c1 = Conv2D(16, (3, 3), activation=activation_func, kernel_initializer=init_method, padding='same')(inputs)
c1 = Dropout(0.1, )(c1)
c1 = Conv2D(16, (3, 3), activation=activation_func, kernel_initializer=init_method, padding='same')(c1)
p1 = MaxPooling2D((2, 2))(c1)
```

Figure 4.1: Example of convolution block($x = 16$, $z = 0.1$)

In this project a **transpose convolution block**(x,z) consists of:

1. **2D transpose convolutional layer with** x **channels** with 2x2 filter dimensions, stride of 2x2 and "same"padding.
2. **Skip connection** concatenate weights of 2D transpose convolutional layer with x channels and the weights of the previous 2D convolutional layer with the same number of x channels.
3. **2D convolutional layer with** x **channels** with 3x3 filter dimensions, stride of 1 and "same"padding so that if you use a stride of 1, the layer's outputs will have the same spatial dimensions as its inputs.
4. **Dropout layer with** z **rate** of input units to drop.
5. **2D convolutional layer with** x **channels** with 3x3 filter dimensions, stride of 1 and "same"padding so that if you use a stride of 1, the layer's outputs will have the same spatial dimensions as its inputs.

This can be seen as Python Tensorflow Keras code in Figure 4.2

```
u9 = Conv2DTranspose(16, (2, 2), strides=(2, 2), padding='same')(c8)
u9 = concatenate([u9, c1], axis=3)
c9 = Conv2D(16, (3, 3), activation=activation_func, kernel_initializer=init_method, padding='same')(u9)
c9 = Dropout(0.1, )(c9)
c9 = Conv2D(16, (3, 3), activation=activation_func, kernel_initializer=init_method, padding='same')(c9)
```

Figure 4.2: Example of transpose convolution block($x = 16$, $z = 0.1$)

With the above concepts and the U-Net architecture diagram introduced in Figure 2.16 in mind, the initial baseline U-Net architecture used in this project is composed of the following blocks in sequential order:

1. convolution block(16, 0.1), convolution block(32, 0.1), convolution block(64, 0.2), convolution block(128, 0.2)
2. convolution block(256, 0.3)

3. transpose convolution block(128,0.2), transpose convolution block(64,0.2), transpose convolution block(32,0.1), transpose convolution block(16, 0.1)

4. finally an output layer of a 1x1 2D convolutional layer with a sigmoid activation function to create the per-pixel class probability map

This is depicted in a lower level of detail by Figure 4.3, where yellow shade represents a Convolutional layer,darker yellow shade its activation function, red shade represents a Pooling layer, blue shade represents a Transpose Convolutional layer and arrows represent the skip connection.

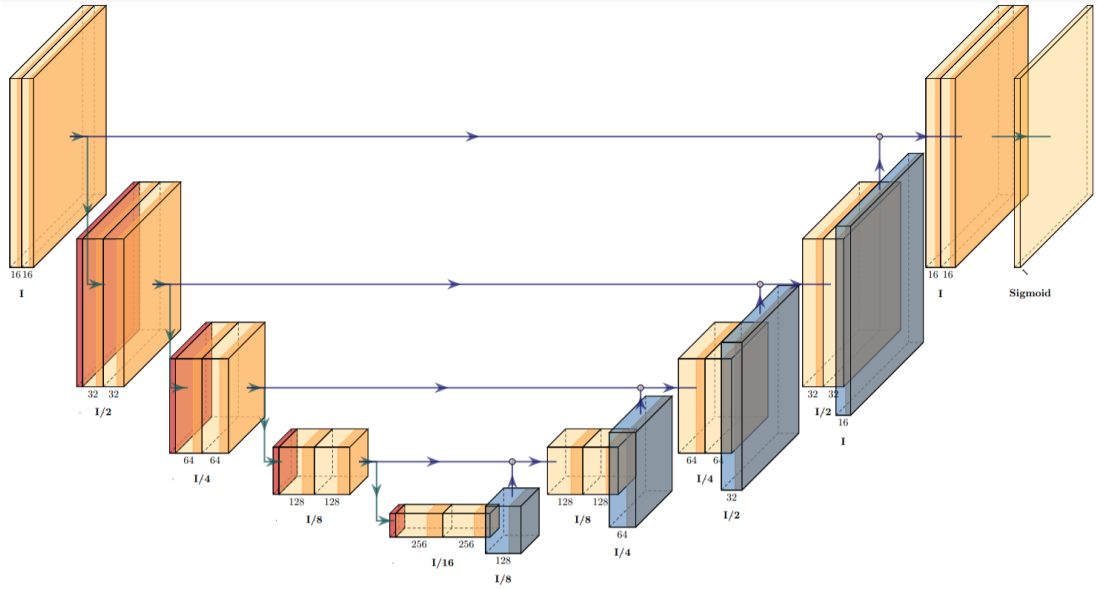


Figure 4.3: U-Net architecture used in this project

4.1.2 Evaluating the use of transfer learning

Given the small labelled dataset and the difference compared to ImageNet, since we are dealing with aerial shots of landscapes, the rule of thumb would be to fine-tune the hidden layers.

However, given that most models are only trained on a depth of 3 channels, the weights of such pre-trained models are not compatible with the choice of 4 channels or more. Therefore, this author deemed it better to re-train the model even though there is a small labelled dataset present.

4.1.3 Pixel-level class imbalance

Given the process of data collection adopted initially in this project where the area of interest i.e. the RTS positive pixels are in the centre of the image and an area of 256x256 is "cut"around it to create a patch.

The large area of the patch created a pixel-wise class imbalance which led to the model's poor performance when identifying positive RTS sample. This is due to the low level of RTS pixels where 75% of images have less than 360 pixels this when compared with the area size of a 256x256 image is less than 0.55% of the image, a needle in a haystack as some would say. By for example reducing the image area by 16 times to 4096, the area of a 64x64 image that is now more almost 9% of the image.

It also made accuracy a very misleading metric to follow, since the model would be rewarded for the hundreds of negative sample pixels it correctly identified, but wouldn't absorb the cost of misclassifying the pixels this project is most interested in finding the RTS pixels.

For example, say that 1% of the image is made up of positive RTS pixels while the rest is made up of background negative pixels, if the model predicts negative background pixel for the whole image it would still be 99% accurate, even though it didn't classify a single RTS pixel correctly, this can be very misleading. This unsuitability of accuracy as a performance measure, is also part of the justification for using the dice score introduced in section 3.2.4 as the performance measure of choice in the experiments that follow. Being the harmonic mean between precision and recall, it makes it one of the best candidates for unbalanced class problems.

Following this realisation, the size of the input patch was reduced to create a more even pixel-wise class distribution, the results of this experiment will be described in detail in section 4.2.1.

4.1.4 Experiment set up

In order to identify a baseline model, trial runs for only 20 epochs were performed in a grid search methodology to identify a suitable baseline model to use in the following experiment sections. After evaluating the dice coefficient, the parameters of the baseline model are:

Parameter	**Value**
Patch Size	64x64
Normalisation	naïve
Augmentation Method	None
Activation Function	Exponential Linear Unit (ELU)
Loss Function	CE Dice Loss
Optimiser	Adam
Batch size	4
Initial Learning Rate	0.0001
Learning Rate Schedule	ReduceLROnPlateau
Initialisation method	"He"Normal
Regulariser	Dropout with varying rates

For each of the experiments, there will be trial runs of each potential value for 50 epochs to identify the best value, keeping everything else as the baseline value. The objective is to maximise the performance measure of choice, the dice coefficient (coef) by minimising the loss.

4.2 Preprocessing experiments

The experiments in this section ran for 50 epochs, due to the non-deterministic nature of neural networks the results are not always of comparable performance between experiments, but were run within the same Tensorflow session within each experiment for comparability.

4.2.1 Size of the input patch

As described in section 4.1.3, class imbalance became an issue with model performance, in order to test if cropping the image to create a more balanced pixel-level class distribution could be linked to model performance, experiments were carried out. This was done by cropping the input patch from 256×256 to 128×128, 64×64 and finally 32×32 pixels, which would be reflected in the input layer dimensions of the model as well.

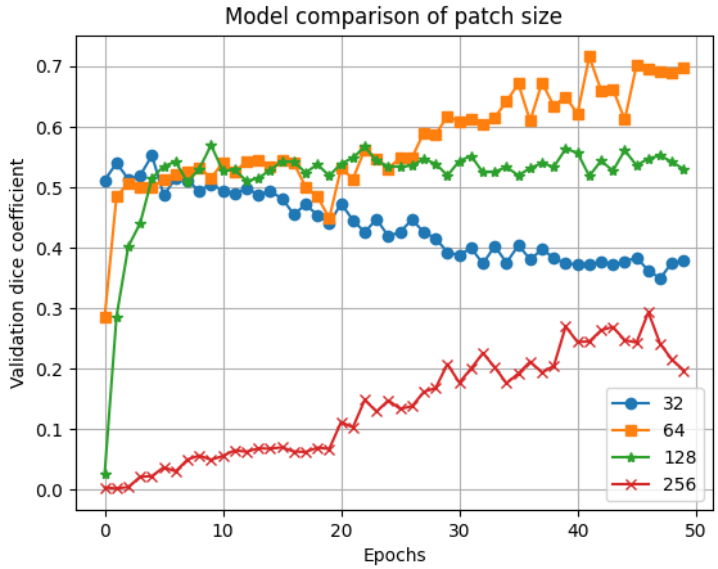


Figure 4.4: Input patch size Dice Coefficient comparison

The model with the highest validation dice coefficient was the one where the image was cropped to a 64×64 pixel area, with a validation dice coefficient of 0.7 vastly outperforming the other patch sizes, shown in figure 4.4.

	Dice Coefficient			Loss		
Patch size	Validation	Training	Test	Validation	Training	Test
32	0.38	0.59	0.34	1.2	0.82	1.27
64	0.7	0.82	0.78	0.5	0.26	0.3
128	0.53	0.6	0.62	0.57	0.48	0.43
256	0.2	0.31	0.14	0.84	0.72	0.9

Table 4.1: Input patch size comparison of Dice Coefficient and Loss

The results shown in table 4.1, indicate increasing performance with the decrease of the patch size, up until the 64×64 patch size, which would indicate that at 32×32 patch size the context surrounding the larger thaw slumps (which just fit in the 64×64 patch) is lost leading to decreasing performance, therefore the remaining experiments of this project will use a 64×64 patch size and input layer dimensions.

4.2.2 Normalisation of input

Another key factor in the success of the training of the model is adequate normalisation of the inputs, specially with the extremely high values reflectance values in remote sensing images can take, as described in section 3.2.1.2. Getting this wrong can lead to explosive gradients and invalid (NaN) loss and performance metrics when training the model, as witnessed in this project.

The naïve approach of simply converting any pixel $x > 10,000 = 10,000$ followed by $x = x/10,000$ was the one used for the initial experiments.

In order to implement the other two transformations $[0,1]$ interval and z-score, introduced in detail in section 3.2.1.2, the maximum, mean and standard deviation of all 64×64 training set images were calculated across each band/channel independently. This is done using only the training set images to prevent the distribution of the validation and test set leaking into the model.

Given the better performance of the naïve normalisation method seen in Figure 4.5, it will be used for the rest of the experiments here onward.

	Dice Coefficient			Loss		
Normalisation method	Validation	Training	Test	Validation	Training	Test
max	0.69	0.87	0.71	0.49	0.18	0.41
naive	0.75	0.89	0.8	0.39	0.16	0.27
z score	0.7	0.92	0.8	0.5	0.13	0.28

Table 4.2: Normalisation method comparison of Dice Coefficient and Loss

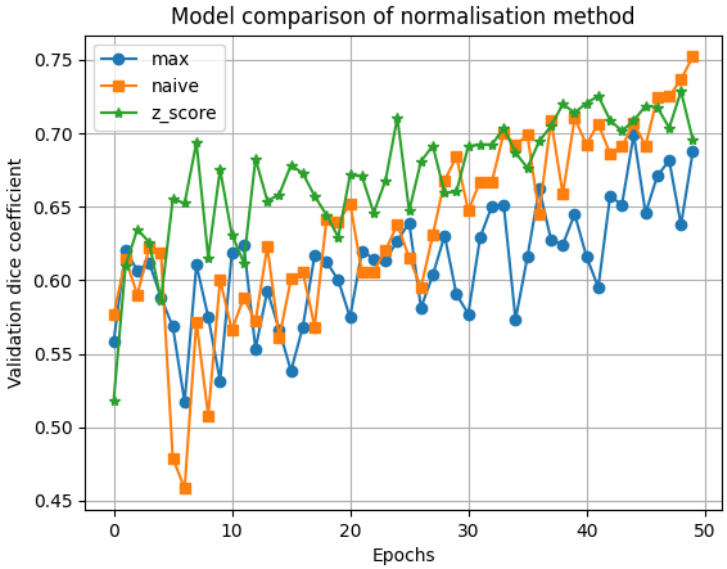


Figure 4.5: Normalisation method Dice Coefficient comparison

4.2.3 Data Augmentation

Data Augmentation can be invaluable when the dataset is small, and even more when the data collection is biased. The main purpose of data augmentation in this project is to correct the bias of RTS positive pixels always being in the centre of the image.

In that sense, the most useful data augmentation transformation is translation or shift, where the image is translated to the right or left according to certain parameter. The implementation of this was done in the NN itself has one of the initial layers with Keras's "RandomTranslation" layer. One layer with an output shifted vertically and horizontal by a random amount of 20% in each direction, to fill the pixels left blank by this translation the fill mode of choice was bipolar interpolation of the nearest pixels.

At this stage a rotation augmentation method was also implemented for comparison, in the same way with the same parameter settings but using Keras's "RandomRotation" layer instead, which rotates the image clockwise or anti-clockwise instead by the random amount of 20%, with the same fill method.

The experiment results can be seen in Figures 4.6, no augmentation (baseline) yields better results than rotation, translation, or both implementations at once.

This could be due to changes in the geometry or lighting causing the ROI or important contextual background pixels in the image to lose the original features that helped identify the RTS pixels correctly.

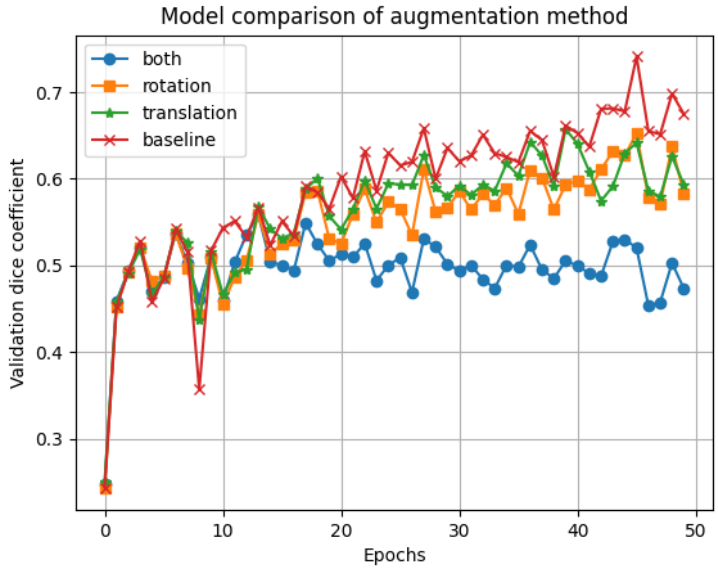


Figure 4.6: Augmentation method Dice Coefficient comparison

At this point in the experiment process, the experiments will proceed without any augmentation, other options will be explored to address the data collection bias element.

4.3 Training experiments

4.3.1 Network Hyperparameters

Given the complexity of an optimiser's performance and its interaction with the learning rate and batch size parameters, a comprehensive grid search experiment was performed to avoid a misleading evaluation of the parameters to follow.

For simplicity, the same approach of only varying one parameter at a time, to ease interpretation, will be presented in the following subsections.

4.3.2 Optimiser

As introduced in section 3.2.3.2, there are several optimisers that could significantly affect the performance of the model. Even though SGD seems to be the algorithm used by all the state-of-the-art segmentation models presented in section 2.4, the results of varying four different optimisers will be presented to see how the model is impacted by it.

Contradicting the literature, perhaps given the small dataset in this project compared with the millions of examples used to train state-of-the-art models, RMSProp outperforms the other optimisers when it comes to loss minimisation as seen in Figure 4.7.

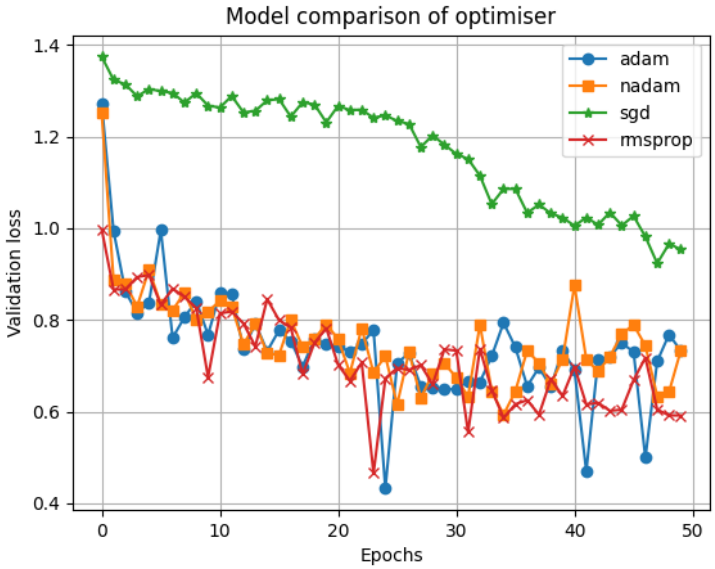


Figure 4.7: Optimiser Loss comparison

	Dice Coefficient			Loss		
Optimiser	Validation	Training	Test	Validation	Training	Test
adam	0.75	0.89	0.72	0.55	0.16	0.4
nadam	0.77	0.9	0.72	0.55	0.15	0.38
sgd	0.56	0.62	0.62	0.78	0.57	0.51
rmsprop	0.79	0.9	0.76	0.53	0.15	0.34

Table 4.3: Optimiser comparison of Dice Coefficient and Loss

Figure 4.7 shows similar validation loss coefficient between RMSProp, NAdam and Adam, in order of minimisation performance, with SGD lagging behind significantly, given the baseline learning rate of 0.0001 and mini-batch size of 4.

This may be due to the fact that the default parameters (apart from learning rate) of the Tensorflow implementation of all the optimisers are being used and need tuning for better performance and perhaps that Adam is better used in problems with a lot of data and/or parameters. (Kingma and Ba, 2017)

Given its performance, RMSProp will be the optimiser of choice in the experiments to follow.

4.3.3 Learning rate

The Keras method "ReduceLROnPlateau"was applied, this involved reducing the learning rate by a factor of 10, if there is no improvement in the validation set loss for 10 epochs. This is enabled by the callback method that monitor loss and other metrics

for each epoch.

The initial learning rate was chosen from 0.01, 0.001, 0.0001, based on the literature introduced in section 3.2.3.6 for this experiment, the results can be seen in Table 4.4.

	Dice Coefficient			Loss		
Learning rate	Validation	Training	Test	Validation	Training	Test
0.01	0.35	0.35	0.33	1.03	0.91	0.9
0.001	0.75	0.92	0.8	0.86	0.11	0.28
0.0001	0.79	0.9	0.76	0.53	0.15	0.34

Table 4.4: Learning rate comparison of Dice Coefficient and Loss

Figure 4.8: Learning rate Dice Coefficient comparison

Figure 4.8 shows a big difference between the performance of 0.01 learning rate and the other values, this could be due to the large learning rate overshooting the global or a local minimum and arriving at a suboptimal final set of weights. In the grid search experiment, the initial learning rate of 0.01 seemed to only perform well with the SGD optimiser and vice versa, the SGD optimiser only performed well with the 0.01 learning rate. This could be because the initital large learning rate allows SGD to do exploration of the search space early on and later on reducing through the "ReduceLROnPlateau"learning rate schedule allowing for the explotation of the search space in later epochs to get closer to the optimal loss.

The top learning rate for the RMSProp optimiser was 0.0001, therefore this will be used for the experiments.

4.3.4 Batch size

Varying the mini-batch size has great implications in the convergence of the model, given its small labelled dataset, this project is working with smaller batch sizes than it is standard in the industry.

This is validated by the results of the experience, as it can be seen in Table 4.5 the batch sizes that show the worst performance are the 2 highest values of 16 and 20 mini-batch size. All the other values display a very close performance, with the best being a batch size of 1.

	Dice Coefficient			Loss		
Batch size	Validation	Training	Test	Validation	Training	Test
1	0.85	0.93	0.8	0.47	0.1	0.29
2	0.79	0.88	0.77	0.46	0.17	0.31
4	0.68	0.84	0.71	0.6	0.23	0.39
6	0.67	0.83	0.73	0.61	0.25	0.37
10	0.58	0.65	0.64	0.65	0.51	0.49
16	0.55	0.7	0.5	0.63	0.45	0.7
20	0.49	0.65	0.66	0.82	0.51	0.48

Table 4.5: Batch size comparison of Dice Coefficient and Loss

4.3.5 Initialisation

As presented in section 3.2.3.4, "He"initialisation methods outperform the "Xavier"(glorot) initialisation method, according to the literature "He"initialisation works better mathematically by adressing the nonlinearities of the ReLU activation function, which are also present in the ELU activation function.

The random normal initialiser has the same final performance as the "He"normal method in Table 4.6, this results may have been due to a particularly good random weight initialisation, in order to assess it's validity it should be run several times to establish it's significance, however due to time and computational constraints this author did not investigate this further.

In a complete opposite way, using the random uniform initialisation function leads to a low stagnated validation dice coefficient, as can be seen from Figure 4.9. This may be due to a bad initialisation, which lead to the backpropagation algorithm being unable to identify the right direction to transform the weights to optimise the cost function, which consequently stalls training.

However, by looking at Figure 4.9 "He"Normal seems to be more stable and achieve a better performance at an earlier step than the Random Normal initialiser, therefore going forward the "He"Normal initialisation method will be used.

	Dice Coefficient			Loss		
Initialisation method	Validation	Training	Test	Validation	Training	Test
he normal	0.82	0.93	0.76	0.57	0.1	0.35
he uniform	0.8	0.92	0.77	0.65	0.11	0.34
glorot normal	0.79	0.87	0.71	0.47	0.19	0.4
glorot uniform	0.77	0.85	0.75	0.51	0.24	0.34
random normal	0.82	0.91	0.76	0.59	0.14	0.34
random uniform	0.69	0.72	0.67	0.64	0.45	0.45

Table 4.6: Initialisation method comparison of Dice Coefficient and Loss

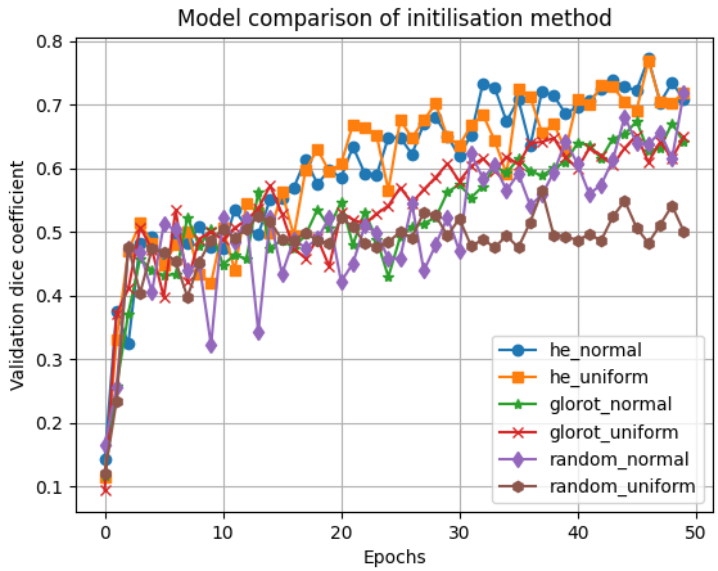


Figure 4.9: Initialisation method Dice Coefficient comparison

From here on, "He"Normal initialisation will be used for the consequent experiments.

4.3.6 Activation function

The discussion of the advantages and disadvantages of different the different activation functions has been discussed in detail in section 2.3.2.

From Figure 4.11 ReLU stands out since the validation loss completely stalls which indicates that the entire network has died, leading to unsuccessful convergence despite increasing the number of epochs. Dying ReLU is a known phenomena with many explanations on how neurons become inactive always outputting 0 for any given input. Lu et al. argue that symmetric probability distributions such as "He"initialisation for initialisation is a big cause of dying ReLU and propose their own randomised asymmetric initialisation (Lu, 2020).

On the contrary, ELU performs very well, having the highest Validation Dice coefficient, as can be seen from Table 4.7. Since the constant value of 0 for any negative input range is what causes dying ReLU,using one of its variations in this experiment avoids the problem associated with the zero-slope segment.

The Scaled Exponential Linear Unit (SELU) function was also considered but as it is self normalising the NN (Klambauer et al., 2017) it deemed necessary yet another implementation, LeCun Normal initialisation and a special version of Dropout called Alpha Dropout, which this author did not perform at this point.

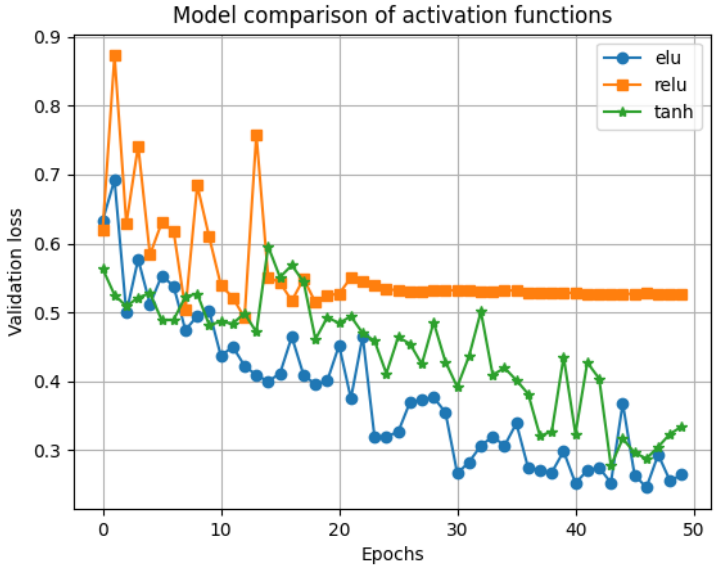


Figure 4.10: Activation function Loss comparison

	Dice Coefficient			Loss		
Activation Function	Validation	Training	Test	Validation	Training	Test
ELU	0.85	0.93	0.78	0.26	0.11	0.31
ReLU	0.67	0.73	0.64	0.53	0.45	0.52
tanh	0.8	0.88	0.77	0.33	0.19	0.32

Table 4.7: Activation function comparison of Dice Coefficient and Loss

ELU proves to be the highest performing activation function, so it will remain being used in the following experiments.

4.3.7 Loss Function

The chosen measure of performance for this project is the dice coefficient, an adequate loss function needs to be implemented in a way that minimising it, maximises the dice coefficient. To achieve this, the loss functions introduced in section 3.2.3.1

were implemented in Python and compared to identify the best suited function for this project.

The implementation of the loss functions in Python under the Tensorflow Keras framework required a substantial amount of effort, as they were custom losses and required special attention to make sure matrix dimensions were correct for backpropagation to work and training to be successful.

Figure 4.11 shows that they are all successfully training the model as the chosen model scoring metric is showing an increasing trend as the number of epochs increase as would be expected if the gradient descent is working correctly.

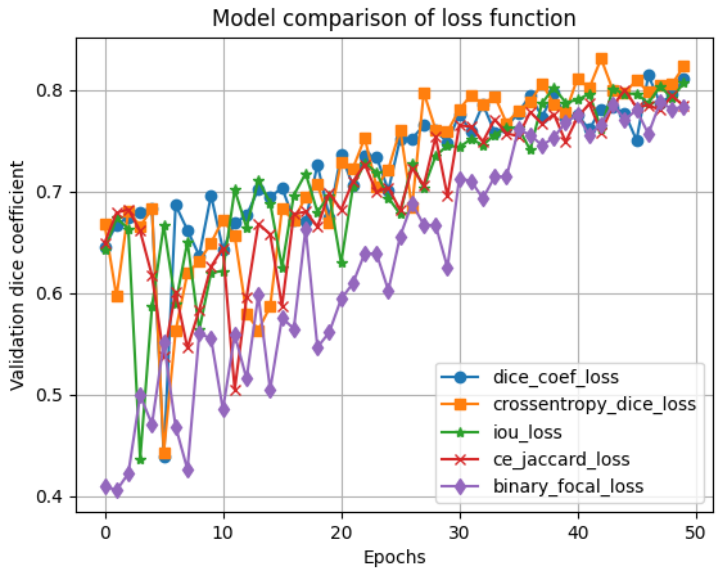


Figure 4.11: Loss function Dice Coefficient comparison

	Dice Coefficient			Loss		
Loss Function	Validation	Training	Test	Validation	Training	Test
dice coef loss	0.81	0.9	0.75	0.19	0.1	0.25
CE dice loss	0.82	0.93	0.81	0.57	0.1	0.28
iou loss	0.81	0.91	0.71	0.29	0.15	0.45
CE jaccard loss	0.78	0.93	0.8	1.06	0.18	0.51
binary focal loss	0.78	0.86	0.77	2.2	0.25	0.45

Table 4.8: Loss function comparison of Dice Coefficient and Loss

The best loss function is still the CE dice loss, this could be because combining the two losses allows for some diversity, whilst still benefitting from the stability of CE, so it will continue to be used in the following experiments.

4.3.8 Preventing over-fitting

4.3.8.1 ES

In the previous section the models ran for 50 epochs without ES implementation, given the experiments performed what is the impact of running a couple of the experiments applying ES and how it affects the performance of the model.

The early stopping strategy consisted of monitoring the validation set loss for 15 epochs (patience parameter) after which training will be stopped if there is no improvement (absolute change of less than 0 (min delta parameter)) in the validation loss.

In order to perform this experiment an arbitrarily large number of epochs was set to 200, ES was triggered at 79 epochs when validation loss did not improve from 0.24, as can be seen from Figure 4.12 ES was successful at preventing an overfitting situation, as the loss increased after epoch 79.

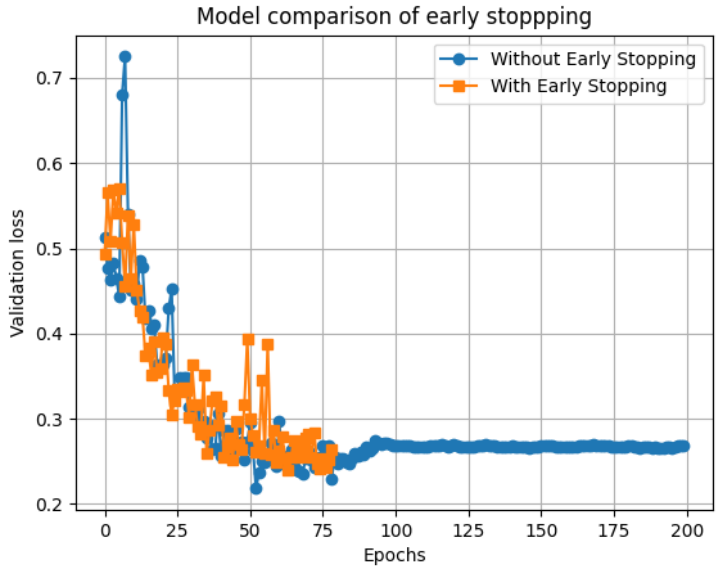


Figure 4.12: ES usage Loss comparison

4.3.8.2 Dropout

As part of the base network architecture introduced in section 4.1.1, Dropout has been implemented at different degrees starting at 0.1 dropout rate and increasing by 0.1 as one progresses deeper into the network.

As an experiment, the network with the same architecture with one change, all the dropout values introduced will be reduced to 0, so there is no Dropout in the network.

Without dropout in place the validation loss starts increasing drastically which triggers ES much earlier, as we can see from Figure 4.13 this could be due to exploding

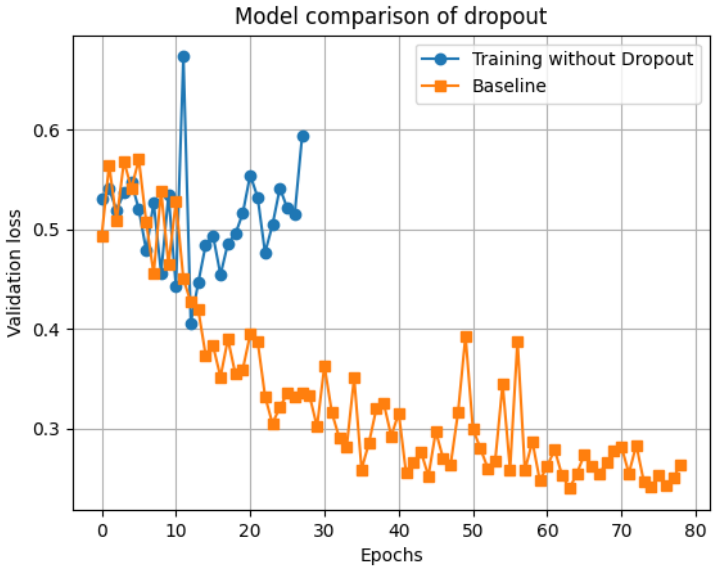


Figure 4.13: Removal of Dropout Loss comparison

gradients.

4.4 Learnings from conducting experiments

In summary, there are a few things that have been concluded from the experiments conducted with the *GEE data* and will motivate the decisions made for the section that follows where the model will be trained on the *JP2 data* .

1. When there is a background and foreground class, where the foreground class represents the ROI, in this case the RTS pixels, it is important to adjust the patch size of the image as a way of reducing pixel imbalance between the negative more frequent background class and the positive more valuable foreground class. Depending on the size of the ROI this has to be done taking into consideration that context around the ROI is also valuable in its correct identification. For example in the context of this thesis reducing the patch size showed improvement until the 64x64 patch size, but once it was further reduced to 32x32 the performance was worst.

2. The relationship between the learning rate hyperparameter and the optimiser is very strong, this was particularly evident with a higher learning rate of 0.1, which incredibly benefitted the SGD optimiser but drastically reduced the performance of the other optimisers, perhaps more experiments with the other parameters of optimisers and different learning rate schedules would be interesting, given more time.

3. The relationship between mini-batch size and learning rate also seems to be very important, despite changing optimiser, it was observed through the grid search experiment that batch size of 1 works better with 0.0001 learning rate, whereas batch size of 10 performs better with 0.001 learning rate, this may be due to the fact that by seeing more data with a bigger batch size the model is less likely to overshoot a minimum and hence can tolerate a higher initial learning rate and vice-versa.

4. The correct technical implementation of loss functions is challenging, specially when dealing with bespoke data ingested from GeoTiFF file format but essential for the successful training of the model.

5. ES is very effective in preventing overfitting.

6. Dropout is very effective in preventing exploding gradients.

5

Training the model on more data

In section 3.1.3 JP2 data is introduced as a larger dataset sourced to address the issues of bias in data collection. This chapter is dedicated to the evaluation of the impact of using such data on the model fine-tuned on the smaller GEE dataset through the experiments in Chapter 4.

5.1 Using the weights of previous model to train on new data

The model was frozen using the best hyperparameters at the end of the Chapter 4 and the model was trained on new data using those weights, this will be compared with the performance of training the model from scratch on new data in section 5, the same ES strategy will be used for this experiment to prevent overfitting.

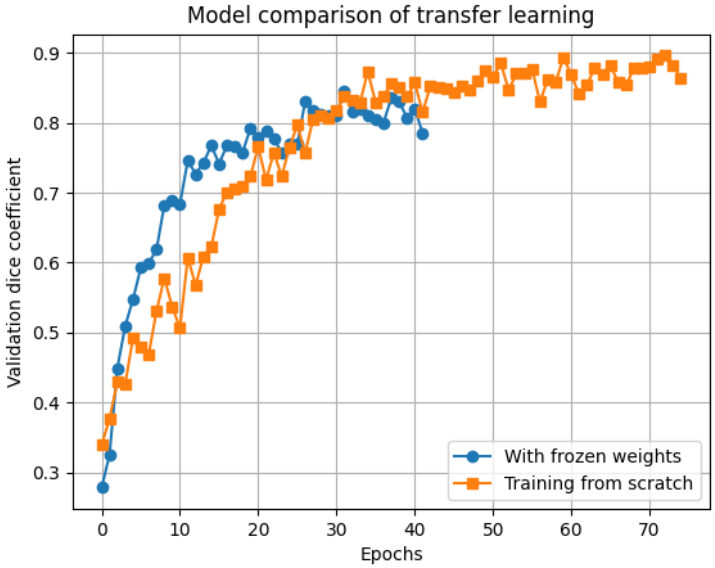


Figure 5.1: Retraining with frozen weights Dice Coefficient comparison

Using the frozen weights did not improve model performance, it had the opposite

effect which is expected since the original model has a smaller dataset than the one it was trained on now. It can be seen from Figure 5.1 that up to 20 epochs the transfer learning by using frozen weights from the model trained on less data is performing better than the model training from scratch.

Early stopping is also triggered in the model using frozen weights around 30 epochs before the model being trained from scratch, which makes sense since when there is less data the model has a greater likelihood of overfitting.

5.2 Training the model from scratch on new data

5.2.1 Comparison with the GEE dataset model - same parameters

With the same model architecture and hyperparameters as in the previous Chapter, the DL model was trained on the *JP2 data* for the same number of epochs to evaluate the effect of using more data on the model performance.

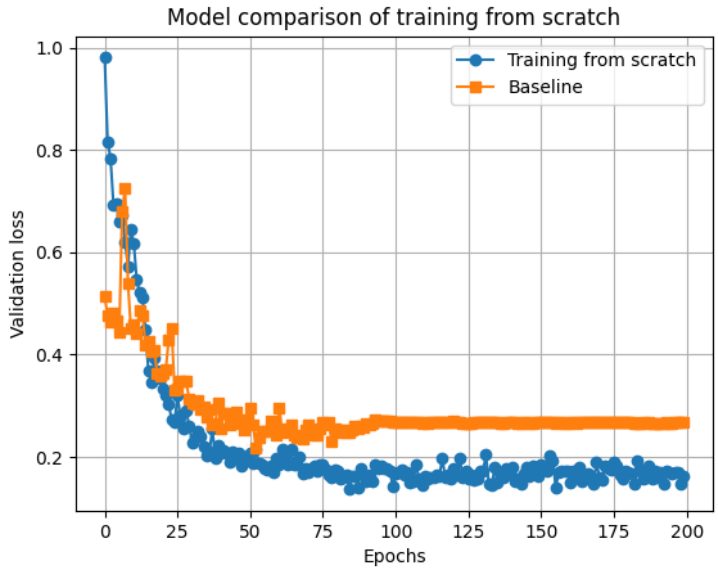


Figure 5.2: Retraining from scratch Loss comparison

As can be seen from Figure 5.2 the model trained on more data has a lower validation loss consistently after 20 epochs despite initialising with a higher validation loss, which shows it's better performance.

In terms of evaluation metrics, it can be seen from Table 5.1 that the improvement in performance is the most evident in the Test dataset where there is an improvement of 0.08 in the dice coefficient metric when using Early stopping (ES) to prevent overfitting.

Model type	Dice Coefficient			Loss		
	Validation	Training	Test	Validation	Training	Test
Training from scratch	0.89	0.96	0.88	0.16	0.05	0.28
Training from scratch with ES	0.86	0.94	0.86	0.2	0.08	0.3
Baseline model	0.88	0.97	0.82	0.27	0.05	0.27
Baseline Model with ES	0.88	0.96	0.78	0.26	0.07	0.33

Table 5.1: Retraining on new data comparison of Dice coefficient and Loss

5.2.2 Hyperparameter tuning from scratch

What works well for a dataset in terms of hyperparameters may not work well for another, especially when there is a considerable difference in sample size. With this in mind, hyperparameter tuning using random search was performed on the new data with the following parameter variations using the same Early Stopping strategy as before, restoring the weights of the best performing epoch w.r.t. Validation Loss:

- Learning Rate: 0.0001 and 0.001
- Optimiser: RMSprop,Adam and Nadam
- Loss Function: CE Dice Loss
- Batch Size: 1 and 10
- Activation Function: ELU
- Initialisation Method: He Normal

Results were logged using Tensorboard, the best validation dice coefficient combinations of hyperparameters are shown in Figure 5.3, the best hyperparameters for the JP2 data were batch size of 10, RMSProp optimiser and learning rate of 0.001.

As it can be seen from Table 5.2 the difference in performance with regards to the Validation Dice Coefficient is marginal between the different hyperparameter combinations, showing that the previous experiments in Chapter 4 provided good insights that generally apply to a different bigger dataset.

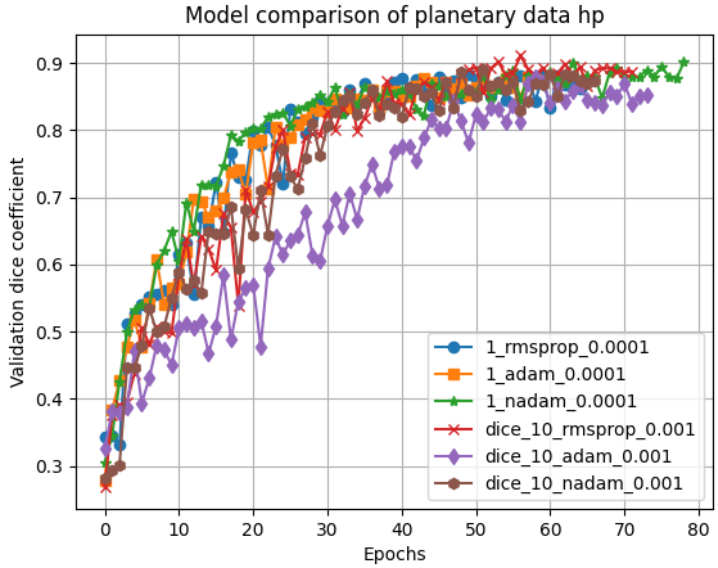


Figure 5.3: Subset of hyperparameter experiments with new data Validation dice coefficient

	Dice Coefficient			Loss		
Hyperparameter combination	Validation	Training	Test	Validation	Training	Test
1 rmsprop 0.0001	0.86	0.94	0.93	0.2	0.07	0.15
1 adam 0.0001	0.87	0.93	0.91	0.19	0.09	0.14
1 nadam 0.0001	0.86	0.93	0.76	0.2	0.09	0.22
10 rmsprop 0.001	0.91	0.96	0.95	0.13	0.05	0.11
10 adam 0.001	0.6	0.68	0.64	0.55	0.44	0.36
10 nadam 0.001	0.59	0.72	0.84	0.56	0.38	0.33

Table 5.2: Hyperparameter combinations comparison of Dice Coefficient and Loss

6

Final model and evaluation

6.1 Final Model

The best model found was that from the hyperparameter tuning performed on the new *JP2* data in Section 5.2.2, the performance of this model was summarised in Table 5.2, more in depth analysis per epoch will now be performed in this section.

As it can be seen from Figure 6.2 there is no evidence of overfitting the Early Stopping strategy was kicked off at the 88th epoch, it can also be seen from Figure 6.6 that the Training and Validation curve follow a similar shape with less fluctuation towards the final epochs, as would be expected from a well performing model.

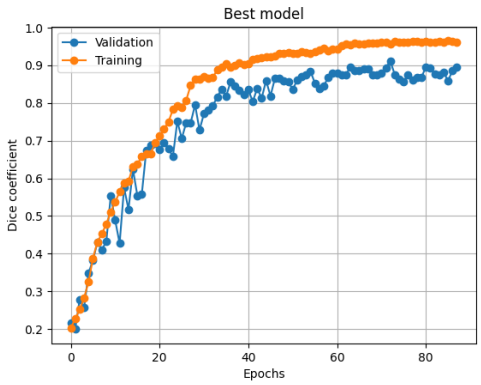


Figure 6.1: Dice Score comparison

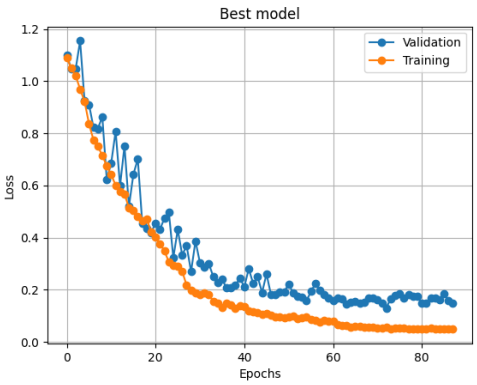


Figure 6.2: Loss comparison

6.2 Test Evaluation

During the Training/ Validation/ Test split process 39 images were assigned to the test set, that is completely unseen data by the model. Using the best model's frozen weights (the best weights are restored as part of the Early Stopping strategy) and architecture,

inference is performed of the test dataset to assess model performance by comparing the predictions against the ground truth mask.

The overall performance of the model at image level is summarised in Table 6.1, where 3 categories are defined according to model performance, the individual performance evaluation scores of each test image are then compared in Figure 6.3 according to those categories.

Category	Dice Score range	Number of images
High	Greater than or equal to 0.7	29
Medium	Between 0.7 and 0.4	5
Low	Smaller or equal to 0.4	5

Table 6.1: Dice score test image summary

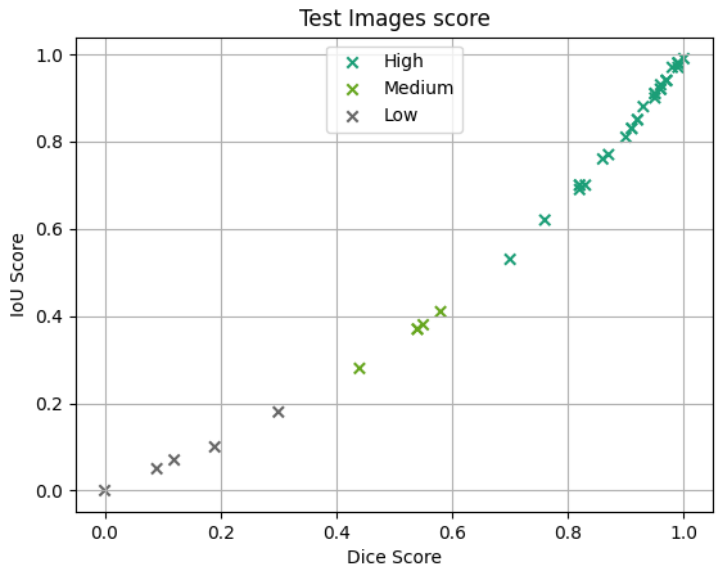


Figure 6.3: Dice Score vs. IoU Score Test images scatter plot

A short image by image analysis will be performed for 2 images of each category, where purple colour represents background (non-RTS) pixels and yellow represents foreground (RTS) pixels, with varying opacacy according to prediction confidence, the lighter the shade the lower the confidence, there is also a red contour which represends the ground truth mask outline.

As can be seen from Figure 6.4 the algorithm can deal with both simpler thaw slump shapes (on the left) and more complex shapes (on the right), although the later shows

a lower confidence in the pixel prediction confidence, as can be seen by lower intensity pixel shading and is also reflected in a lower dice score.

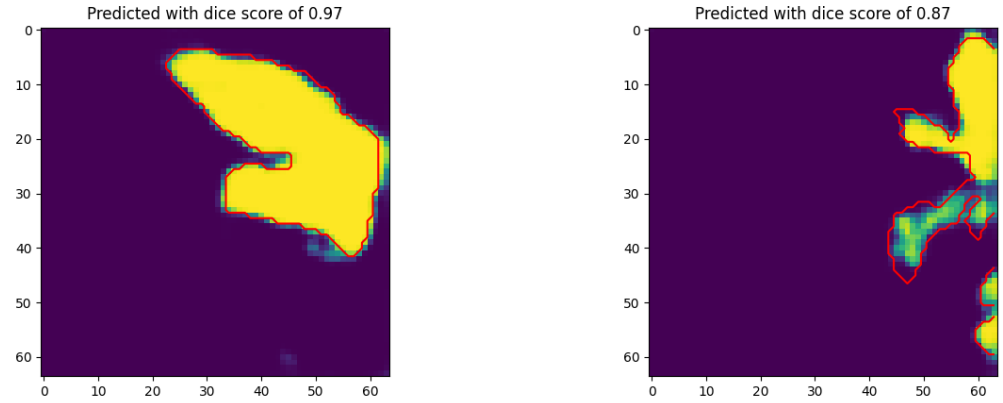


Figure 6.4: Test set predicted vs. ground truth high score examples

The model struggles with elongated thaw slumps as seen in Figure 6.5 on the left and in some instances correctly predicts the thaw slump ROI but also predicts False Positive RTS which then has an impact on the dice score (right side). This presence of False Positives could be an indication of an actual RTS, which hasn't been labeled yet. Due to the lack of expert knowledge, these False Positives haven't been validated, but could point researchers in the right direction when looking for new RTS.

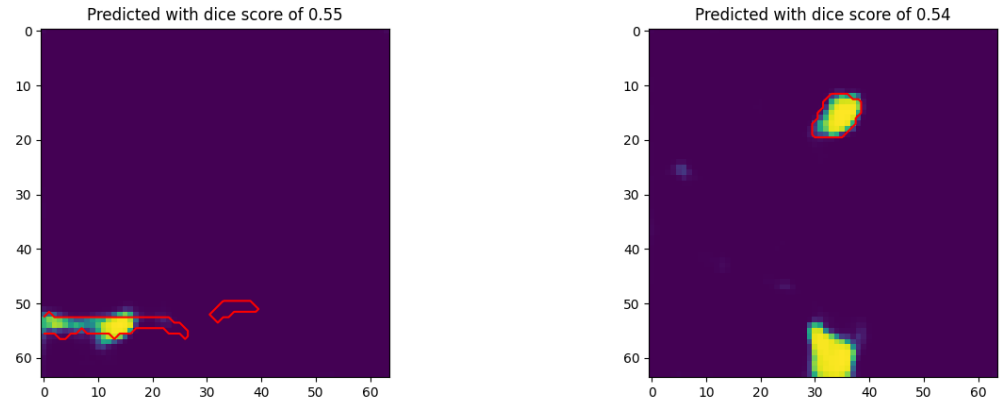


Figure 6.5: Test set predicted vs. ground truth medium score examples

The really low performing test images can be seen in Figure 6.6, the model seems to not be very good at predicting very small RTS, this could be due to the fact that each pixel represents a $10\,m^2$ area and some RTS may be smaller than that, or that the image reprojection has a slight misalignment, as can be seen from the image on the left which predicts a False Positive RTS to the left of the ground trutch red contour.

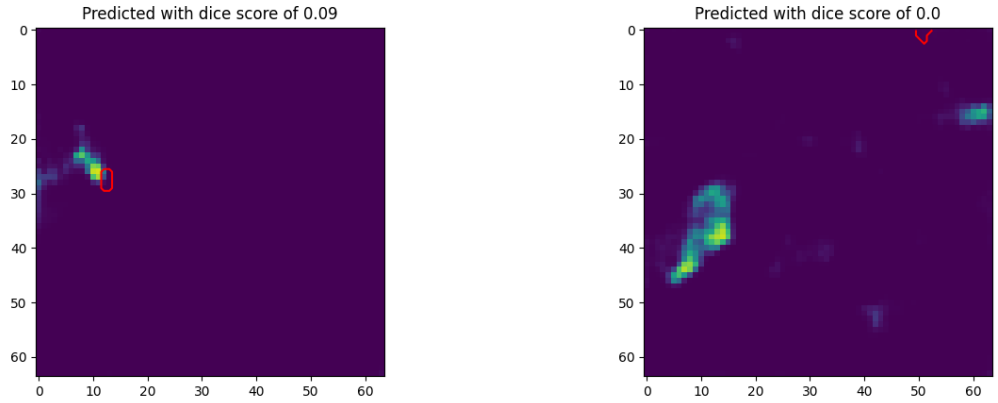


Figure 6.6: Test set predicted vs. ground truth low score examples

Another level of analysis, involves looking at the input images (X) next to the gorund truth and predictions (y and predicted y), to check if there is any visible evidence of the assumptions made above. An example of this can be seen in Figure 6.7, by analysing the input image (on the left) it can be seen that the image is corrupted at the top, this could be perhaps due to an error in the multispectral instrument during the collection, or perhaps during further post-porcessing. When it comes to the RTS ROI, the lighter shading in the input image corresponds to the predicted area in the right hand side, indicating that perhaps indeed the reprojection of the labelled data (red contour) to the input image's CRS is perhaps incorrect and the algorithm is indeed more accurate than the dice score of 0.58 seems to suggest.

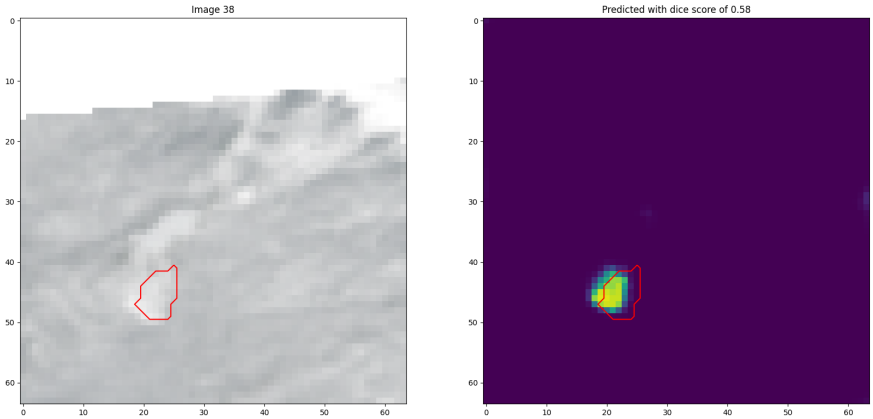


Figure 6.7: Test set predicted vs. ground medium score example

Given more time, all the above assumptions would be investigated further.

7

Conclusions and Future Work

7.1 Conclusions

Two models trained on different datasets were fine-tuned and evaluated in this project, during this process there were several conclusions that can be made:

1. Despite the challenges of data extraction and preprocessing, once a script is in place that does the extraction and pre-processing in a semi-automatic way it is relatively easy to iterate on that and improve. The challenge comes in the time required to understand the open-source APIs one can access the data through and the format of that data, for example GeoTiff files are not as common as JPEG or PNG files so there is a certain level of upskilling and understanding of this different files.

2. Changing input parameters has a great impact in the performance of the model it was seen that the patch size was a very important hyperparameter in the improvement of the model performance, this was partly to overcome the imbalance between background (non-RTS) pixels and foreground (RTS) pixels.

3. Satisfactory results were achieved with 10-meter resolution images for the identification of thaw slumps, however for a more valuable task of change monitoring in said RTS, the model could benefit from higher resolution images as the change maybe less than 10m and therefore be hard to track. It could also help with the performance on smaller RTS or those with more intrecate shapes as it would provide a more detailed view.

4. All the points above put a great emphasis on the input data, from this project the importance of having the right data, collected and pre-processed in the right way is essential for the successful performance of the model, in hindsight this author would have spent more time getting this step correct to avoid having to backtrack to get more data half way through the project.

5. Learning rate and batch size hyperparameters are some of the most important hyperparameters to tune in a model, this together with the correct regularisation strategy to prevent overfitting were some of the most important aspects found when conducting experiments.

7.2 Limitations

The data extraction part of the project came with many limitations. The dependency on third party tools to extract data led to a small dataset to work with, given the limited time constraints to focus on a more automated process for data extraction.

The GEE and JavaScript data extraction method introduced bias and potentially a misalignment between the input data and the mask. This was addressed by the second batch of data extraction, where JP2 tiles were extracted directly and processed into 64x64 windows, this allowed for debiased images and greater sample size to work with improving generalisation.

Since this model was only trained on tiles containing positive labels, it does not know how to identify tiles with no thaw slumps, which is an important characteristic when performing inference across the Arctic.

No model architecture hyperparameter tuning was made, that is the number of filters, kernel size, padding type, stride, pooling stride, type of pooling weren't optimised and other model architectures weren't evaluated, so potentially the solution presented may not be the most efficient or the network might be too deep for the problem at hand.

7.3 Future Work

In the future an approach that aims to measure the change in RTss through time, like in Huang et al.'s latest paper (L. Huang et al., 2021) would be more beneficial to the problem this project aims to address. This increases the complexity of the task at hand as it introduces an additional dimension of time and it's sequence to ensure adequate treatment of the time series but provides better means for estimation and prediction of permafrost thaw year on year.

It would also be beneficial to try different architecture parameters such as filter size and kernel size or even other architectures such as DeepLab v3+ or FCN and assess the impact on the model performance.

To improve the generalisation of the model, a wider research area not limited to the sites where labels were provided should be covered so that the model can be used in other geographical characteristics.

Given more time, there are considerations on model inference time, throughput and size are areas that could be improved by simplifying the U-Net model to a simpler

model architectures, this could be useful if the model were to be integrated in edge devices or applications that require low latency.

The use of Bayesian optimisation techniques or even Constraint Active Search (Malkomes et al., 2021) to perform hyperparameter tuning rather than using Random Search techniques would likely lead to better model optimisation without the time consuming experiment set up presented in Chapter 4

In the future, the model could be trained on higher resolution images such as those available through Planet data (3m) , to evaluate how it affects model performance specially when dealing with smaller RTS and those with more complex shapes.

To increase model performance on tiles with no or small number of thaw slumps the model should be trained on a balanced dataset of tiles containing thaw slumps, as well as, some not containing any.

Bibliography

Agency, T. E. S. (n.d.). The european space agency website.

Alvarez, J. L. H., Ravanbakhsh, M., & Demir, B. (2020). S2-cgan: Self-supervised adversarial representation learning for binary change detection in multispectral images.

Belenguer-Plomer, M. A., Tanase, M. A., Chuvieco, E., & Bovolo, F. (2021). Cnn-based burned area mapping using radar and optical data. *Remote Sensing of Environment*, *260*, 112468. https://doi.org/https://doi.org/10.1016/j.rse.2021.112468

Belgiu, M., & Drăguţ, L. (2016). Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, *114*, 24–31. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2016.01.011

Benedetti, P., Ienco, D., Gaetano, R., Ose, K., Pensa, R. G., & Dupuy, S. (2018). M^3Fusion: A deep learning architecture for multiscale multimodal multitemporal satellite data fusion. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *11*(12), 4939–4949. https://doi.org/10.1109/JSTARS.2018.2876357

Bramhe, V. S., Ghosh, S. K., & Garg, P. K. (2018). Extraction of built-up areas using convolutional neural networks and transfer learning from sentinel-2 satellite images. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *XLII-3*, 79–85. https://doi.org/10.5194/isprs-archives-XLII-3-79-2018

Calel R., S. D. e. a., Chapman S.C. (2020). Temperature variability implies greater economic damages from climate change. *Nat Commun*, *11*. https://doi.org/https://doi.org/10.1038/s41467-020-18797-8

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2016). Semantic image segmentation with deep convolutional nets and fully connected crfs.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2017). Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs.

Chen, L.-C., Zhu, Y., Papandreou, G., Schroff, F., & Adam, H. (2018). Encoder-decoder with atrous separable convolution for semantic image segmentation. *Proceedings of the European Conference on Computer Vision (ECCV)*.

Cheng, G., & Han, J. (2016). A survey on object detection in optical remote sensing images. *ISPRS Journal of Photogrammetry and Remote Sensing*, *117*, 11–28. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2016.03.014

chuur E. McGuire, C. e. a., A. Schädel. (2015). Climate change and the permafrost carbon feedback. *Nature*, *520*, 171–179. https://doi.org/https://doi.org/10.1038/nature14338

Cloud, G. (n.d.). Sentinel-2 data public dataset documentation.

Cohen J., F. J. e. a., Screen J. (2014). Recent arctic amplification and extreme mid-latitude weather. *Nature Geosci*, *520*, 627–637. https://doi.org/https://doi.org/10.1038/ngeo2234

Dozat, T. (2016). Incorporating nesterov momentum into adam.

Dumoulin, V., & Visin, F. (2018). A guide to convolution arithmetic for deep learning.

Everdingen, R. O. v., & International Permafrost Association (USA). (1998). *Multi-language glossary of permafrost and related ground-ice terms in chinese, english, french, german ...* Arctic Inst. of North America University of Calgary.

Fathi, E., & Maleki Shoja, B. (2018). Chapter 9 - deep neural networks for natural language processing. In V. N. Gudivada & C. Rao (Eds.), *Computational analysis and understanding of natural languages: Principles, methods and applications* (pp. 229–316). Elsevier. https://doi.org/https://doi.org/10.1016/bs.host.2018.07.006

Ferretti, J., Randazzo, V., Cirrincione, G., & Pasero, E. (2020). 1-d convolutional neural network for ECG arrhythmia classification. *Progresses in artificial intelligence and neural systems* (pp. 269–279). Springer Singapore. https://doi.org/10.1007/978-981-15-5093-5_25

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256). PMLR. url%7Bhttp://proceedings.mlr.press/v9/glorot10a.html%7D

Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Guo, Y., Cao, X., Liu, B., & Gao, M. (2020). Cloud detection for satellite imagery using attention-based u-net convolutional neural network. *Symmetry*, *12*(6). https://doi.org/10.3390/sym12061056

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Deep residual learning for image recognition.

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

He, K., Zhang, X., Ren, S., & Sun, J. (2015c). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Hiep, P., & Joo, R. (2018). A deep learning approach for classification of cloud image patches on small datasets. *Journal of Information and Communication Convergence Engineering*, *16*(3), 173–178. https://doi.org/https://doi.org/10.6109/JICCE.2018.16.3.173

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2261–2269. https://doi.org/10.1109/CVPR.2017.243

Huang, L., Liu, L., Jiang, L., & Zhang, T. (2018). Automatic mapping of thermokarst landforms from remote sensing images using deep learning: A case study in the northeastern tibetan plateau. *Remote Sensing*, *10*(12). https://doi.org/10.3390/rs10122067

Huang, L., Liu, L., Luo, J., Lin, Z., & Niu, F. (2021). Automatically quantifying evolution of retrogressive thaw slumps in beiluhe (tibetan plateau) from multi-temporal cubesat images. *International Journal of Applied Earth Observation and Geoinformation*, *102*, 102399. https://doi.org/https://doi.org/10.1016/j.jag.2021.102399

Huang, L., Luo, J., Lin, Z., Niu, F., & Liu, L. (2020). Using deep learning to map retrogressive thaw slumps in the beiluhe region (tibetan plateau) from cubesat images. *Remote Sensing of Environment*, *237*, 111534. https://doi.org/https://doi.org/10.1016/j.rse.2019.111534

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift.

Jegou, S., Drozdzal, M., Vazquez, D., Romero, A., & Bengio, Y. (2017). The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Katanforoosh, K. e. a. (2019). Parameter optimization in neural networks. %5Curl%7Bdeeplearning.ai%7D

Kemker, R., Salvaggio, C., & Kanan, C. (2018). Algorithms for semantic segmentation of multispectral remote sensing imagery using deep learning.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima.

Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization.

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks.

Kokelj, S., Tunnicliffe, J., Lacelle, D., Lantz, T., Chin, K., & Fraser, R. (2015). Increased precipitation drives mega slump development and destabilization of ice-rich permafrost terrain, northwestern canada. *Global and Planetary Change*, *129*, 56–68. https://doi.org/https://doi.org/10.1016/j.gloplacha.2015.02.008

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 1097–1105.

Lantuit, H., & Pollard, W. (2008). Fifty years of coastal erosion and retrogressive thaw slump activity on herschel island, southern beaufort sea, yukon territory, canada [Paraglacial Geomorphology: Processes and Paraglacial Context]. *Geomorphology*, *95*(1), 84–102. https://doi.org/https://doi.org/10.1016/j.geomorph.2006.07.040

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Lewkowicz, A. G., & Way, R. G. (2019). Extremes of summer climate trigger thousands of thermokarst landslides in a high arctic environment. *Nature Communications*, *10*(1), 1329. https://doi.org/10.1038/s41467-019-09314-7

Li, Y., Zhang, H., Xue, X., Jiang, Y., & Shen, Q. (2018). Deep learning for remote sensing image classification: A survey. *WIREs Data Mining and Knowledge Discovery*, *8*(6), e1264. https://doi.org/https://doi.org/10.1002/widm.1264

Lin, M., Chen, Q., & Yan, S. (2014). Network in network.

Liu, W., Rabinovich, A., & Berg, A. C. (2015). Parsenet: Looking wider to see better.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation.

Lu, L. (2020). Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, *28*(5), 1671–1706. https://doi.org/10.4208/cicp.oa-2020-0165

Ma, J., Chen, J., Ng, M., Huang, R., Li, Y., Li, C., Yang, X., & Martel, A. L. (2021). Loss odyssey in medical image segmentation. *Medical Image Analysis*, *71*, 102035. https://doi.org/https://doi.org/10.1016/j.media.2021.102035

Ma, L., Liu, Y., Zhang, X., Ye, Y., Yin, G., & Johnson, B. A. (2019). Deep learning in remote sensing applications: A meta-analysis and review. *ISPRS Journal of Photogrammetry and Remote Sensing*, *152*, 166–177. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2019.04.015

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*.

Maggiori, E., Tarabalka, Y., Charpiat, G., & Alliez, P. (2016). Fully convolutional neural networks for remote sensing image classification. *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 5071–5074. https://doi.org/10.1109/IGARSS.2016.7730322

Malkomes, G., Cheng, B., Lee, E. H., & Mccourt, M. (2021). Beyond the pareto efficient frontier: Constraint active search for multiobjective experimental design. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 7423–7434). PMLR. https://proceedings.mlr.press/v139/malkomes21a.html

Milletari, F., Navab, N., & Ahmadi, S.-A. (2016). V-net: Fully convolutional neural networks for volumetric medical image segmentation. *2016 Fourth International Conference on 3D Vision (3DV)*, 565–571. https://doi.org/10.1109/3DV.2016.79

Mishra, P., & Sarawadekar, K. (2019). Polynomial learning rate policy with warm restart for deep neural network. *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, 2087–2092. https://doi.org/10.1109/TENCON.2019.8929465

Mountrakis, G., Im, J., & Ogole, C. (2011). Support vector machines in remote sensing: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, *66*(3), 247–259. https://doi.org/https://doi.org/10.1016/j.isprsjprs.2010.11.001

Murton, J. (2021). Periglacial processes and deposits. In D. Alderton & S. A. Elias (Eds.), *Encyclopedia of geology (second edition)* (Second, pp. 857–875). Academic Press. https://doi.org/https://doi.org/10.1016/B978-0-12-409548-9.11925-6

Nabiyev, N., & Malekzadeh, S. (2021). Anomalous sound localization estimation. https://doi.org/10.13140/RG.2.2.25949.95201

Nitze, I., Grosse, G., Jones, B., Romanovsky, V., & Boike, J. (2018). Remote sensing quantifies widespread abundance of permafrost region disturbances across the arctic and subarctic. *Nature Communications*, *9*. https://doi.org/10.1038/s41467-018-07663-3

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1520–1528. https://doi.org/10.1109/ICCV.2015.178

Olthof, I., Fraser, R. H., & Schmitt, C. (2015). Landsat-based mapping of thermokarst lake dynamics on the tuktoyaktuk coastal plain, northwest territories, canada since 1985. *Remote Sensing of Environment*, *168*, 194–204. https://doi.org/https://doi.org/10.1016/j.rse.2015.07.001

Osterkamp, T., & Jorgenson, M. (2009a). Permafrost conditions and processes. *Geological Monitoring*. Geological Society of America. https://doi.org/10.1130/2009.monitoring(09)

Osterkamp, T., & Jorgenson, M. (2009b). Permafrost conditions and processes. *Geological Monitoring*. Geological Society of America. https://doi.org/10.1130/2009.monitoring(09)

Osterkamp, T., & Jorgenson, M. (2005). Response of boreal ecosystems to varying modes of permafrost degradation in alaska. *Canadian Journal of Forest Research*, *35*, 2100–2111. https://doi.org/10.1139/x05-153

Panchbhaiyye, V., & Ogunfunmi, T. (2020). A fifo based accelerator for convolutional neural networks. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1758–1762. https://doi.org/10.1109/ICASSP40776.2020.9053228

Philipp, M., Dietz, A., Buchelt, S., & Kuenzer, C. (2021). Trends in satellite earth observation for permafrost related analyses—a review. *Remote Sensing*, *13*(6). https://doi.org/10.3390/rs13061217

Rahman, M. A., & Wang, Y. (2016). Optimizing intersection-over-union in deep neural networks for image segmentation. *Advances in visual computing* (pp. 234–244). Springer International Publishing. https://doi.org/10.1007/978-3-319-50835-1_22

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation.

Rowland, J. C., Jones, C. E., Altmann, G., Bryan, R., Crosby, B. T., Hinzman, L. D., Kane, D. L., Lawrence, D. M., Mancino, A., Marsh, P., McNamara, J. P., Romanvosky, V. E., Toniolo, H., Travis, B. J., Trochim, E., Wilson, C. J., & Geernaert, G. L. (2010). Arctic landscapes in transition: Responses to thawing permafrost. *Eos, Transactions American Geophysical Union*, *91*(26), 229–230. https://doi.org/https://doi.org/10.1029/2010EO260001

Ruder, S. (2017). An overview of gradient descent optimization algorithms.

Schaefer, K., Lantuit, H., Romanovsky, V., Schuur, E., & Witt, R. (2014). The impact of the permafrost carbon feedback on global climate. *Environmental Research Letters*, *9*, 085003. https://doi.org/10.1088/1748-9326/9/8/085003

Shanmugamani, R. (2018). *Deep learning for computer vision: Expert techniques to train advanced neural networks using tensorflow and keras*. Packt Publishing. https://books.google.co.uk/books?id=6tRJDwAAQBAJ

Shao, Z., Zhou, W., Deng, X., Zhang, M., & Cheng, Q. (2020). Multilabel remote sensing image retrieval based on fully convolutional network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *13*, 318–328. https://doi.org/10.1109/JSTARS.2019.2961634

Shenoy, A. (2019). *Feature optimization of contact map predictions based on inter-residue distances and u-net++ architecture* (Doctoral dissertation). Stockholm University. https://doi.org/10.13140/RG.2.2.16796.23688

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Stock, S., Armengol Urpi, A., Kovacs, B., Maier, H., Gerdes, M., Stork, W., & Sarma, S. (2020). A system approach for closed-loop assessment of neuro-visual function based on convolutional neural network analysis of eeg signals. *Neurophotonics*, 8. https://doi.org/10.1117/12.2554417

Sultana, F., Sufian, A., & Dutta, P. (2020). Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*, *201-202*, 106062. https://doi.org/https://doi.org/10.1016/j.knosys.2020.106062

Sun, W., & Wang, R. (2018). Fully convolutional networks for semantic segmentation of very high resolution remotely sensed images combined with dsm. *IEEE Geoscience and Remote Sensing Letters*, *15*(3), 474–478. https://doi.org/10.1109/LGRS.2018.2795531

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Vladimir Iglovikov, V. O., Sergey Mushinskiy. (2017). Satellite imagery feature detection using deep convolutional neural network: A kaggle competition.

Xu, Y., Du, B., & Zhang, L. (2018). Multi-source remote sensing data classification via fully convolutional networks and post-classification processing. *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, 3852–3855. https://doi.org/10.1109/IGARSS.2018.8518295

Yang, J., Hines, E., Guymer, I., Iliescu, D., Leeson, M., King, G., & Li, X. (2008). A genetic algorithm-artificial neural network method for the prediction of longitudinal dispersion coefficient in rivers. *Advancing Artificial Intelligence Through Biological Process Applications*, 358–374. https://doi.org/10.4018/978-1-59904-996-0.ch019

Yi, Y., Zhang, Z., Zhang, W., Zhang, C., Li, W., & Zhao, T. (2019). Semantic segmentation of urban buildings from vhr remote sensing imagery using a deep convolutional neural network. *Remote Sensing*, *11*(15). https://doi.org/10.3390/rs11151774

Yu, F., & Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions.

Yuan, Q., Shen, H., Li, T., Li, Z., Li, S., Jiang, Y., Xu, H., Tan, W., Yang, Q., Wang, J., Gao, J., & Zhang, L. (2020). Deep learning in environmental remote sensing: Achievements and challenges. *Remote Sensing of Environment*, *241*, 111716. https://doi.org/https://doi.org/10.1016/j.rse.2020.111716

Zeiler, M. D., & Fergus, R. (2013). Visualizing and understanding convolutional networks.

Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2017). Pyramid scene parsing network.

Zheng, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., & Torr, P. H. S. (2015). Conditional random fields as recurrent neural networks. *2015 IEEE International Conference on Computer Vision (ICCV)*. https://doi.org/10.1109/iccv.2015.179

Zhong, Y., Fei, F., Liu, Y., Zhao, B., Jiao, H., & Zhang, L. (2017). Satcnn: Satellite image dataset classification using agile convolutional neural networks. *Remote Sensing Letters*, *8*, 136–145.