Technische Universität Berlin

Faculty of Electrical Engineering and Computer Science
Dept. of Computer Engineering and Microelectronics
Remote Sensing Image Analysis Group

Deep Learning based Image Compression and Hashing in Large-Scale Remote Sensing Archives

Master of Science in Geoinformation and Geodesy

October, 2021

Jun Xiang

Matriculation Number: 404895

Supervisor: Prof. Dr. Olaf Hellwich
Prof. Dr. Begüm Demir

Advisor: Gencer Sümbül
Dr. Nimisha Vishwanathan

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Sämtliche benutzten Informationsquellen sowie das Gedankengut Dritter wurden im Text als solche kenntlich gemacht und im Literaturverzeichnis angeführt. Die Arbeit wurde bisher nicht veröffentlicht und keiner Prüfungsbehörde vorgelegt.

Hereby I declare that I wrote this thesis myself with the help of no more than the mentioned literature and auxiliary means.

Berlin, Date 29/10/2021

Jun Xiang
.....

Name Surname

Acknowledgements

I would like to thank Prof. Dr. Begüm Demir, Department of Computer Engineering and Microelectronics, TU Berlin, for her supervision and great support during the production of this work. Also, I would like to thank my advisors, Gencer Sümbül and Dr. Nimisha Vishwanathan for their constant guidance and assistance during the realization of this thesis. My sincere thanks also goes to Dr. Mahdyar Ravanbakhsh for his suggestions and help at the early stage of this work.

Abstract

This thesis proposes a multi-task framework that simultaneously generates compressed bitstream and hashcodes in large-scale remote sensing (RS) archives. The proposed framework consists of an auto-encoder-based compression part and a shallow CNN-based hashing part with an attention module. The thesis also proposes a novel optimization scheme for training this multi-task framework. The optimization is divided into two stages. In stage one, the CNN-based compression part is optimized by a multiple-gradient descent algorithm (MGDA) to obtain a range of bit rates. This is followed by the second stage where the hashing part is jointly trained with the pre-trained compression part from stage one. During the joint training, a very small learning rate is set to the compression part and the hashing loss is optimized by projecting conflicted gradients (PCGrad) among sub losses. Performance analysis of the joint training shows that this proposed optimization strategy achieves compression performance at par with the single-task pre-trained compression baseline, and also achieves a competitive retrieval performance when compared to the single-task hashing baseline. The retrieval performance of the proposed framework is invariant to the position of the hashing part in the image domain or the compressed domain. Additionally, the optimization scheme proposed is fully automated and no hyperparameter optimization is required.

Contents

List of Acronyms vi

List of Figures vii

List of Tables viii

1 Introduction 1
- 1.1 Background and Motivation . . . 1
- 1.2 Objective . . . 4
- 1.3 Outline . . . 4

2 Related Work 5
- 2.1 Overview of Learning-based Compression Methods for Remote Sensing Images 5
- 2.2 Overview of Deep Supervised Hashing Algorithms for Remote Sensing Image Retrieval . . . 9
- 2.3 Overview of Optimization Techniques in Multi-Task Learning . . . 13
  - 2.3.1 Loss Weighting . . . 13
  - 2.3.2 Gradient Modulation . . . 13
  - 2.3.3 Multi-Objective Optimization . . . 15

3 Proposed Framework on Learning-based Compression and Hashing 17
- 3.1 Introduction to Proposed Multi-task Framework . . . 18
  - 3.1.1 Introduction to the Compression Part . . . 19
  - 3.1.2 Introduction to the Hashing Part . . . 21
- 3.2 Optimization of Proposed Multi-task Framework . . . 23
  - 3.2.1 Optimization of the Compression Part . . . 23
  - 3.2.2 Optimization of the Hashing Part . . . 25

4 Dataset and Experiments 26
- 4.1 Dataset . . . 26
- 4.2 Experimental Setups for the Proposed Framework . . . 28
  - 4.2.1 Experimental Setup for the Compression Part . . . 28
  - 4.2.2 Experimental Setup for the Hashing Part . . . 31
  - 4.2.3 Performance Analysis of the Hashing Part . . . 33

5 Experimental Results and Discussion **35**
5.1 Training Results of the Proposed Framework 35
5.1.1 Training Results of the Compression Part 35
5.1.2 Training Results of the Hashing Part 39
5.1.3 Performance Analysis of the Hashing Part 42

6 Conclusion **51**

Bibliography **53**

List of Acronyms

AE	Arithmetic Encoder
AD	Arithmetic Decoder
ANN	Approximate Nearest Neighbor
BPP	Bits Per Pixel
CBIR	Context Based Image Retrieval
CLC	CORINE Land Cover
CNN	Convolutional Neural Network
D	Distortion
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
DWA	Dynamic Weight Averaging
GDN	Generalized Divisive Normalization
LSTM	Long Short-Term Memory
MGDA	Multiple-Gradient Descent Algorithm
MTL	Multi-Task Learning
NTD	Non-negative Tucker Decomposition
PCA	Principal Components Analysis
PCGrad	Project Conflicted Gradients
PSNR	Peak Signal-to-Noise Ratio
Q	Quantization
R	bit-Rate
RD	Rate-Distortion
RNN	Recurrent Neural Network
RS	Remote Sensing
SA	Spectral Angle
SSIM	Structural Similarity Index Measure
STE	Straight-Through Estimator
TD	Tucker Decomposition

List of Figures

1.1 General diagram of proposed multi-task framework 2

2.1 General diagram of learning-based CNN compression method 6
2.2 General diagram of supervised deep hashing 10
2.3 Case N=2: possible positions of vector $\mathbf{u}$ wrt. two gradient vectors $\mathbf{g}_1$ and $\mathbf{g}_2$ . . 15

3.1 Overall architecture of the proposed multi-task framework 18
3.2 Visualization of Pareto points during the training of CNN compression part . . 23

4.1 Spectral bands and corresponding ground resolutions of Sentinel-2 images . . . 26
4.2 Example patch visualization 27
4.3 The architecture of RNN compression network 28
4.4 Overall architecture of hashing baseline 31
4.5 General diagram of proposed multi-task framework with the hashing part at the decoder side 34

5.1 Training results of the CNN compression part 35
5.2 Gradient weights generated by MGDA during the training of the CNN compression part 36
5.3 Reconstructed multi-spectral image at different bit-rates by the CNN compression part 37
5.4 Comparison of compression performance on BigEarthNet Serbia summer area . 38
5.5 Comparison of compression and decompression time on BigEarthNet Serbia summer area 38
5.6 Comparison of retrieved images of hashing baseline at hashbits 16, 32 and 64 . 40
5.7 Comparison of retrieved images between joint training and hashing baseline . . 41
5.8 Joint training results with different learning rates of the compression part . . . 42
5.9 Joint training results with or without the attention module in the hashing part . 43
5.10 Comparison of retrieved images when joint training with or without attention module in the hashing part 44
5.11 Comparison of retrieved images when joint training with different compression parts 46
5.12 Comparison of retrieved images in the image domain and in the bitstream domain 49
5.13 Comparison of retrieved images in the bitstream domain when joint training with different compression parts 50

List of Tables

2.1 PCGrad Update Rule 14

3.1 Structure of the hashing part, S and P are short for stride and padding 21

4.1 Choices of activation functions for hash coding 31
4.2 Default setting of joint training of the proposed framework 33

5.1 Retrieval performance of CNN hashing baseline at hashbits 32 with different combinations of activation functions and optimization methods 39
5.2 Retrieval performance of CNN hashing baseline when Greedy Hash is the activation function in the hashing layer and optimization method is PCGrad 39
5.3 Comparison of retrieval performance between joint training and hashing baseline 41
5.4 Comparison of retrieval performance when joint training with or without attention module in hashing part 44
5.5 Comparison of retrieval performance when joint training with different compression parts 45
5.7 Comparison of retrieval performance when joint training with hashing module at encoder or decoder side 47

1 Introduction

1.1 Background and Motivation

With the rapid development of space remote sensing (RS) technology in recent years, the temporal, spatial, and spectral resolution of remote sensing images have been continuously improved, the corresponding data scale has increased dramatically. For example, by the end of 2019, the total volume of data that was archived from the Sentinel missions was estimated to be more than 12.5 PB, and this massive volume of RS data still keeps growing by the speed of 10 Terabytes (TB) every day [27]. Also, with gradual reduction of satellite build and launch cost, more commercial small low-orbit satellites are launched to capture high resolution RS images, for example, until 2020 the company Planet has already about 200+ small satellites in orbit collecting 350 million km^2 of multi-spectral imagery daily [50]. Such huge and rapid growing volume of RS data makes it hard to store and search for relevant images in these immense archives. Thus it is important to develop efficient techniques for RS image compression and retrieval.

Traditional compression algorithms for multi-spectral images can be mainly summarized to the following three categories: (1) prediction-based approach; (2) vector quantization approach; (3) transform-based approach. The prediction-based approach is mainly applied to lossless compression. It uses the correlation between pixels to predict the unknown data based on its neighbors, then encodes the residual between the real value and the predicted value. This type of compression method has low complexity, however, the compression ratio is also considerably low, which is not a good option for massive volume of RS data. As for vector quantization approach, it uses codebooks to group and quantize a large set of vectors for compression. It can achieve moderate compression ratio comparing to prediction-based approach, however it is very slow since obtaining these codebooks is quite time-consuming during the training. The most widely used traditional compression method for multi-spectral images is transform-based approach due to its high compression ratio and fast calculations. This algorithm reduces the correlation between pixels by converting the data to frequency domain, so that information can be concentrated, quantified and encoded.

With the rapid development of deep learning, learning-based compression frameworks [3, 4, 9, 46] have achieved much better performance than traditional compression methods like JEPG2000, WebP, BPG etc on RGB images. Since RGB images have 3 channels, which can be considered as a special kind of multi-spectral image, these learning-based image compression methods for RGB images can also be adapted to multi-spectral images. Kong et al. [29] adapted end-to-end ResNet compression frameworks (ResConv) to multi-spectral images, which reported obtaining higher PSNR than that of JPEG2000 by about 2 dB, and the recovered images did not have obvious block effects which existed in those recovered by JPEG2000 or 3D-SPIHT [17]. Later Kong et al. [30] further improved the feature extraction module in the previous

ResConv, so that both spectral and spatial information in the multi-spectral images can be properly extracted and recovered. Kong reported that this improved compression algorithm outperformed the previous ResConv [29], JPEG2000 and 3D-SPIHT [17] on multi-spectral images of the Landsat 8 satellite and the WorldView-3 satellite.

The retrieval methods of RS images are also improved significantly due to deep learning. Traditional image retrieval techniques, also called metadata-based retrieval, strongly depend on metadata (such as keywords, tags, location, acquisition date or time) to extract images from the archive. However, the learning-based image retrieval can be done on the basis of the 'contents', when given a query image, similar images are returned independent of labor-intensive metadata. This content-based image retrieval (CBIR) requires efficient feature representations (descriptors) of the image data. Usually these image descriptors have high dimensions which not only occupy a lot of storage space but also increase the computation time for nearest neighbor search. To address these problems, hashing methods are introduced to encode high-dimensional image descriptors to low-dimensional hash codes, which can preserve the discriminative ability of the original image descriptors. When a query image comes for retrieval, the Hamming distance between hash codes of the query image and the searching data set are calculated. The candidates with relative smaller Hamming distance are selected for nearest neighbor search. Since the distance calculation only contains a simple binary XOR operation, it is very fast in practical applications. The time required for retrieval is linearly correlated with the number of images in the searching dataset, also the short hash codes can save a lot of storage space. Thus, deep learning-based hashing method is a good choice for large scale RS image retrieval.

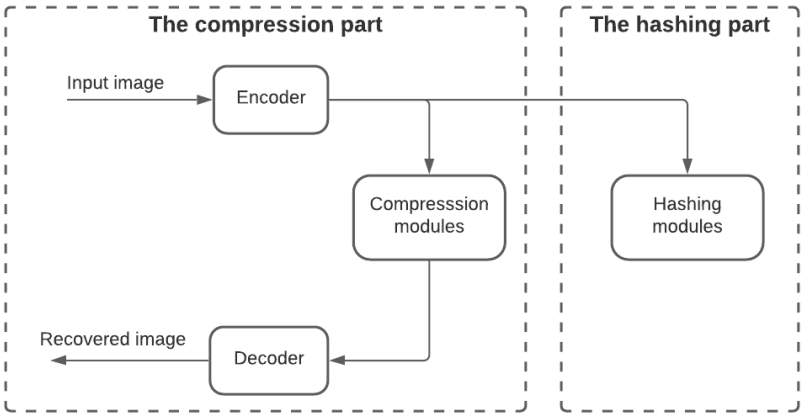


Figure 1.1: General diagram of proposed multi-task framework

The learning-based compression and hashing methods for RS image compression and retrieval are often tackled in isolation [71], i.e, a separate neural network is designed and trained for each of the compression and hashing tasks, which greatly waste computational resources in large-scale image applications. In this thesis, we explore the feasibility of a multi-task framework, which combines compression and hashing together.

Fig.1.1 shows the general diagram of the proposed multi-task framework. The backbone of the compression part is an encoder-decoder pair, in which the encoder extracts image features

from the input image for entropy encoding in compression modules, and the decoder reconstructs the input image from extracted image features. The hashing part is trained on image features extracted by the compression part. By joint training the compassion and hashing part, this trained multi-task framework can generate bitstream and hash codes simultaneously from the raw images.

The main challenge in joint training of this multi-task framework is that, the compression or hashing performance can be deteriorated by joint training together. As the objective of both tasks have different requirements on the the underlying shared image features, an optimization of joint training is difficult. The compression task requires the image features to be compact and contain enough information for image reconstruction. However, the hashing task does not need image features with such high reconstruction ability, since too much focus on trivial variations in the images may degrade the retrieval performance.

Another challenge lies in the training of the compression part for multiple compression rates, which can be extremely time-consuming if it is trained in the conventional way. The goal of the compression part is to get high compression rate (bit-rate) and good reconstructed quality (lower distortion) of the input image. These two objectives are mutually contrary, lower bit-rate usually leads to a higher distortion in the reconstructed image. A trained convolutional neural network (CNN) compression model is only able to get one optimal trade-off between the bit-rate and the distortion. In order to get multiple compression rates, the current CNN compression models have to be trained multiple times from scratch to get different optimal trade-offs, and the hyperparameters for different trade-offs are usually obtained by very computationally expensive grid-search.

The development of Multi-Task Learning (MTL) has provided many possible solutions for the above two challenges. We adopt MTL optimization techniques to solve the above two challenges. The first challenge can be solved by training the compression part first to secure the compression performance, then fine-tuning the compression part according to the needs of the hashing part. But how well can the hashing part perform by training it on top of the feature extractor which was trained on the compression task? Liu et al.[44] proposes task-specific attention modules which allows the network to emphasize the parts in the shared features which are more important for the corresponding task, and downplay the effect of unimportant parts. Inspired by Liu's work, an attention module is inserted to the hashing part for a better hashing performance. As for the second challenge of training CNN compression models for variable bit-rates, since the compression task is made of two conflicted sub-tasks: bit-rate minimization and image reconstruction, MTL optimization algorithm MGDA [16] or its variation MGDA-UB [54], which are developed to find Pareto optimal solutions for multiple-objective optimization, can be applied to get optimal trade-offs between the bit-rate and the distortion without any hyperparameter searching.

The main contributions of the thesis are as follows:

1 A multi-task deep learning framework for RS image compression and hashing as presented in Fig.1.1

2 An effective optimization strategy made of MTL optimization techniques for the proposed framework

1.2 Objective

The aim of this work is to design and train a multi-task deep learning framework which can compress RS images at desire rate-distortion (RD) ratio, also at the same time generate hash codes for fast CBIR in large-scale RS archives.

The optimization of the proposed framework includes two stages. The first stage is to train the compression part to a desired rate-distortion point. Then at the second stage, the hashing part is trained jointly with the pre-trained compression part, a very small learning rate is set to the compression part during the joint training. Performance analysis on different settings of the joint training shows that, this proposed training strategy can secure the compression performance, and achieve competitive hashing performance when compared to corresponding hashing baseline.

In case of RS image compression, we adapt the state-of-art CNN-based compression method proposed by Zhang et al. [11], which employs residual blocks, GDN [3] and attention modules in the backbone to extract compact image representations, and uses an entropy model made of context module, mixed Gaussian module and hierarchical priors for bit-rate optimization on the extracted image features. MGDA [16] is applied in training to achieve a variable bit-rate adaption. The state-of-art RNN-based compression method proposed by Islam et al. [23] is also investigated for comparison in the ablation study.

As for the objective of fast and accurate CBIR, a hashing network, which employs shallow convolution layers to downsize the image feature and applies GreedHash [57] for hashing coding, is designed to generate compact binary hash codes. The backbone of the compression part is served as the feature extractor for the designed hashing part, and an attention module is inserted before the hashing network for better hashing performance. PCGrad [70] is selected as the optimization method for the hashing loss. Several other designed hashing networks with different hashing coding layer are also investigated for comparison in the ablation study.

1.3 Outline

This chapter has provided an overview of the background and the motivation of the thesis. The rest of the thesis is organized into five chapters. **Chapter 2** introduces the existing compression and supervised hashing methods in the RS domain, as well as optimization algorithms in multi-task learning. **Chapter 3** introduces the proposed framework of learning-based compression and hashing. **Chapter 4** introduces the dataset, and discusses the experimental setups for ablation studies and performance analysis. **Chapter 5** represents the experiment results of ablation studies and performance analysis. **Chapter 6** summarizes the thesis and presents potential future work.

2 Related Work

2.1 Overview of Learning-based Compression Methods for Remote Sensing Images

The RS images applied in this work are multi-spectral images which contains multiple bands of wavelengths ranged from 380 nm to 3000 nm. They can be viewed as a three dimensional (3D) matrix which has two spatial dimensions and one spectral dimension. Naturally there are two kinds of redundancies in RS images: (1) spatial redundancy in neighboring pixels which are located in regions spanned by similar image features; (2) spectral redundancy in nearby bands which depict the same spatial area multiple times.

One of the earliest multi-spectral compression methods is CCSDS-MDC [2], a lossless compression method, which predicts the current band based on the previous band and encodes the residuals to bitstream. It has low complexity, but it also has a very low compression ratio. Later a lot of lossy compression methods are proposed to further increase the compression ratio, they can be divided into three types: (1) vector quantization approaches, which independently reduce clusters of pixels with similar characteristics by grouping them together; (2) transform-based approaches, which convert the image data to transform domain representation, so that the correlation among pixels can be reduced, common transform-based approaches are Karhunen-Loéve transform (KLT) [21], discrete cosine transform (DCT) [1] and discrete wavelet transform (DWT) [60]; (3) tensor decomposition-based approaches, such as Tucker Decomposition (TD) which decomposes a tensor into a set of matrices and one small core tensor, TD can be considered as a higher-order Principal Components Analysis (PCA).

In order to get a better compression ratio, the above traditional lossy compression methods are often combined to treat the spectral redundancy and spatial redundancy independently. For example, Qian et al. [18] applies PCA to reduce the correlation along the spectral dimension, and applies JPEG2000 (a DWT based compression method) to reduce the correlation in the spatial dimension. Karami et al. [26] applies TD on the DWT coefficients of spectral bands, which gains higher PSNR than PCA + JPEG2000 in [18]. Báscones et al. [5] proposes to use a vector quantization step to partition the spectral dimension space into Voronoi diagrams. Different PCAs are applied to further reduce the Voronoi diagrams to remove the correlation in the spectral dimension, then JPEG2000 is applied to different principal components to remove the spatial redundancy. Compared to PCA + JPEG2000 proposed in [18], [5] reports an increase of 1 to 3 dB in Signal Noise Ratio (SNR) for the same compression ratio. Despite the high compression performance of the above methods, they suffer of high calculation complexity, which is impractical for large-scale RS image compression.

In order to reduce the computation complexity of the tensor decomposition method, Li et al. [34] uses a forward CNN to reduce the scale of the original 3D RS images, then 3D-DCT

is applied to the small-scaled tensors to remove the spectral and spatial correlations. Later Non-negative Tucker Decomposition (NTD) is applied to the small-scale DCT tensors to further reduce the correlations, and an entropy encoder is used to encode the output core tensor to bitstream. In order to make sure the small-scaled DCT tensors keep good structural information of the original 3D RS images, a reconstruction loss is applied to the input image and the reconstructed image which is recovered from the small-scaled tensor. Compared to the conventional method that uses NTD and DCT for compression, [34] is faster in compression time with only a little sacrifice on PSNR, which does not impact the image quality.

Inspired by the success of learning-based compression methods for RGB images [3], Kong et al. [29] adapts an end-to-end CNN compression framework with optimized residual units for multi-spectral image compression. It consists of an auto-encoder for image feature extraction, a pair of quantizer and reverse quantizer, a rate-distortion optimizer for entropy encoding based on the importance map [35], an entropy encoder for compressing quantized features into bitstream, and an entropy decoder for decompressing bitstream. [29] outperforms conventional methods like JPEG2000 and 3D-SPIHT [17]. Later Kong et al. [30] improves the feature extractor (the auto-encoder) by adding two parallel modules, which can extract spatial features and spectral features separately. After the feature extraction, these two kinds of features are fused element-wise to form the feature map, which will be quantized and compressed by the entropy encoder. This improvement makes [30] outperform [29] and all the previous conventional compression methods like JPEG2000 and 3D-SPIHT [17] on datasets from Landsat-8 and WorldView-3 satellites.

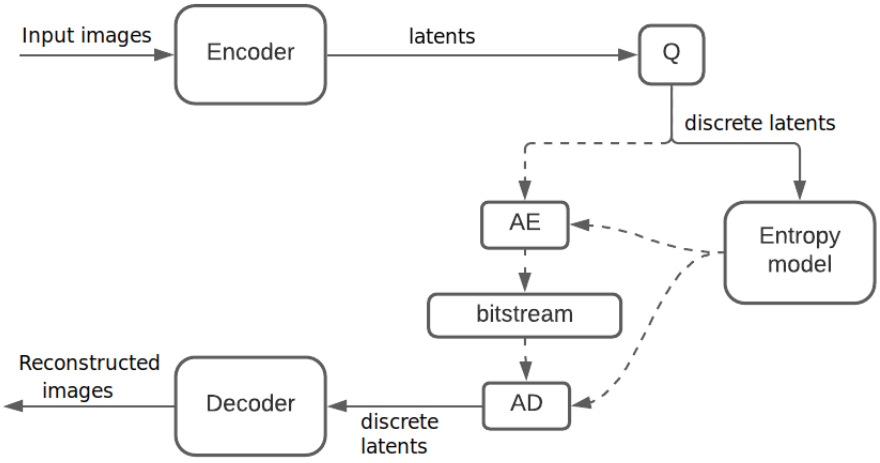


Figure 2.1: General diagram of learning-based CNN compression method

Fig.2.1 shows the general diagram of learning-based CNN compression methods. They usually consist of encoder and decoder pairs. Given an input image, the encoder and the quantization function Q generate the discrete feature representation (latent) of the image data. Then the image is recovered from the discrete latent by the decoder. Note that the quantization function Q is often approximated by the addition of uniform noise during the training due to the gradient back-propagation problem on discrete values. Arithmetic encoder (AE) is used to compress the

discrete latent into a bitstream. AE is a lossless compression method based on entropy encoding. When a string is compressed by AE, frequently used characters will be stored with fewer bits and vice versa, resulting in fewer bits used in total. Thus the number of bits (bit-rate) of compressed data is bound by the entropy of the latent. Since it is hard to get the real probability distribution of the discrete latent, it is necessary to use an entropy model to estimate it. Hence, the bit-rate is determined by the cross entropy of the real and estimated probability of the discrete latent.

The learning-based CNN compression frameworks can be improved by incorporating more powerful feature extracting modules in the encoder and decoder blocks, so that the spatial and spectral redundancy can be further reduced in the image data. Then less correlated latents can be generated, which leads to less bits for entropy encoding. GDN [3], residual blocks [62], and some more advanced convolutional architectures like attention module [11], and non-local networks [9] have been employed to optimize the network structure, which results in lower bit-rate for entropy encoding.

The performance of CNN compression frameworks can be more effectively improved by optimizing the entropy model. By learning a more accurate estimation of the probability distribution of the latents, the bit-rate for entropy encoding can be better controlled. In 2017 Ballé et al. [3] estimates the probability distribution of latents with a factorized density model, which is a non-parametric univariate density model defined in an explicit derivative. In 2018 Ballé et al. [4] adds a simple hyperprior model to extract hyper latents as side information for entropy encoding, which has greatly advanced the compression performance. In the same year, Cheng et al. [35] suggests generating an importance map from the image content, then uses this content weighted map to guide the bit-rate for different contents of the image. Also in 2018, Minnen et al. [47] proposes to use an autoregressive module (context model) to learn the spatial dependencies among elements of the latents, then this causal context model is combined with the hyperprior to get conditional entropy estimation of the latents. In 2019, Lee et al. [33] introduces another way to combine context model and hyperprior together to learn entropy parameters from the latents. The causal context model (masked convolution) can help with more accurate entropy estimation, however the sequential pixel-by-pixel operations caused by masked convolution have significantly slowed down the encoding and decoding speed. Later, Hu et al. [22] introduces a coarse-to-fine model, which estimates the conditional entropy from multi-layer latents without using a causal context model. In 2020, discretized Gaussian mixture likelihoods [12] are added to further improve the statistic description of latents to enhance entropy coding. Recently Tong et al. [9] suggests to use parallel 3D masked convolution in the context model, since it can break the original sequential pixel-by-pixel operations caused by 2D masked convolution, which would speed up the encoding and decoding process.

The core problem of lossy image compression is to minimize the bit-rate R, and the distortion D between input image and reconstructed image. Since these two tasks are mutually conflicted, a coefficient λ is introduced to control the rate-distortion trade-off. By optimizing $R + \lambda D$, an overall good rate-distortion performance can be obtained by jointly training the parameters of encoder, decoder and the entropy model. Usually multiple RD trade-offs are often required in practice. Since one λ only results in one RD trade-off, the CNN-based compression model has to be trained from scratch multiple times for different trade-offs. Also, expensive grid search has to be applied to get different λ. Thus, it is not trivial to train CNN-based compression models

for variable bit-rates.

The development of recurrent neural network (RNN) compression frameworks is relatively slow compared to that of CNN compression frameworks. Since Toderici et al. [63] in 2016 proposes the first RNN compression framework, the main components of this method have not changed much. There are three modules in a single iteration, i.e., an encoding network, a binarizer and a decoding network, where encoder and decoder contain recurrent network components. The residual signals between the input image patch and the reconstructed one from decoding network can be further compressed into bitstream in the next iteration. More specifically, the proposed method is an multi-iteration compression architecture supporting variable bit-rate compression in progressive style. Compared to the approximation of bit-rate estimation in CNN-based compression models, this RNN-based image compression scheme uses a scaled-additive coding framework to restrict the number of coding bits. To further improve the RNN-based image compression, Minnen et al. [14] presents a spatially adaptive image compression framework, in which the input image is divided into tiles which is similar to the existing image codecs such as the JPEG and JPEG2000. For each tile, an initial prediction is generated by a full CNN from the spatial neighboring tiles which have been decoded in the left and above regions. However, based on the released results, the proposed method only outperforms JPEG while it is inferior to JPEG2000.

Both CNN and RNN compression frameworks have its own pros and cons. In case of training time, the one-time feed-forward CNN networks take less time than RNN networks do, since the latter have a longer path of back-propagation due to multiple iterations in time. In case of support for variable range of bit-rates, RNN networks naturally can handle variable-rate compression by changing the number of iterations, however, CNN networks have to be trained separately for different RD coefficient λ. In case of encoding and decoding time, RNN networks may take more time than CNN networks when higher bit-rates are required, because RNN networks have to be executed for more iterations to generate higher bit-rates. Despite these pros and cons, both Johnston et al. [49] and Minnen et.al [47] in 2018 reported that the compression performance of CNN framework was generally higher than that of RNN frameworks. Since variable-bit-rate compression is widely required by applications, it is important to find an effective way for CNN compression networks to support variable range of bit-rates.

In this work, the state-of-art CNN-based compression method [11] is adopted for the proposed multi-task framework. MTL optimization techniques are applied in the training to get variable bit-rates. Also, we will compare this CNN-based compression method against the state-of-art RNN-based compression method [23] for completion.

2.2 Overview of Deep Supervised Hashing Algorithms for Remote Sensing Image Retrieval

Due to the huge and rapidly growing volume of RS data, it is computationally expensive to search and retrieve images by exhaustively comparing the query image with each image of the RS archive in high-dimensional feature space. In addition to that, the high-dimensional features of RS images also occupy a lot of storage space. To address these problems, hashing based approximate nearest neighbor (ANN) search schemes have attracted a large amount of research interest due to their high efficiency in both storage cost and search /retrieval speed. The purpose of the hashing algorithm is to map high dimensional features into a Hamming space, generating compact hash codes made of 0 and 1. In the case of CBIR, the hash codes generated by a good hash algorithm should be able to preserve the distance order of the original image space as much as possible. By this way, the most similar images to the query image can be efficiently retrieved based on the hamming distance with simple bit-wise operations, and the compact hash codes can save a lot of storage space comparing to the previous high-dimensional features.

Traditional hashing-based RS CBIR systems extract hand-crafted image features, and map these high-dimensional representations to low-dimensional binary codes by hashing functions [15, 36, 52]. Since the image feature extraction and hashing functions are mutually independent, the hash codes generated by traditional methods are sub-optimal in preserving the similarity of images in original semantic space. In order to optimize the alignment of similarity between the image feature space and the hash code space, several deep hashing based CBIR methods are introduced in the RS domain. For example, Li et al. [39] in 2018 proposes a supervised deep hashing neural network (DHNN), which contains a deep CNN for image feature learning and a fully connected layer for hash coding. Because of the gradient vanish problem when training with binary hash codes, DHNN uses approximated continuous hash-like codes to replace the binary hash codes during the training. DHNN applies likelihood pairwise loss to align the similarities of image pairs computed from the input space and the approximated hash-like code space. Also, it uses a quantization loss to narrow the gap between approximated hash-like codes and discrete hash codes, which gets competitive performance on the common RS dataset UCMD [69] and SAT4 [6].

Another representative deep hashing method in the RS domain is MiLAN [53], which employs a pre-trained Inception Net as image feature extractor and a shallow CNN for hash code learning. MiLAN combines a triplet loss, quantization loss and bit balance loss to learn a robust and efficient hashing network on the dataset UCMD [69]. The bit balance loss is to encourage each bit to have an approximately 50% chance of being 0 or 1, thereby maximizing code variance and information. The triplet loss is to preserve the similarity orders for more than two images that are computed in input space and hash code space. Compared to the pairwise loss, the triplet loss can avoid the situation that similar images are clustered together in the hash code space. Note that the performance of the triplet loss highly depends on triplet sampling methods. As it is difficult to sample triplets when the dataset is multi-label, the triplet loss can become computationally expensive because of triplet sampling when compared to the pairwise loss.

DHCNN [56] is another representative deep hashing method in the RS domain. It takes full advantage of image label information by adding a classification layer (fully-connected layer) to

the hash layer, so that predicted image labels can be obtained from the hash codes. By jointly optimizing the classification loss, pairwise loss and quantization loss, the hash codes generated by DHCNN are more discriminative, which avoids the disadvantage that similar images are clustered together in hash code space caused by pairwise loss.

After examining the above representative deep hashing methods in the RS domain, we can find that there are generally four issues in designing a deep supervised hashing method:

(1) Which kind of architecture to generate hash codes?
Fig 2.2 illustrates the common overall architecture of deep supervised hashing methods in the RS domain, which comprises an *Image feature extractor*, a *Hash network*, a *Hash layer* for hash coding, and a *Quantization* layer for generating binary hash codes. The choice of *Image feature extractor* often depends on the complexity of the dataset. In case of simple datasets, a shallow convolution network is enough. In case of more complex datasets like multi-label RS datasets BigEarthNet [58] or MLRSNet [48], deep learning models such as ResNet50, VGGNet and Inception are often utilized. The *Hash network* is to downsize high-dimensional image features to low-dimensional hidden features, it is usually made of several convolution layers with Relu or pooling operation in-between. The *Hash layer* is usually made of a fully connected layer and an activation function (sigmoid / tahn). The *Quantization* layer is a sign function.

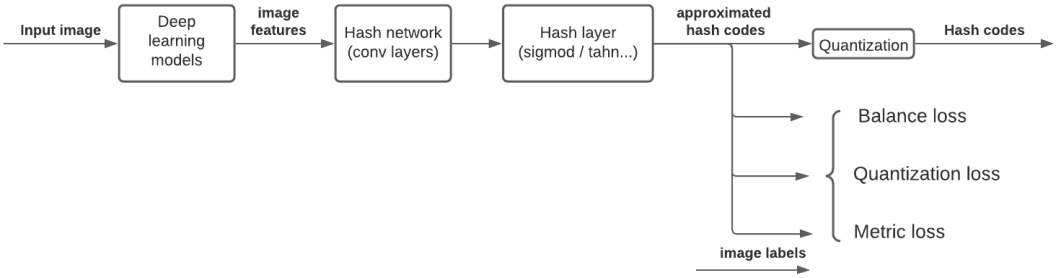


Figure 2.2: General diagram of supervised deep hashing

(2) How to obtain gradients when training with binary hash codes?
As for obtaining gradients when training with binary hash codes, a common way is to approximate the binary hash codes by the continuous output before the *Quantization* layer. Since this method doesn't optimize the hash codes directly, a quantization loss [39] is applied to push the approximated binary-like values close to the discrete hash codes. By this way, the sign function in the *Quantization* layer is gradually approximated by sigmoid/tahn function in the *Hash layer*. There are some other ways to solve the gradient vanish problem caused by training with binary hash codes. Su et al. [57] proposes Greedy Hash for hash coding. It uses the sign function in the forward pass to directly generate binary codes. In back propagation, the straight-through estimator (STE) [8] is applied to calculate the gradients by simply ignoring the sign function. Thus, the quantization loss is not required by Greedy Hash method. Li et al. [40] designs a Bi-half layer, which also adapts STE method to obtain gradients for binary codes. Differ to Greedy Hash, Bi-half

not only generates binary codes directly, but also makes binary codes approximate the optimal half-half bit distribution, so Bi-half does not need quantization loss and bit balance loss.

(3) How to design the metric loss to keep the similarity information of the original image space?
Designing the metric loss for similarity preservation is the essential goal of deep hashing. The similarity preservation can be generally categorized to two types: (1) pairwise similarity algorithms, which align the distances or similarities of a pair of items computed from the input space and the hash code space; (2) multiwise similarity algorithms such as the popular triplet loss, which preserve the similarity orders for more than two items that are computed in input space and hash code space.
Suppose there are N images in the current training batch. The multi-hot label vectors for i-th image and j-th image are $\boldsymbol{l}^{(i)}$ and $\boldsymbol{l}^{(j)}$, the hash code vectors are $\mathbf{b}^{(i)}$ and $\mathbf{b}^{(j)}$. Let the semantic similarity and distance of this image pair be denoted as s_{ij}^o and d_{ij}^o; the similarity and distance of the image pair in the hash code space be denoted as s_{ij}^h and d_{ij}^h. We can get:

$$\begin{aligned} s_{ij}^o &= \frac{<\boldsymbol{l}^{(i)}, \boldsymbol{l}^{(j)}>}{\|\boldsymbol{l}^{(i)}\|_2 \|\boldsymbol{l}^{(j)}\|_2}, \\ d_{ij}^o &= \|\boldsymbol{l}^{(i)} - \boldsymbol{l}^{(j)}\|_2, \\ s_{ij}^h &= <\mathbf{b}^{(i)}, \mathbf{b}^{(j)}>, \\ d_{ij}^h &= \|\mathbf{b}^{(i)} - \mathbf{b}^{(j)}\|_2, \end{aligned}$$

where $<.>$ means dot product, $\xi = \{(i,j) \mid i,j \in \{1,2,3,...N\}\}$
As summarized by Luo et al. [45], pairwise losses generally have three forms to align the similarity information between the input space and the hash code space.

- The similarity-distance product form
  Eg. min $\sum_{(i,j)\in\xi} s_{ij}^o d_{ij}^h$ in the pairwise loss of DSH [42] or PCDH [68], which tries to make semantically similar image pairs have smaller distance in hash code space.
- The difference form
  Eg. min $\sum_{(i,j)\in\xi} (s_{ij}^o - s_{ij}^h)^2$ or $\sum_{(i,j)\in\xi} (d_{ij}^o - d_{ij}^h)^2$ in the pairwise loss of CNNH [67] or DDSH [25].
- The likelihood form
  Eg. $\sum_{(i,j)\in\xi} \left(log(1+e^{s_{ij}^h}) - s_{ij}^o s_{ij}^h \right)$ in the pairwise loss of DHNN [39].

As for multiwise loss, it tries to preserve the relative similarity orders among more than two items in both input space and hash code space. The most popular multiwise loss is the triplet loss applied in DNNH [32] or MiLAN [53], it also has a negative likelihood form introduced by DTSH [64]:

$$\zeta(\mathbf{b}^{(a)}, \mathbf{b}^{(p)}, \mathbf{b}^{(n)}) = log(1 + e^{s_{ap}^h - s_{an}^h - m}) - (s_{ap}^h - s_{an}^h - m). \tag{2.1}$$

where a, p and n are indexes for the anchor, the positive and the negative samples. m is the margin.
The performance of triple loss can be significantly impacted by the triplet sampling methods. If there are many negative samples which satisfy $d^h_{an} > d^h_{ap} + m$, the triplet loss will be close to zero, which is hard to be reduced. It is necessary to sample more hard negative samples to make triplet loss work well. However, it is difficult to choose triplets especially when the training dataset is multi-label. Comparing to pairwise loss, triplet loss can be more computationally expensive because of triplet sampling.

(4) What other skills can improve the hashing performance?
Bit balance and bit independence are very common techniques to improve the hashing performance. Bit balance means that each bit has an approximately 50% chance of being +1 or -1, thereby maximizing code variance and information. DAPH [55] has showed that code balance is very significant for hashing. Bit independence means that different bits are uncorrelated. Since there is very little redundancy in the codes, a given set of bits can represent more information within a given code length. Bit independence is also added to the loss function by some representative hashing methods like SH-BDNN [61] and DAPH [55].
Classification loss is another skill to improve the hashing performance. For example, Luo et al. [45] investigated the retrieval performance of representative deep hashing methods on CIFAR-10 [31] and NUS-WIDE [13] datasets. The experiment results showed that DSDH[37] outperformed DPSH [38] evidently. These two methods are very similar except that DSDH adds a classification loss to the loss function to make the hash codes be more discriminative.
In addition to that, recent work like PCDH [68] reports that adding a pair sampling module can improve the optimization of its product-formed pairwise loss. PCDH employs a pair-sampling method called Pairwise Hard, which samples positive pairs with the maximum distance in the deep feature space, and samples negative pairs randomly with the distance smaller than the threshold. The pairs sampled by Pairwise Hard have large loss for effective hash code learning.

All above works are trained on fully annotated images. In this work, since we focus more on jointly training deep compression and hashing on multi-label RS datasets, we will design a deep supervised hashing module based on above-mentioned techniques. The exploration on more cost-effective hashing methods such as supervised hashing on cheep noisy data, semi-supervised or unsupervised hashing will be left to future works.

2.3 Overview of Optimization Techniques in Multi-Task Learning

In machine learning, it is typical to optimize the model for a particular metric, whether it is a precision score on a certain benchmark or a business KPI. In order to do this, a single model or ensemble of models are trained to perform the desired task. By laser-focusing on one single task, we may ignore a lot of other information from related tasks which may help to improve the metric. Also, it can be a big waste of computation resources when several tasks which share a lot of parameters are trained separately in different models. By jointly training different tasks which have shared representation, we can enable one model to optimize multiple tasks. This approach is called Multi-Task learning (MTL). A significant challenge in MTL comes from the optimization procedure itself. In particular, we need to address the optimization problems caused by dominant tasks or conflicted tasks during the joint training.

2.3.1 Loss Weighting

One common optimization technique in MTL is loss weighting. It adjusts the weights of different losses, and optimizes the aggregated weighted sum of all task-specific losses to get a global optimum for all the tasks. Kendall et al. [28] proposes to learn weights by the uncertainty to optimize multiple regression and classification objectives. Following Kendall et al., some researchers suggest to learn the weights by the learning speed on that task. Liu et al.[43] proposes Dynamic Weight Averaging (DWA), which explicitly set the loss weight of one task using a ratio of the current loss to a previous loss. Let the loss of task i at time step t be denoted as $\zeta_i(t)$, the weight of task i at time step t be denoted as $w_i(t)$, which can be formulated as:

$$\begin{aligned} r_i(t-1) &= \zeta_i(t-1)/\zeta_i(t-2), \\ w_i(t) &= \frac{Ne^{r_i(t-1)/T}}{\sum_j e^{r_j(t-1)/T}}, \end{aligned} \tag{2.2}$$

where N is the number of tasks, T is a temperature hyperparameter.
DWA uses the ratio of the losses from the last two training steps to set the loss weight for the current step. The loss weights can also be learned by the changing rate of the performance of each task. Dynamic Task Prioritization [20] uses the performance metrics other than the loss function to weigh tasks. Gong et al. [19] in 2019 conducts an empirical comparison of loss weighting by uncertainty and by speed, and finds that careful selection of task pairs is very important for loss weighting methods. In a lot of the cases, loss weighting methods can not improve MTL performance when a lot of conflicting gradient signals exist among randomly combined tasks.

2.3.2 Gradient Modulation

Another type of popular MTL optimization techniques is based on gradients. These gradient-based methods manipulate the back-propagation of the network. Inspired by DWA, Chen et al.[10] in 2018 introduces GradNorm, which doesn't scale losses explicitly. Instead, GradNorm

dynamically scales gradient norms on the last shared layer according to the learning speeds of different tasks. The weights are shifted among gradient magnitudes so that a higher learning speed yields gradients with a smaller magnitude, and a lower learning speed yields gradients with a larger magnitude. Chen's experiments show that GradNorm matches or outperforms exhaustive grid search methods on classification or regression tasks. However, similar to DWA, GradNorm only solves the optimization problems caused by dominated tasks, it doesn't work well when a lot of conflicted gradient signals exist among tasks. Yu et al. [70] in 2020 proposes PCGrad, which hypothesizes the following three conditions resulting in detrimental gradient interference:

- Negative gradient cosine similarity
  $CosineSimilarity = cos_{ij} = \frac{\mathbf{g}_i^T \mathbf{g}_j}{\|\mathbf{g}_i\|_2 \|\mathbf{g}_j\|_2} < 0$
- Small (close to zero) gradient magnitude similarity
  $MagnitudeSimilarity = \frac{2\|\mathbf{g}_i\|_2 \|\mathbf{g}_j\|_2}{\|\mathbf{g}_i\|_2^2 + \|\mathbf{g}_j\|_2^2} < 1$
- High multi-task curvature
  $Curvature = (1 - cos_{ij}^2) \frac{\|\mathbf{g}_i - \mathbf{g}_j\|_2^2}{\|\mathbf{g}_i + \mathbf{g}_j\|_2^2}$

where $\mathbf{g}_i, \mathbf{g}_j$ are gradients of task i and task j.
Gradient cosine similarity is to measure the angle between task gradients. A negative cosine value indicates the gradients of two tasks are conflicting. PCGrad suggests to avoid these detrimental gradient interference by projecting one task's gradient onto the normal plane of any other task's gradient when their gradients are conflicting. This projection operation can also help to solve the problems caused by small magnitude similarity and high learning curvature.
Suppose there are T sub tasks in total. Each task has its own loss function denoted as ζ_i, where $i \in \xi = \{1, 2, ..., T\}$. The model parameters are denoted as θ. Table2.1 shows the PCGrad Update rules for updating θ.

PCGrad Update Rule
1: for $i \in \xi$:
2: $\quad \mathbf{g}_i^{PC} = \nabla_\theta \zeta_i$
3: $\quad$ for random $j \in \xi$:
4: $\quad\quad \mathbf{g}_j = \nabla_\theta \zeta_j$
5: $\quad\quad$ if $i \neq j$ and $cos_{ij} = \frac{\mathbf{g}_i^{PC} \mathbf{g}_j^T}{\|\mathbf{g}_i\|_2 \|\mathbf{g}_j\|_2} < 0$:
6: $\quad\quad\quad \mathbf{g}_i^{PC} = \mathbf{g}_i^{PC} - \frac{\mathbf{g}_i^{PC} \mathbf{g}_j^T}{\|\mathbf{g}_j\|_2^2} \mathbf{g}_j$
7: $\Delta\theta = \sum_i^T \mathbf{g}_i^{PC}$

Table 2.1: PCGrad Update Rule

The limited part of PCGrad is that it only alters the gradients when negative cosine similarity exists among task pairs. It becomes inactive when there are detrimentally low positive cosine similarity between task pairs. Wang et al. [66] proposes GradVac to deal with both negative and

positive cases in gradient cosine similarity between task pairs. Javaloy and Valera [24] argue that the gradient updates provided by methods like PCGrad do not guarantee that the global optima moves towards the local optimal points of individual tasks, because conflicted gradients may indicate that the local optimal points exist in completely different parts of the shared parameter space. Thus, they introduce RotoGrad, which can cooperate with GradNorm to homogenize the per-task gradient magnitudes, but also brings the local optima of different tasks closer to each other by rotating the shared-representation space.

2.3.3 Multi-Objective Optimization

Though DWA, GradNorm or PCGrad works well on classification, regression and segmentaion tasks, they can't work on mulit-objective problems like bitrate and reconstruction in deep compression, in which tasks are so conflicted that the performance of one task can only be improved by worsening the performance of another task. To address the optimization of conflicted objectives, Désidéri proposes a multiple-gradient descent algorithm (MGDA) [16] to converge the training to an optimal trade-off point among task losses, which is the best feasible solution for solving multi-objective problems. This optimal trade-off point is also called Pareto optimal point, which should satisfy the following condition:

$$\begin{aligned} &\mathbf{g}_i = \nabla_\theta \zeta_i, \quad i \in [1,2,3,...T], \\ &w_i \geq 0 (\forall i), \quad \sum_i^T w_i \mathbf{g}_i = \mathbf{0}, \quad \sum_i^T w_i = 1, \end{aligned} \tag{2.3}$$

where ζ_i is the loss for task i, $\mathbf{g}_i$ is the gradient of task i, T is the total number of tasks, w_i is the weight for task i.
If the current point is not Pareto optimal, the descent direction can be obtained by optimizing the following problem:

$$min\left\{ \|\mathbf{u}\|_2^2 \;\middle|\; \mathbf{u} = \sum_i^T w_i \mathbf{g}_i, \sum_i^T w_i = 1, w_i \geq 0, \forall i \right\}. \tag{2.4}$$

In case it is a two-objective optimization problem (T=2), the above optimization problem can be rewritten as:

$$min\left\{ \|\mathbf{u}\|_2^2 \;\middle|\; \mathbf{u} = w\mathbf{g}_1 + (1-w)\mathbf{g}_2, 0 \leq w \leq 1 \right\}. \tag{2.5}$$

As shown in Fig.2.3, the gradient descent direction is along vector u (blue). It is either a perpendicular case or an edge case.

Figure 2.3: Case N=2: possible positions of vector $\mathbf{u}$ wrt. two gradient vectors $\mathbf{g}_1$ and $\mathbf{g}_2$

In 2018 Sener and Koltum [54] extend MGDA to a form that scales well to an increasing number of tasks in deep learning by minimizing an upper bound to the MGDA loss. When the number of tasks increases, this improvement yields a smaller computational overhead comparing to traditional MGDA. Since MGDA is only able to produce one Pareto optimal point, in 2019 Lin et al. [41] extend MGDA further to find a set of Pareto optimal solutions by decomposing a given MTL problem into several sub-problems with a set of preference vectors, and finding one Pareto solution in each restricted preference region. Lin's method is more flexible and useful than a single solution. However, its performance highly depends on a proper selection of preference vectors, which requires a lot of empirical experiences on related objectives.

In the next chapter, we will apply the above MTL optimization techniques to gain competitive compression and hashing performance in the jointly training of both tasks without any grid-search for hyperparameters.

3 Proposed Framework on Learning-based Compression and Hashing

Given a training set of N images $\mathbb{X} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N)}\}$, where $\mathbf{x}^{(i)}$ is the i-th image vector, and its label sets $\mathbb{L} = \{\boldsymbol{l}^{(1)}, \boldsymbol{l}^{(2)}, ..., \boldsymbol{l}^{(N)}\}$, where $\boldsymbol{l}^{(i)}$ is the multi-hot label vector of i-th image. The goal of learning-based compression in the proposed framework is to learn a quantized representation $\hat{\mathbf{y}}$, and its estimated distribution $p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}})$ from a given input image vector $\mathbf{x}$, so that the given image can be encoded to bitstream with minimum bit-rates R, and can be reconstructed from bitstream with tolerated distortion D. Let the set of N image representations learned by the compression part be denoted as $\mathbb{Y}$, where $\mathbb{Y} = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, ..., \mathbf{y}^{(N)}\}$, the goal of hash learning in the proposed framework is to learn a mapping $f : \mathbb{Y} \mapsto \{-1, 1\}^q$, so that an input image feature vector $\mathbf{y}^{(i)}$ can be encoded into a q-bit binary vector $\mathbf{b}^{(i)}$, with the semantic similarities of images being kept aligned with the similarities in the hash code space. The semantic similarity is defined by the cosine similarity between $\boldsymbol{l}^{(i)}$ and $\boldsymbol{l}^{(j)}$. The similarity of image pairs in hash code space is defined by the dot product of $\mathbf{b}^{(i)}$ and $\mathbf{b}^{(j)}$.

In order to make sure the performance of the compression part not be deteriorated by the hashing part, the optimization of this proposed multi-task framework is divided into two stages. The first stage is to train the compression part to desired rate-distortion trade-off points. Traditionally it is non-trivial to train CNN-based compression networks for a variable range of bit-rates, i.e, the network has to be trained multiple times separately from scratch for different rate-distortion (RD) trade-off points, and grid search is required during the training to get the weights for each RD trade-off point. In this work, we propose to apply MGDA [16] to optimize the distortion and bit-rate loss, so that multiple optimal trade-offs can be obtained without grid-searching for hyperparameters, also without training the network multiple times from scratch for different bit-rates.

The second stage is to train the hashing part on the image features which were trained by the compression part. In order to get a competitive hashing performance compared to the single-task hashing baseline, an attention module is inserted into the hashing part to extract task-specific features from the shared image features. Also, a very small learning rate is set to the pre-trained compression part, so that the shared image features can be fine-tuned according to the needs of the hashing part. PCGrad [70] is applied to optimize the hashing loss.

The novelty of the proposed framework includes: i) Joint compression and hashing framework with performance equivalent to the baseline trained individual; ii) An optimization scheme using MTL optimization methods that avoids hyperparameter search.

3.1 Introduction to Proposed Multi-task Framework

As illustrated in Fig.3.1, the proposed framework consists of two parts: the compression part and the hashing part. The compression part adapts the state-of-art CNN-based image compression method, which consists of two sub-networks. The first sub-network is the core autoencoder (*Encoder* and *Decoder* blocks), which is responsible for image feature extraction and image reconstruction. The second sub-network is responsible for entropy encoding, which consists an auto-regressive model (*Context* model), *Entropy parameters* module and a hyper-network (*Hyper Encoder*, *Hyper Decoder* and *Factorized entropy* model). The hashing part contains an attention module [11] for task-specific feature extraction, a *Hash Network* for downsizing the image features, a *Hash Layer* for generating discrete hash codes and a *Classification Layer* for predicting image labels.

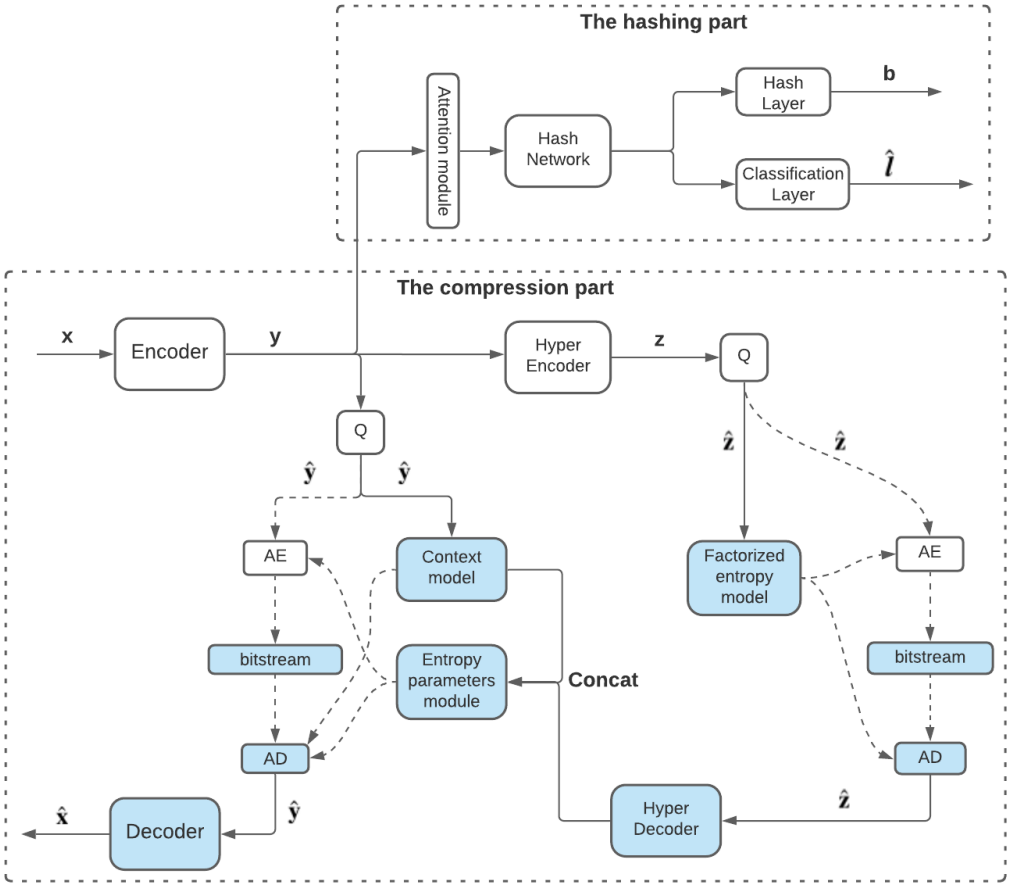


Figure 3.1: Overall architecture of the proposed multi-task framework

3.1.1 Introduction to the Compression Part

The processing flow of the first sub-network of the compression part is described in Eq.3.1. The *Encoder* block extracts latents $\mathbf{y}$ from the input image $\mathbf{x}$. During training, a uniform noise (function U) is added to latents to generate approximated discrete values ($\hat{y}$). During inference, a real round function Q is employed to get discrete values. The reconstructed image $\hat{x}$ is recovered by the *Decoder* block from the discrete latents. The above processing can be formulated as:

$$\begin{aligned} \mathbf{y} &= Encoder(\mathbf{x}), \\ \hat{\mathbf{y}} &= \begin{cases} \mathbf{y} + U(-\frac{1}{2}, \frac{1}{2}), & during\ training \\ round(\mathbf{y}), & otherwise \end{cases} \\ \hat{\mathbf{x}} &= Decoder(\hat{\mathbf{y}}). \end{aligned} \tag{3.1}$$

The probability estimation of the discrete hyper-latents is described in Eq.3.2. The latents $\mathbf{y}$ are fed to the *Hyper Encoder* block to get hyper-latents $\mathbf{z}$, then the same quantization function is applied to the hyper-latents to get discrete hyper-latents $\hat{\mathbf{z}}$. The *Factorized entropy* model in the hyper-network is to learn an arbitrary distribution $p_{\hat{\mathbf{z}}}$ from the quantized hyper-latents, $p_{\hat{\mathbf{z}}}$ is modeled by a non-parametric, fully factorized density model ψ, in which the vector $\psi^{(i)}$ contains parameters of each univariate distribution $p_{z_i|\psi^{(i)}}$. The above processing can be formulated as:

$$\begin{aligned} \mathbf{z} &= HyperEncoder(\mathbf{y}), \\ \hat{\mathbf{z}} &= \begin{cases} \mathbf{z} + U(-\frac{1}{2}, \frac{1}{2}), & during\ training \\ round(\mathbf{z}), & otherwise \end{cases} \\ p_{\hat{\mathbf{z}}|\psi}(\mathbf{z}|\psi) &= \prod_{i=1} \big(p_{z_i|\psi^{(i)}} * U(-\frac{1}{2}, \frac{1}{2})\big)(\hat{z}_i), \end{aligned} \tag{3.2}$$

where z_i denotes the i-th element of $\mathbf{z}$, and i specifies to the position of each element.

The conditional probability estimation of the discrete latents is described in Eq.3.3. The *Context* model learns the context-based predictions from the quantized latents $\hat{\mathbf{y}}$, and the hyper-network learns information from quantized hyper-latents $\hat{\mathbf{z}}$ to correct the context-based predictions. The outputs of *Context* model and the hyper-network are concatenated and fed to *Entropy parameters* module, which learns parameters (weights w, means μ and variances δ^2) for a mixture conditional Gaussian distribution $p_{\hat{\mathbf{y}}|\hat{\mathbf{z}}}(\hat{\mathbf{y}}|\hat{\mathbf{z}})$ of the quantized latents. The above processing can be formulated as:

$$\begin{aligned} p_{\hat{\mathbf{y}}|\hat{\mathbf{z}}}(\hat{\mathbf{y}}|\hat{\mathbf{z}}) &= \prod_i p_{\hat{\mathbf{y}}|\hat{\mathbf{z}}}(\hat{y}_i|\hat{\mathbf{z}}), \\ p_{\hat{\mathbf{y}}|\hat{\mathbf{z}}}(\hat{y}_i|\hat{\mathbf{z}}) &= \Big(\sum_{k=1}^{K} w_i^{(k)} N(\mu_i^{(k)}, \delta_i^{2(k)}) * U(-\frac{1}{2}, \frac{1}{2})\Big)(\hat{y}_i) \\ &= \sum_{k=1}^{K} w_i^{(k)} * \Big(c\Big(\frac{\hat{y}_i + \frac{1}{2} - \mu_i^{(k)}}{\delta_i^{(k)}}\Big) - c\Big(\frac{\hat{y}_i - \frac{1}{2} - \mu_i^{(k)}}{\delta_i^{(k)}}\Big)\Big), \end{aligned} \tag{3.3}$$

where i specifies the location of each element of the vector, i.e. y_i denotes the i-th element of the latent vector $\mathbf{y}$. k denotes the index of the Gaussian model, $k \in \{1, 2, 3\}$ by default, the k-th

mixture Gaussian model for each element $\hat{y}_i$ is characterized by three parameters, i.e weight $w_i^{(k)}$, mean $\mu_i^{(k)}$ and variance $\delta_i^{(k)}$. c(.) is a cumulative function of a standard normal distribution.

The core problem of lossy image compression is to minimize the distortion D between input image and reconstructed image, also to minimize the bit-rate R. In this work, the distortion D is measured by structural similarity index measure [65] (SSIM). The bit-rate is determined by the cross entropy of the real and estimated probability of the discrete latents. Since the real probability distributions are usually unknown, Eq.3.2 and Eq.3.3 are utilized to get estimated distributions of the discrete latents. Thus, the loss function of the compression part $\zeta_{compression}$ is formulated as:

$$\begin{aligned} \zeta_D &= 1 - SSIM(\mathbf{x}, \hat{\mathbf{x}}), \\ \zeta_R &= E\big[-log_2(p_{\hat{\mathbf{y}}}(\hat{\mathbf{y}}))\big] \\ &= E\big[-log_2(p_{\hat{\mathbf{y}}|\hat{\mathbf{z}}}(\hat{\mathbf{y}}|\hat{\mathbf{z}}))\big] + E\big[-log_2(p_{\hat{\mathbf{z}}|\psi}(\hat{\mathbf{z}}|\psi))\big], \\ \zeta_{compression} &= \zeta_R + \lambda\,\zeta_D. \end{aligned} \tag{3.4}$$

where λ is a coefficient for different rate-distortion trade-offs.

The network structure of the compression part is adapted from current state-of-art CNN compression network proposed by [11], in which the *Encoder* and *Decoder* blocks employ residual blocks [62] with GDN [3] activation and attention modules [11] for better image reconstruction and compression ratio. The *Context* model is a 5x5 2D mask convolution layer. The structures of the *Hyper Encoder* and *Hyper Decoder* are unsymmetrical. The output of *Context* model and *Hyper Decoder* are concatenated and sent to the *Entropy parameters* module (three 1x1 convolution layers) to get parameters of the mixture Gaussian model, which has 3xNxK channels. K is the number of Gaussian models, $K = 3$ by default. N is the number of channels in latents, $N = 192$ by default. The number of channels in hyper-latents is $M = 128$. Suppose the input image size is (B, C, 128, 128), where B is the batch size, C is the number of image channels. Then the output size of the latents will be (B, 192, 8, 8), and the output size of hyper latents will be (B, 128, 2, 2).

3.1.2 Introduction to the Hashing Part

The processing flow of the hashing part is described by Eq.3.5. The latents **y** from the compression part is fed to the *Hash Network* to get downsized hidden features **h**, which are sent to the *Hash Layer* to generate hash codes **b** valued in {1, -1}, and also sent to the *Classification Layer* to get predicted labels $\hat{\boldsymbol{l}}$. The above processing can be formulated as:

$$\begin{aligned} \mathbf{h} &= HashNetwork(\mathbf{y}), \\ \mathbf{b} &= HashLayer(\mathbf{h}), \\ \hat{\boldsymbol{l}} &= ClassificationLayer(\mathbf{h}). \end{aligned} \tag{3.5}$$

Let the size of input image features extracted from the compression part be denoted as (B, 192, 8, 8). The number of label classes is denoted as $|\boldsymbol{l}|$, where $|.|$ is the number of elements presented in a vector, i.e, $|\boldsymbol{l}| = 19$. The length of hashing bits is denoted as q. Table 3.1 shows the structure of the hashing part. Greed Hash [57] is selected as the activation function in the hash layer, which generates discrete hash codes directly in the forward pass, and uses the straight through estimation method (STE) [8] for back-propagation.

Component	Layer	S	P	Input Shape	Output Shape	Activation
Hash network	Conv 5x5	2	1	(B, 192, 8, 8)	(B, 512, 3, 3)	ReLu
	Conv 3x3	1	0	(B, 512, 3, 3)	(B, 512, 1, 1)	ReLu
Classification layer	Conv 1x1	1	0	(B, 512, 1, 1)	(B, $\|\boldsymbol{l}\|$, 1, 1)	Sigmoid
Hash layer	Conv 1x1	1	0	(B, 512, 1, 1)	(B, q, 1, 1)	Greed Hash [57]

Table 3.1: Structure of the hashing part, S and P are short for stride and padding

The soft pairwise loss [72] is selected for metric learning in the hash code space, as it can fully consider the rank difference of semantic pairwise similarity for multi-label images by dividing it into "hard similarity" and "soft similarity". In the case of "hard similarity", the image pair shares no common labels or shares all the labels; In the case of "soft similarity", the image pair shares labels partially. The cross-entropy loss and mean square loss are adapted to the pairwise loss of both cases respectively.
The soft pairwise loss is formulated as:

$$\begin{aligned} s^o_{ij} &= \frac{<\boldsymbol{l}^{(i)}, \boldsymbol{l}^{(j)}>}{\|\boldsymbol{l}^{(i)}\|_2 \|\boldsymbol{l}^{(j)}\|_2}, \\ s^h_{ij} &= <\mathbf{b}^{(i)}, \mathbf{b}^{(j)}>, \\ \zeta_{pairwise} &= \begin{cases} \sum_{(i,j)\in\xi} \left(log(1+e^{s^h_{ij}}) - s^o_{ij} s^h_{ij} \right), & s^o_{ij} \in \{0,1\} \\ \sum_{(i,j)\in\xi} \left(\left\| \frac{1}{2}(s^h_{ij} + q) - s^o_{ij} q \right\|_2^2 \right), & 0 < s^o_{ij} < 1 \end{cases} \end{aligned} \tag{3.6}$$

where ξ is the set of index pairs, $\xi = \{(i,j) |\ i,j \in \{1,2,3,...N\}\}$, $<\boldsymbol{l}^{(i)}, \boldsymbol{l}^{(j)}>$ calculates the inner product, s^o_{ij} is the semantic pairwise similarity between i-th image and j-th image, s^h_{ij} is the

pairwise similarity in hash code space between the image pair.
$\zeta_{pairwise}$ can be revised as:

$$m_{ij} = \begin{cases} 1 & s^o_{ij} \in \{0,1\} \\ 0 & 0 < s^o_{ij} < 1 \end{cases}$$
$$\zeta_{pairwise} = \sum_{(i,j)\in\xi} \Big[(1-m_{ij}) \Big(log(1+e^{\alpha s^h_{ij}}) - \alpha s^h_{ij} s^o_{ij} \Big) + \gamma (1-m_{ij}) \Big(\big\| \frac{1}{2}(s^h_{ij}+q) - s^o_{ij} q \big\|_2^2 \Big) \Big], \tag{3.7}$$

where m_{ij} is an indicator on the types of semantic similarity, $m_{ij} = 1$ indicates the hard similarity. $\alpha = 5/q, \gamma = 0.1/q$ by default.

Another important loss for better hashing performance is the balance loss, which pushes the codes to have a half-half distribution of -1 and +1, thereby maximizing code variance and information. The balance loss is formulated as:

$$\zeta_{balance} = \sum_{(i,j)\in\xi} \Big(\|(\mathbf{b}^{(i)^T}\mathbf{1})\|_2^2 + \|(\mathbf{b}^{(j)^T}\mathbf{1})\|_2^2 \Big), \tag{3.8}$$

where $\mathbf{1}$ is a vector of elements 1.

Classification loss is another helpful sub loss, which makes the hidden features $\mathbf{h}$ more discriminative by taking full advantage of image labels. The classification loss is formulated as:

$$\zeta_{classification} = \sum_{(i,j)\in\xi} \Big(\|\hat{\boldsymbol{l}}^{(i)} - \boldsymbol{l}^{(i)}\|_2^2 + \|\hat{\boldsymbol{l}}^{(j)} - \boldsymbol{l}^{(j)}\|_2^2 \Big), \tag{3.9}$$

where $\hat{\boldsymbol{l}}_i$ and $\hat{\boldsymbol{l}}_j$ are predicted multi-hot label vectors.

Finally, the hashing loss function $\zeta_{hashing}$ is formulated as:

$$\zeta_{hashing} = w_1 \zeta_{pairwise} + w_2 \zeta_{balance} + w_3 \zeta_{classification}, \tag{3.10}$$

where w_1, w_2, w_3 are weights for each individual loss.

3.2 Optimization of Proposed Multi-task Framework

In order to get competitive performance compared to the corresponding single-task baseline, the optimization of proposed multi-task framework is divided into two stages: (1) optimize the compression part to get desired RD trade-off points; (2) optimize the hashing part jointly with the pre-trained compression part.

3.2.1 Optimization of the Compression Part

Since bit-rate and distortion are two conflicted objectives, decreasing the bit-rate will increase the distortion. Thus, it is better to train one of them separately to get a desired value, i.e, train the distortion loss to its convergence, then jointly train both sub tasks with MGDA to get most nearby RD trade-off point. The training of the compression part is described as follows:

- Step 1: Train the distortion loss to its convergence, then save the model

$$\zeta_{compression} = \zeta_D = 1 - SSIM(\mathbf{x}, \hat{\mathbf{x}}). \tag{3.11}$$

  The convergence point of the distortion loss is illustrated as the red dot in Fig.3.2.

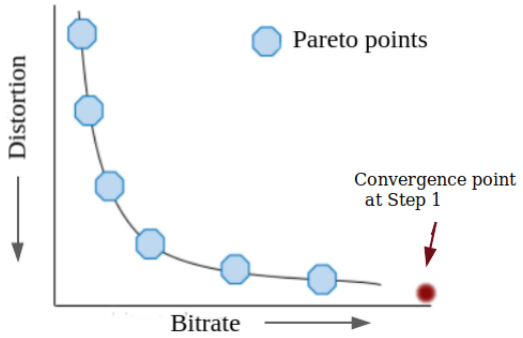


Figure 3.2: Visualization of Pareto points during the training of CNN compression part

- Step 2: Reload the saved model from stage 1, train bit-rate and distortion jointly with MGDA [16].

$$\zeta_{compression} = w_1\zeta_D + w_2\zeta_R. \tag{3.12}$$

  Suppose the model parameter is θ, the gradient vectors of the distortion loss and the bit-rate loss are $\nabla_\theta\zeta_D$ and $\nabla_\theta\zeta_R$. According to Eq.2.4, the gradient descent direction for Pareto points can be obtained by optimizing the following formula:

$$min\left\{ \|\mathbf{u}\|_2^2 \;\middle|\; \mathbf{u} = w_1\nabla_\theta\zeta_D + w_2\nabla_\theta\zeta_R, w_1 + w_2 = 1, w_1 \geq 0, w_2 \geq 0 \right\}, \tag{3.13}$$

where w_1 and w_2 can be written as:

$$w_1 = \begin{cases} 1, & \nabla_\theta \zeta_D^T \nabla_\theta \zeta_R \geq \nabla_\theta \zeta_D^T \nabla_\theta \zeta_D \\ 0, & \nabla_\theta \zeta_D^T \nabla_\theta \zeta_R \geq \nabla_\theta \zeta_R^T \nabla_\theta \zeta_R \\ \frac{(\nabla_\theta \zeta_R - \nabla_\theta \zeta_D)^T \nabla_\theta \zeta_R}{\|\nabla_\theta \zeta_R - \nabla_\theta \zeta_D\|_2^2}, & otherwise \end{cases} \tag{3.14}$$
$$w_2 = 1 - w_1.$$

Suppose θ is the parameter of the compression part, and the learning rate is η, the model parameters can be updated as:

$$\begin{aligned} \theta &= \theta - \eta \mathbf{u} \\ &= \theta - \eta (w_1 \nabla_\theta \zeta_D + w_2 \nabla_\theta \zeta_R). \end{aligned} \tag{3.15}$$

Since distortion and bit-rate loss are mutually conflicted, we can get the conclusion that $\nabla_\theta \zeta_D^T \nabla_\theta \zeta_R < 0$, the value of w_1 and w_2 will always between 0 and 1. In the early training of Step 2, the distortion loss ζ_D starts from its convergence point, which has a very small magnitude, w_1 will be very close to 1, and w_2 is very close to 0. As the training goes on with a small learning rate (1e-5), the bit-rate loss will decrease until it reaches its first Pareto point. Then by increasing the learning rate a little bit, the weight on bit-rate part (w_2) will gradually go up accordingly, and the bit-rate loss will decrease to reach another Pareto point. By adjusting the learning rate, we obtain multiple optimal trade-off points between distortion and bit-rate.

3.2.2 Optimization of the Hashing Part

After training the compression part to obtain a range of bit-rates, we load the pre-trained compression part at certain bit-rate to the proposed framework, and train the hashing part jointly. The following loss function is optimized during the joint training:

$$\zeta = \zeta_{compression} + \zeta_{hashing}. \tag{3.16}$$

Let the model parameters be denoted as θ. According to Fig.3.1, θ can be divided into two parts: θ_c for the compression part, and θ_h for the hashing part. The learning rates of each part is denoted as η_c and η_h respectively. The model parameters are updated as following:

$$\begin{aligned}
\theta_c &= \theta_c - \eta_c(\nabla_{\theta_c}\zeta_{compression} + \nabla_{\theta_c}\zeta_{hashing}) \\
&= \theta_c - \eta_c(w_1\nabla_{\theta_c}\zeta_D + w_2\nabla_{\theta_c}\zeta_R + \nabla_{\theta_c}\zeta_{hashing}), \\
\theta_h &= \theta_h - \eta_h(\nabla_{\theta_h}\zeta_{compression} + \nabla_{\theta_h}\zeta_{hashing}) \\
&= \theta_h - \eta_h\nabla_{\theta_h}\zeta_{hashing},
\end{aligned} \tag{3.17}$$

where w_1, w_2 are obtained from Eq.3.14. $\nabla_{\theta_h}\zeta_{compression} = \mathbf{0}$, as the back-propagation of the compression loss does not pass through the hashing part. $\nabla_{\theta_c}\zeta_{hashing}$ and $\nabla_{\theta_h}\zeta_{hashing}$ can be combined to $\nabla_\theta\zeta_{hashing}$, and updated by PCGrad Update Rule in Table 2.1.

In order to get competitive hashing performance compared to single-task hashing baseline, a very small value (i.e. 1e-7) is set to η_c, so that the model parameters of the compression part can be fine-tuned by the hashing part without changing the compression performance.

4 Dataset and Experiments

4.1 Dataset

The experiments are conducted on patches from BigEarthNet-S2 [58], which is a multi-label large-scale RS archive released by Sümbül et al. in 2019. The data source of BigEarthNet-S2 come from Sentinel-2 images, which consist of 13 bands with spectrum ranged from visible (380nm-700nm), near-infrared (700 nm - 1100nm) to short-wave infrared (1100nm-3000nm) light. Fig.4.1 illustrates 13 bands and their corresponding ground resolution of Sentinel-2 images. The low-spatial resolution 60m bands (B01, B09, B10) are helpful to detect aerosol, water vapour and cirrus cloud. The 20m bands (B05, B06, B07, B8A) in the near-infrared range are designed for identifying vegetation, while the rest 20m bands (B11, B12) in the short-wave infrared spectrum are designed for snow, ice, and cloud detection. The high-spatial resolution 10m bands (B04, B03, B02) are classic RGB channels, the last 10m band B08 covers a broader spectrum than 20m band B08A, since B08A is less resistant against water vapor contamination in the spectral reflectance. To cover up all use-cases, both B08 and B08A are included in Sentinel-S2 images. Sümbül et al. [58] included 12 bands except B10 of these 13 bands in BigEarthNet-S2 patches, because B10 only provides information for cirrus clouds, which is not useful for image retrieval and classification on the earth surface.

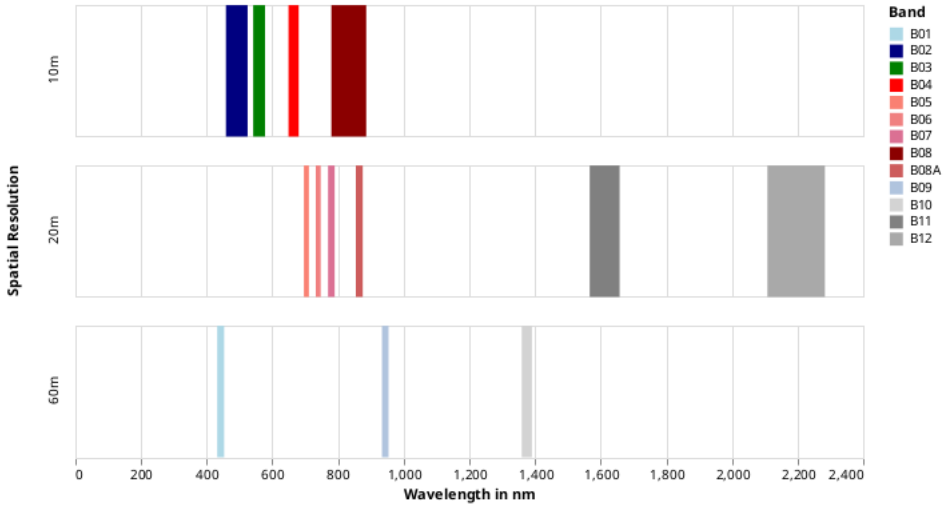
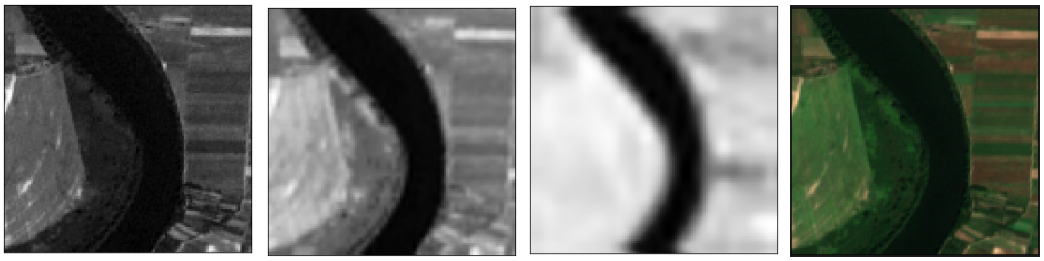


Figure 4.1: Spectral bands and corresponding ground resolutions of Sentinel-2 images

BigEarthNet-S2 consists of 590,326 pairs of Sentinel-2 image patches across over 10 different European countries (Austria, Belgium, Finland, Ireland, Kosovo, Lithuania, Luxembourg, Por-

tugal, Serbia, Switzerland). After removing 71,042 patches which are covered by seasonal snow and clouds, 519,284 patches are left for image retrieval and classification. Each patch covers a region of 1200m x 1200m, and has 12 bands (B10 is excluded) with ground resolutions of 10m, 20m, and 60m, thus, there are three width/height dimensions (120px, 60px and 20px) for each patch. Fig.4.2 shows an example of individual bands which have been interpolated to the same width/height, as well as the RGB colored image of the same area.



(a) B02 (10m band) (b) B05 (20m band) (c) B09 (60m band) (d) Ture color image (B04, B03, B02)

Figure 4.2: Example patch visualization

BigEarthNet-S2 has two types of labels, the original 43 class-nomenclature associated by class labels from CORINE Land Cover (CLC) database of the year 2018 (CLC 2018), and the recommended 19 class-nomenclature [59] which drops 11 original labels since they are hard to be accurately described by only considering BigEarthNet-S2 images. For the following experiments, the recommended 19 class-nomenclature is used.

In order to reduce the training time, the following experiments are conducted on 14832 patches selected from BigEarthNet Serbia Summer area. The selected patches are split to the train set (7761), the validation set (3508), and the test set (3563).

For data preprocessing, cubic interpolation is applied to 20m bands and 60m bands, so that all bands in each patch have the same width/height dimensions (120x120). Then each patch (12x120x120) is scaled down to (0, 1] by dividing the maximum pixel value (20566) of the whole BigEarthNet-S2.

4.2 Experimental Setups for the Proposed Framework

The Adam optimizer is applied for updating the network during the training. Batch size is 32 for all the experiments. The training of the proposed framework is divided into two stages: (1) train the compression part to a range of bit-rates; (2) train the hashing part jointly with the pre-trained compression part.

4.2.1 Experimental Setup for the Compression Part

At the first stage, the CNN compression part of the proposed multi-task framework is trained to get desired RD trade-off points. In order to get a range of bit-rates, the training of the first stage is divided to two steps. Step 1 is to optimize the distortion loss only, the learning rate is changed according to the averaged PSNR on the validation set.

$$\eta_{step1} = \begin{cases} 1e-4, & PSNR(\mathbf{x},\hat{\mathbf{x}}) < 45 \\ 5e-5, & 45 \leq PSNR(\mathbf{x},\hat{\mathbf{x}}) \leq 50 \\ 1e-5, & PSNR(\mathbf{x},\hat{\mathbf{x}}) > 50 \end{cases} \tag{4.1}$$

Step 2 starts when the distortion loss is close to its convergence point. Step 2 optimizes both distortion loss and bit-rate loss. The learning rate is set to 1e-5 at the beginning. After the bit-rate loss reaches its first plateau (around 1.5 bit per pixel (bpp) on current validation set), gradually increase the learning rate from 1e-5 to 9e-5 to obtain more lower bit-rates.

Ablation Study on the Compression Part

The proposed framework employs a CNN-based compression method in the compression part. In order to prove that CNN-based method is superior to other compression methods, a RNN-based compression method and JPEG 2000 are investigated for comparison in terms of rate-distortion performance, compression and decompression time.

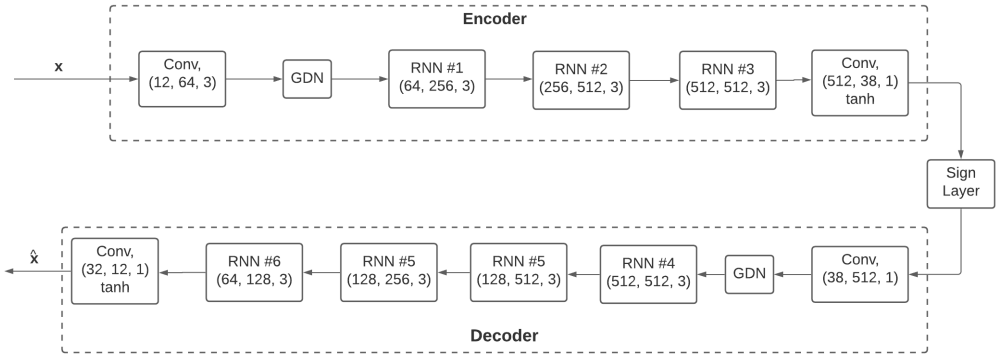


Figure 4.3: The architecture of RNN compression network

Fig 4.3 shows the architecture of the recent RNN compression network adapted from [23]. The encoder is made of a 3x3 convolution layer with stride 2 and padding 1, a GDN [3] layer, three RNN cells and another 1x1 convolution layer. The decoder is made of a 1x1 convolution

layer, an inverse GDN layer, four up-sampling RNN cells and a 1x1 convolution layer in the end. The latents are binarized to bitstream valued in {-1,1} by the *Sign Layer*. Straight through estimation (STE) [51] is applied in the *Sign Layer* for the back-propagation of the gradients from the binarized latents. The decoder reconstructs the image from binarized latents. Compared to the first RNN compression network proposed by Toderici et al.[63], [23] adds GDN layers to the encoder and the decoder.

The compression is done by progressively encoding the iterative residues of the input image. In order to finally reconstruct the input image, bits from all the iterations are decoded sequentially to recover the residue at each iteration. Then these decoded bits are added together to form the reconstructed image. The loss function takes the skip-connection scheme, in which the loss function encourages the total sum of the outputs from all iterations to be close to the input image. In this case the whole RNN network acts like a residual network. The whole process can be formulated as:

$$\begin{aligned}
\mathbf{r}^{(0)} &= \mathbf{x}^{(0)}, \\
\mathbf{b}^{(0)} &= SignLayer(Encoder(\mathbf{r}^{(0)})), \\
\hat{\mathbf{r}}^{(0)} &= Decoder(\mathbf{b}^{(0)}), \\
\mathbf{r}^{(1)} &= r_0 - \hat{\mathbf{r}}^{(0)}, \\
\mathbf{b}^{(1)} &= SignLayer(Encoder(\mathbf{r}^{(1)})), \\
\hat{\mathbf{r}}^{(1)} &= Decoder(\mathbf{b}^{(1)}), \\
&... \\
\mathbf{r}^{(t-1)} &= \mathbf{r}^{(t-2)} - \hat{\mathbf{r}}^{(t-2)}, \\
\mathbf{b}^{(t-1)} &= SignLayer(Encoder(\mathbf{r}^{(t-1)})), \\
\hat{\mathbf{r}}^{(t-1)} &= Decoder(\mathbf{b}^{(t-1)}), \\
bitstream &= np.stack(\mathbf{b}^{(0)}, \mathbf{b}^{(1)}, ..., \mathbf{b}^{(t-1)}), \\
\mathbf{x} = \mathbf{x}^{(0)}, \quad \hat{\mathbf{x}} &= \sum_{t=1}^{T} \hat{\mathbf{r}}^{(t-1)}, \\
\zeta_{compression} &= 1 - SSIM(\mathbf{x}, \hat{\mathbf{x}}),
\end{aligned} \tag{4.2}$$

where T is the number of iteration, $t \in \{1,2,3,...T\}$, $\mathbf{x}^{(0)}$ is the initial input data, $\mathbf{r}^{(0)}$ is the initial residual, $\hat{\mathbf{r}}^{(0)}$ is the recovered output at the initial iteration, $\mathbf{r}^{(1)}$ is the residual at the first iteration. $\mathbf{b}^{(0)}$ is the compressed binary code at iteration 1.

A variable range of bit-rates is achieved by changing the total iterations T. For example, suppose the size of the input data is (C, H, W), C stands for channels, H stands for heights, W stands for widths. The size of the latents from the encoder will be (38, H/16, W/16), after T iterations, the size of bitstream (binarized latents) is $T*38*H/16*W/16$, and the bit-rate for T iterations is $\frac{T*38*H/16*W/16}{H*W} = \frac{38*T}{16*16}$.

We set batch size to 32, use Adam optimizer, and train the RNN compression method for 280 epochs with iterations 3, 4, 5, 6 and 7 with the learning rate in Eq.4.1. Since the bit-rate is determined by the number of iterations, we first train the model of iterations 4 to its convergence. Then we train the model of other iterations based on the pre-trained model of the

adjacent iterations, i.e. the model of iterations 3 is trained based on that of iterations 4.

Evaluation Metrics for Compression

To evaluate the compression performance on multi-spectral RS images, the metrics of bit-rate, peak signal-to-noise ratio (PSNR) and spectral angle (SA) are applied.

Bit-rate is the number of bits required to represent one pixel in image compression. The unit of bit-rate is bits per pixel (bpp). Lower bit-rate indicates better compression ratio. Suppose the input image has 12 channels with width 100 and height 100, and it is compressed to 1440 bits, then the bit-rate for compression is $\frac{1440}{100*100} = 0.144$ bpp.

PSNR is to measure the quality of reconstruction of lossy image compression. The unit of PSNR is decibel (dB). A higher PSNR indicates better quality of the recovered image. It is mostly defined via Mean Square Error (MSE). Suppose the input image and its reconstruction are denoted as $\mathbf{x}$ and $\hat{\mathbf{x}}$. The width, height and channels of the input image are W, H and C, the maximum pixel value of the input image is MAX_x. PSNR is formulated as:

$$\begin{aligned} MSE &= \frac{1}{W*H*C}\sum_{i=1}^{W}\sum_{j=1}^{H}\sum_{k=1}^{C}[\mathbf{x}(i,j,k)-\hat{\mathbf{x}}(i,j,k)]_2^2, \\ PSNR(\mathbf{x},\hat{\mathbf{x}}) &= 20*log_{10}\frac{MAX_x}{\sqrt{MSE}}, \end{aligned} \tag{4.3}$$

where (i, j, k) is the location index of the pixel in the input image $\mathbf{x}$, $i \in \{1,2,...W\}$, $j \in \{1,2,...H\}$, $k \in \{1,2,...C\}$.

SA measures the spectral similarity between two multi-spectral images. It is the averaged angle between spectra vectors of these two multi-spectral images, note that a spectra is a pixel vector in a space with dimensions equal to the number of bands. The unit of SA is rad (radian). The value of SA is ranged between 0 and $\frac{\pi}{2}$. The closer the SA is to zero, the more similar the two multi-spectral images are. SA is formulated as:

$$SA = \frac{1}{W*H}\sum_{i=1}^{W}\sum_{j=1}^{H} cos^{-1}\Big(\frac{\sum_{k=1}^{C}\big(\mathbf{x}(i,j,k)*\hat{\mathbf{x}}(i,j,k)\big)}{\sqrt{\sum_{k=1}^{C}\mathbf{x}^2(i,j,k)\sum_{k=1}^{C}\hat{\mathbf{x}}^2(i,j,k)}}\Big). \tag{4.4}$$

4.2.2 Experimental Setup for the Hashing Part

At the second stage, the hashing part of the proposed multi-task framework is jointly trained with the pre-trained compression part with bit-rate around 0.6 bpp on the test data. The learning rates of hashing part and compression part is set to 1e-4 and 1e-7 respectively. These two parts are jointly trained for 40 epochs with hash bits at 16, 32 and 64.

Ablation Study on the Hashing Part

The hashing part in the proposed framework selects Greed Hash [57] as the activation function for hash coding, and PCGrad [70] as the optimization algorithm of the hashing loss. To validate this selection, a hashing baseline is designed to test different activation functions in the *Hash Layer*. Also, another optimization algorithm DWA [43] is applied to the hashing loss for comparison with PCGrad in terms of the retrieval performance.

Fig.4.4 shows the overall architecture of the hashing baseline. The *Encoder* and *Decoder* blocks and the hashing part have the same network structures as those in the proposed framework in Fig.3.1.

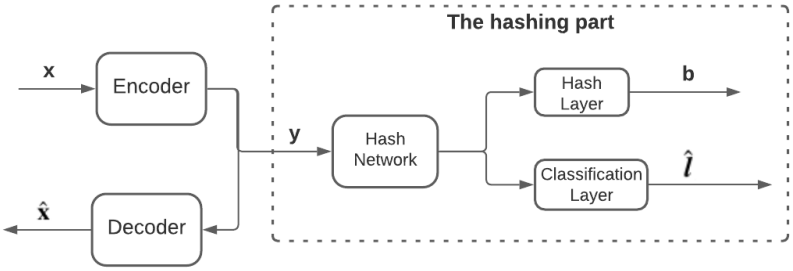


Figure 4.4: Overall architecture of hashing baseline

There are five choices for the activation function of the *Hash Layer*. Table 4.1 presents the forward pass, backward pass and related loss functions for each choice of the *Hash Layer*. In Table 4.1, with N as the number of images in current training batch, $\boldsymbol{U} \in \mathbb{R}^{Nxq}$ are continuous feature representations in the last convolution layer of *Hash Layer*, $\boldsymbol{B} \in \{-1,1\}^{Nxq}$ are encoded compact binary codes, $\hat{\boldsymbol{B}} \in (-1,1)^{Nxq}$ are the continuous approximation of $\boldsymbol{B}$, and γ is a hyper-parameter.

Activation function	Forward pass	Backward pass	Quantization loss	Balance loss
Sigmoid	$\hat{\boldsymbol{B}} = \frac{2}{1+e^{-\boldsymbol{U}}} - 1$	$\frac{\partial \zeta}{\partial \hat{\boldsymbol{B}}}$	✓	✓
Tanh	$\hat{\boldsymbol{B}} = \frac{2}{1+e^{-2\boldsymbol{U}}} - 1$	$\frac{\partial \zeta}{\partial \hat{\boldsymbol{B}}}$	✓	✓
SoftSign	$\hat{\boldsymbol{B}} = \frac{\boldsymbol{U}}{1+\lvert\boldsymbol{U}\rvert}$	$\frac{\partial \zeta}{\partial \hat{\boldsymbol{B}}}$	✓	✓
Greedy Hash [57]	$\boldsymbol{B} = sign(\boldsymbol{U})$	$\frac{\partial \zeta}{\partial \boldsymbol{B}} + \gamma(\boldsymbol{U} - \boldsymbol{B})$		✓
Bi-half [40]	$\boldsymbol{B} = sign(\boldsymbol{U} - median(\boldsymbol{U}))$	$\frac{\partial \zeta}{\partial \boldsymbol{B}} + \gamma(\boldsymbol{U} - \boldsymbol{B})$		

Table 4.1: Choices of activation functions for hash coding

The hashing loss function is different when choosing different activation function for the *Hash Layer*. Sigmoid, Tanh and SoftSign require a quantization loss to narrow the gap between approximated codes and discrete codes, which is formulated as:

$$\zeta_{quantization} = \sum_{(i,j)\in\xi} \left(\|sign(\hat{\mathbf{b}}^{(i)}) - \hat{\mathbf{b}}^{(i)}\|_1 + \|sign(\hat{\mathbf{b}}^{(j)}) - \hat{\mathbf{b}}^{(j)}\|_1 \right) \tag{4.5}$$

GreedyHash [57] and Bi-half [40] don't need quantization loss, since they generate discrete codes directly, and use the straight through estimation method (STE) [8] for back-propagation. Bi-half doesn't need the balance loss, by ranking U column-wisely and assigning the top half of the elements to +1, and the remaining half to -1, the discrete hash codes from Bi-half activation are already half-half distribution.

$$\zeta_{hashing} = \begin{cases} w_1\zeta_{pairwise} + w_2\zeta_{balance} + w_3\zeta_{quantization} & Sigmoid\ or\ Tanh\ or\ SoftSign \\ w_1\zeta_{pairwise} + w_2\zeta_{balance} & GreedyHash \\ w_1\zeta_{pairwise} & Bi-half \end{cases} \tag{4.6}$$

where w_1, w_2, w_3 are weights for each individual loss, $\zeta_{pairwise}$ is presented in Eq.3.7, $\zeta_{balance}$ is presented in Eq.3.8.

Since the backbone of hashing baseline is an autoencoder, a distortion loss (Eq.3.4), which measures the quality of reconstructed image, should be added to the loss of the hashing baseline. Also, it is useful to add a classification loss (Eq.3.9), which narrows the gap between the ground truth labels and predicted labels from the classification layer. Thus, the total loss of hashing baseline is:

$$\zeta_{hashing\_baseline} = \zeta_{hashing} + w_4\zeta_D + w_5\zeta_{classification} \tag{4.7}$$

where w_4, w_5 are weights for corresponding individual loss.

We set the hash bits to 32, the learning rate to 1e-4, and train the hashing baseline of different activation functions for 40 epochs. DWA and PCGrad [70], two commonly used MTL optimization algorithms in classification and regression tasks, are applied to optimize $\zeta_{hash\_baseline}$.

Evaluation Metrics for Retrieval

The metrics to evaluate the hashing retrieval performance are: (1) average precision; (2) average recall.

Precision is the percentage of correctly predicted labels in all predicted labels. Average precision is calculated by averaging the precision of multiple queries. Let the label set for one query image as $\mathbb{L}_q$, the label set for one retrieved image as $\mathbb{L}_r$. $|.|$ denotes the number of elements in the set. Suppose there are Q query images, and the number of retrieved images for each query image is R. Average precision is formulated as:

$$Average\ precision = \frac{1}{Q*R}\sum_{q=1}^{Q}\sum_{r=1}^{R}\frac{|\mathbb{L}_r \cap \mathbb{L}_q|}{|\mathbb{L}_r|}. \tag{4.8}$$

Recall is the percentage of correctly predicted labels in all the ground truth labels. Average recall is calculated by averaging the recall of multiple queries. Average recall is formulated as:

$$Average\ recall = \frac{1}{Q*R}\sum_{q=1}^{Q}\sum_{r=1}^{R}\frac{|\mathbb{L}_r \cap \mathbb{L}_q|}{|\mathbb{L}_q|}. \tag{4.9}$$

In the hashing experiments, the number of retrieved images is fixed to 8 (i.e, R=8), the number of query images is the size of the validation set (i.e, Q= 3508), average precision and average recall are evaluated by performing 3508 queries from the gallery (the test set).

4.2.3 Performance Analysis of the Hashing Part

Since the hashing part is trained on the image features which were trained by the compression part, it is necessary to analyse how well the hashing part performs with different settings during the joint training. Table 4.2 summarizes the default settings of the joint training at the second stage.

Table 4.2: Default setting of joint training of the proposed framework

Modules	Description	losses	Learning rate	Optimization
Compression part	pre-trained CNN compression part at bit-rate 0.63bpp	ζ_D, ζ_R	1e-7	MGDA
Hashing part with GreedyHash for hash coding	attention module in the hashing part	$\zeta_{pairwise}$, $\zeta_{balance}$, $\zeta_{classification}$	1e-4	PCGrad

Analysis of the learning rate of compression part

In order to prove that the learning rate of compression part impacts the hashing performance, we change the learning rate of the compression part $\eta_{compression}$ as follows, and keep other settings unchanged in Table 4.2. Then jointly train the compression and hashing framework for 20 epochs, and compare the retrieval performance at hash bits 64.

- $\eta_{compression}$ =0
- very small $\eta_{compression}$ (i.e, 1e-7),
- $\eta_{compression} = \eta_{hashing}$ = 1e-4

Analysis of the attention module in the hashing part

In order to prove that the attention module is helpful for the hashing performance, we remove the attention module from the hashing part, and keep other settings in Table 4.2 unchanged, then train the CNN framework for 20 epochs, and compare the retrieval performance at hash bits 64 to that with attention module.

Analysis of different pre-trained compression parts

In order to check if the hashing performance changes with different compression parts, we load CNN/RNN pre-trained compression models at different bit-rates (i.e,0.5 bpp to 0.8 bpp) to the compression part, and keep other settings in Table 4.2 unchanged. Then train the framework for 40 epochs, and check the retrieval performance at hash bits 64.

Analysis of different locations of the hashing part

In order to check if the hashing performance changes with the location of the hashing part, we move the hashing part to the decoder side as shown in Fig.4.5, and we load pre-trained compression parts at different bit-rates (i.e, 0.5 bpp to 0.8 bpp), and keep other settings in Table 4.2 unchanged. Then train the framework for 40 epochs, and check the retrieval performance at hash bits 16, 32 and 64.

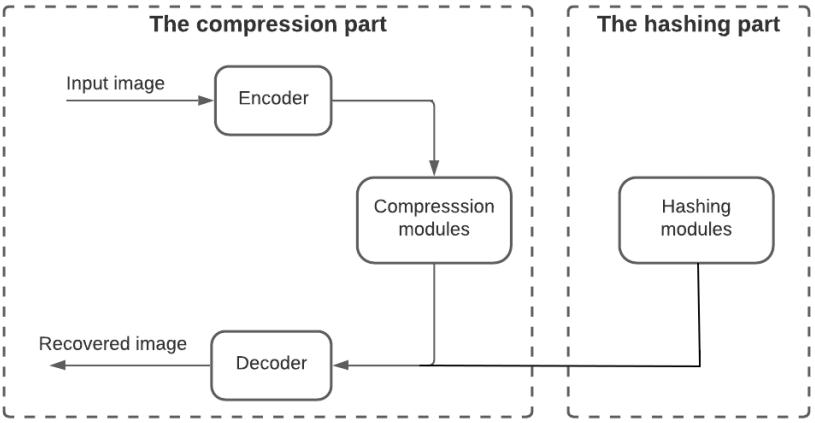


Figure 4.5: General diagram of proposed multi-task framework with the hashing part at the decoder side

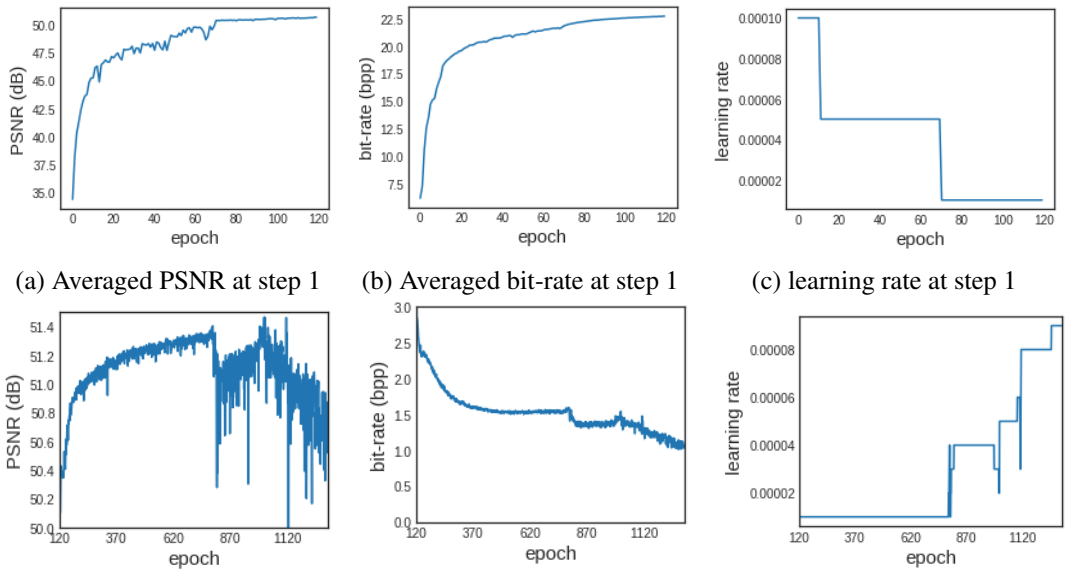
5 Experimental Results and Discussion

All the experimental results were obtained by training on NVIDIA Tesla V100 GPUs with 32 GBs of memory. The code of the compression part was built upon CompressionAI [7].

The compression metrics (bit-rate/PSNR/SA) on the validation set were averaged estimated values generated by testing the trained model on the validation set during the training. Note the compression metrics such as bit-rates on the validation set were calculated from estimated entropy values, as AE and AD were not involved during the training. The compression metrics on the test set were averaged actual values, i.e, the bit-rates were actual bits counted on produced bitstream. The retrieval metrics (precision/recall) were averaged values generated by retrieving 8 most similar samples from the test set (3563) for each patch in the validation set (3508).

5.1 Training Results of the Proposed Framework

5.1.1 Training Results of the Compression Part



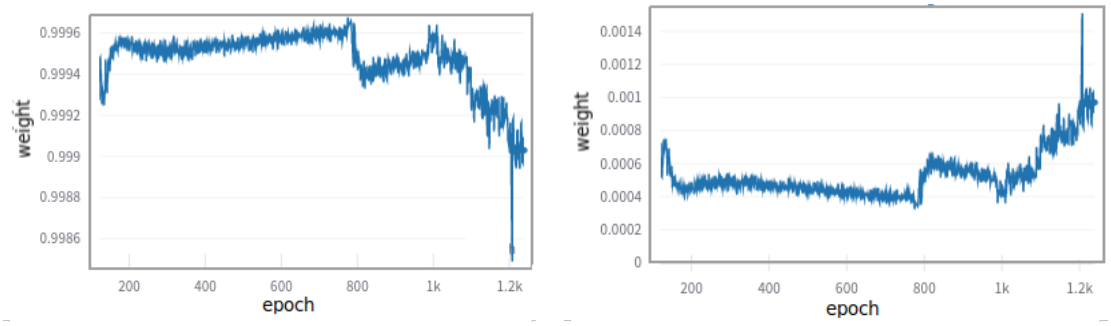
(a) Averaged PSNR at step 1 (b) Averaged bit-rate at step 1 (c) learning rate at step 1

(d) Averaged PSNR at step 2 (e) Averaged bit-rate at step 2 (f) learning rate at step 2

Figure 5.1: Training results of the CNN compression part

Fig.5.1 shows the averaged PSNR and bit-rate on the validation set during the training of the compression part. The first step took about 120 epochs for the distortion loss to converge. Since

the first stage only trained the distortion loss and left the bit-rate loss untrained, the averaged PSNR reached to its highest value around 51db at epoch 120, while the averaged bit-rate increased to its highest value 23 bpp. Then at the second step (after 120 epochs), the distortion loss and the bit-rate loss were trained jointly and optimized by MGDA [16]. The averaged PSNR kept improving while the averaged bit-rate dropped very quickly and reach a plateau at around 1.5 bpp. The averaged bit-rate continued to decrease to lower values when increasing the learning rate.



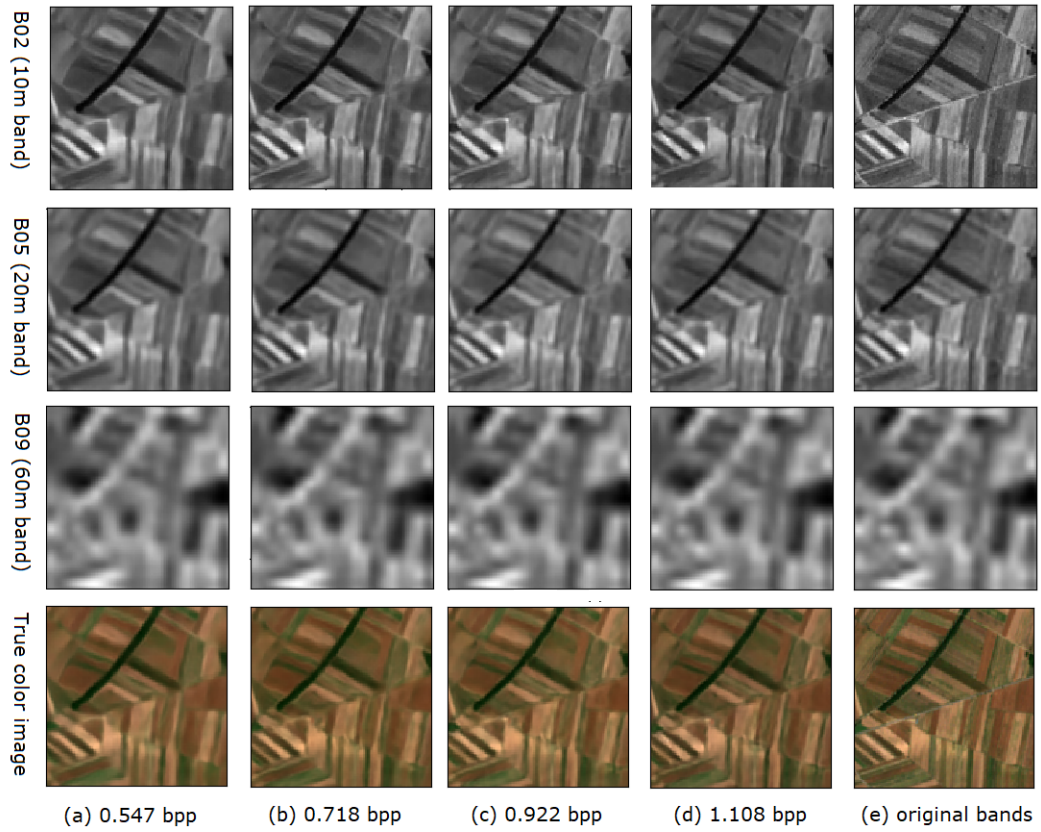
(a) Gradient weights of distortion loss

(b) Gradient weights of bit-rate loss

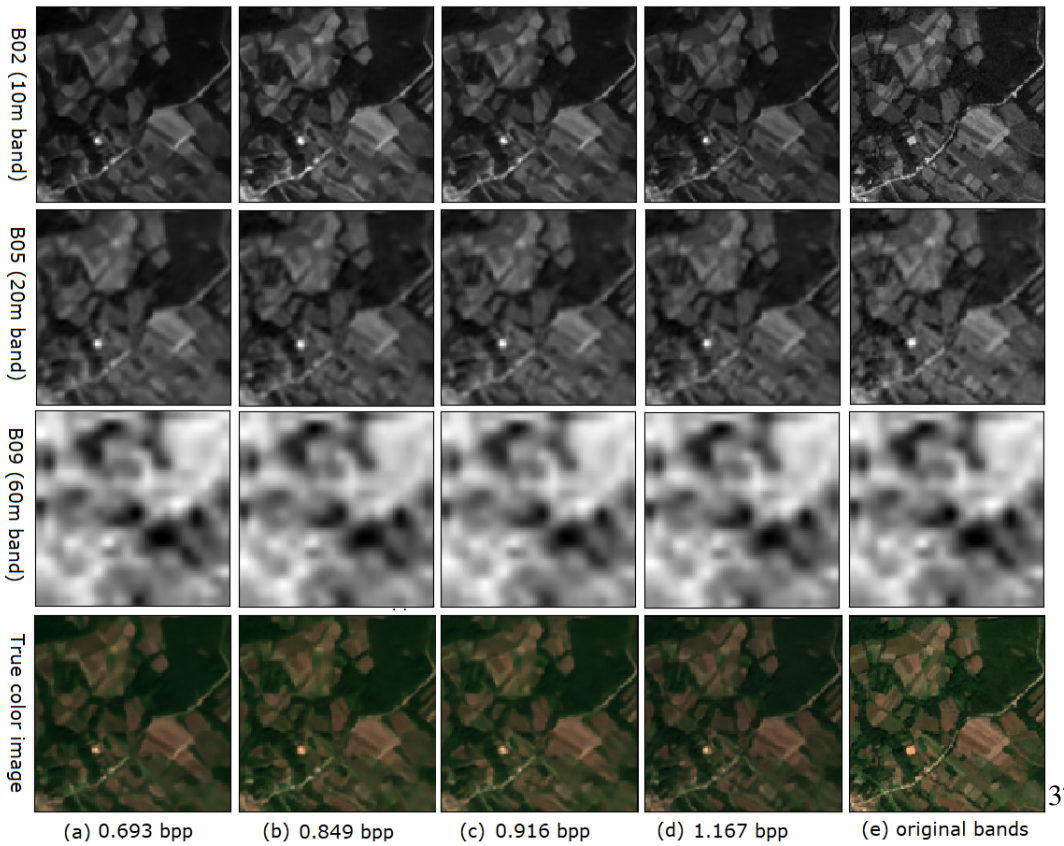
Figure 5.2: Gradient weights generated by MGDA during the training of the CNN compression part

Fig.5.2 shows the gradient weights of the distortion loss and the bit-rate loss during the training of the second step. The gradient weights of the distortion loss were way much higher than that of the bit-rate loss. It dropped a little bit around epoch 820 when the learning rate was increased from 1e-5 to 4e-5, which resulted in a very slow degradation on averaged PSNR and quick decrease on averaged bit-rate in Fig.5.1(d-e). Increasing the learning rate would increase the gradient weight for bit-rate loss, which resulted in lower bit-rates. The range of averaged bit-rates obtained on the test set was between 0.6 bpp to 1.5 bpp by training the network on current training data. Note that the learning rate should be kept at a small value (less than 1e-4), otherwise the averaged PSNR would degrade very suddenly, which resulted in unrecognizable reconstructed images.

For illustrative purposes, Fig.5.3 shows the reconstructed individual bands and colored image of two test patches at different bit-rates by the trained compression part.



(a) Test patch 1

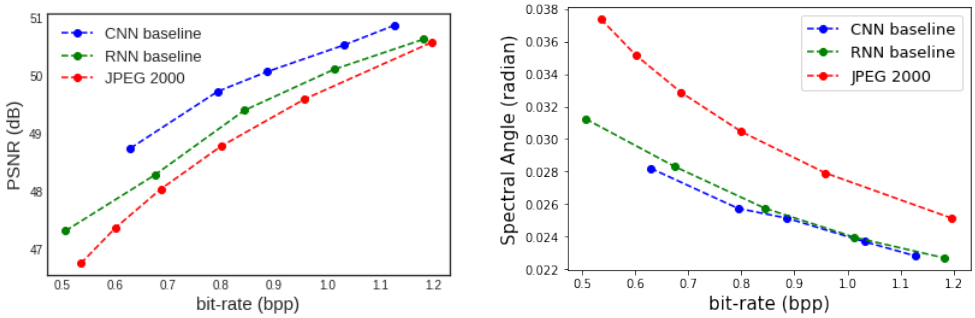


(b) Test patch 2

Figure 5.3: Reconstructed multi-spectral image at different bit-rates by the CNN compression part

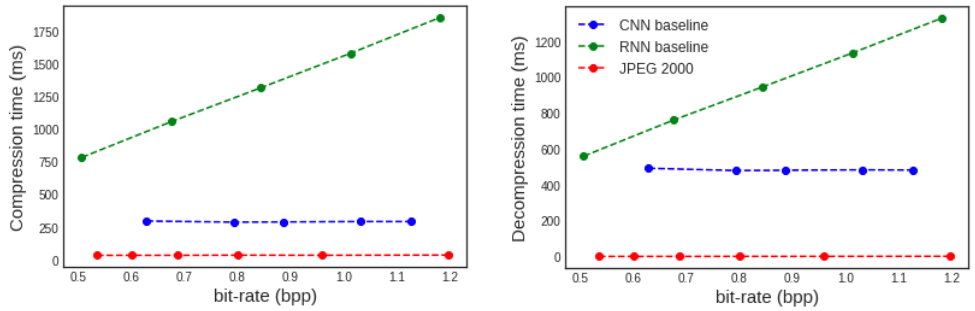
Ablation Study Results on the Compression Part

This ablation study is carried out in order to compare CNN-based compression method against RNN-based compression method and JPEG 2000 in terms of the rate-distortion performance, the compression and decompression time.



(a) Averaged PSNR (b) Averaged spectral angle

Figure 5.4: Comparison of compression performance on BigEarthNet Serbia summer area



(a) Averaged compress time (b) Averaged decompress time

Figure 5.5: Comparison of compression and decompression time on BigEarthNet Serbia summer area

Fig.5.4 shows the comparison of compression performance of different methods. The compared metrics were PSNR, Spectral Angle and bit-rate averaged over 3563 patches of the test data. Both learned compression methods (CNN/RNN) performed better than JPEG 2000. Regarding PSNR, the CNN-based method performed better than the RNN-based method. However, when evaluating the spectral angle (SA), the CNN-based method got similar SA curve over different bit-rates as the RNN-based method did.

PSNR is an approximation to human perception of reconstruction quality, which more focuses on the recovery of the spatial information in the image, and SA is a measurement for spectral information recovery. Since the distortion loss in both CNN and RNN based compression methods were calculated by SSIM [65], which minimized the spatial structure difference between input image and reconstructed image, it is possible that the spectral information recovery was

not further optimized during the training. Also the 2D convolutions applied in the CNN compression baseline mainly focused on extracting structure information, which may be not able to extract the spectral information in the multi-spectral images.

Fig.5.5 shows the comparison of processing time (CPU) require by each method. The compression and decompression time were averaged over 3563 patches of the test data. The processing time of the CNN method is way more than JPEG 2000, but still significantly less than that of the RNN-based method which increased linearly with the bit-rates.

5.1.2 Training Results of the Hashing Part

Ablation Study Results on the Hashing Part

This ablation study is carried out to select proper hash coding function and optimization method for the hashing part. Table 5.1 shows the retrieval performance of CNN hashing baseline at hashbits 32 based on different settings. Regarding both precision and recall scores, Sigmoid, Tanh and Greedy Hash perform better than Softsign and Bi-half. Given different combinations of the activation functions and optimization methods, DWA works best with Sigmoid while PCGrad works well with Tanh and Greedy Hash. Since GreedyHash has got very close precision and recall scores compared to Sigmoid and Tanh, and it doesn't need quantization loss like Sigmoid or Tanh does as illustrated in Table 4.1, Greedy Hash is selected as the activation function in the hashing layer, and PCGrad is selected for the optimization of the hashing loss.

Table 5.1: Retrieval performance of CNN hashing baseline at hashbits 32 with different combinations of activation functions and optimization methods

Optimization	Activation function	Precision	Recall
DWA	**Sigmoid**	**0.73**	**0.70**
	Tanh	**0.71**	**0.71**
	Softsign	0.71	0.71
	Greedy Hash	**0.71**	**0.70**
	Bi-half	0.66	0.66
PCGrad	**Sigmoid**	**0.71**	**0.70**
	Tanh	**0.72**	**0.70**
	Softsign	0.71	0.65
	Greedy Hash	**0.72**	**0.70**
	Bi-half	0.68	0.68

Table 5.2: Retrieval performance of CNN hashing baseline when Greedy Hash is the activation function in the hashing layer and optimization method is PCGrad

Hash bits	Precision	Recall
16	0.72	0.69
32	0.72	0.70
64	0.75	0.71

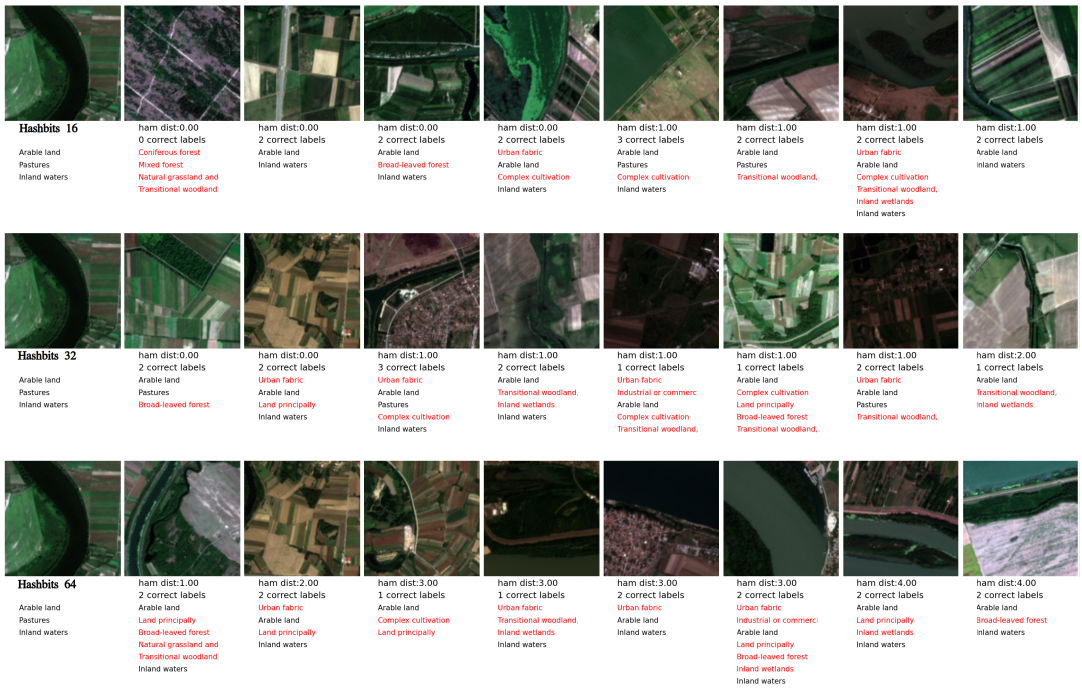


Figure 5.6: Comparison of retrieved images of hashing baseline at hashbits 16, 32 and 64

Fig.5.6 shows retrieved images at hashbits 16, 32 and 64. The first image in each row is the same query image selected from the validation set. The rest 8 images in each row are the retrieved images from the test set. They are ordered ascending according to the hamming distance from the query image, smaller hamming distance means higher similarity in hashing space. The text below each image contains the hamming distance (i.e, ham dist) and the number of correctly matched labels between the query and the retrieved image, as well as the ground truth labels of each image, the red color indicates unmatched labels.

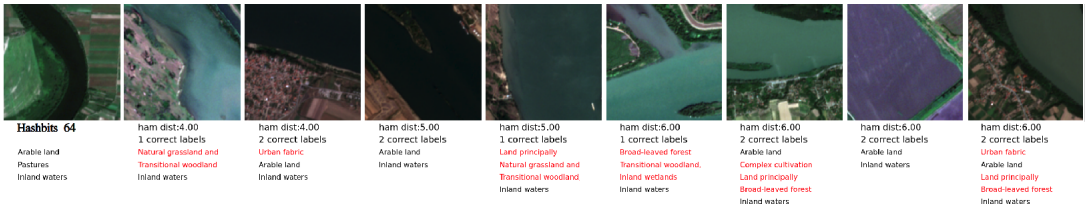
As shown in Fig.5.6, the query image contains three almost equally proportioned land types: pastures, inland waters and arable land. The expected most similar retrieved image should contain at least one of above land types. Obviously hashbits 16 is not long enough to rank different images in hashing space, since the first four retrieved images at hashbits 16 got 0 hamming distance to the query. Hashbits 32 and hashbits 64 have a better ranking of hamming distance than hashbits 16. Their retrieved images all contain at least one land type which exists in the query image. Hashbits 64 retrieved more visually similar images than hashbits 32 did. Since hashbits 16 is too short for meaningful retrieval in hashing space, we will check retrieval results at hashbits 32 and 64 in the following analysis.

Table 5.3 compares the retrieval performance of joint training and hashing baseline (Table 5.2). The precision and recall scores of joint training are very close to those of hashing baseline especially at hash bits 32, but they are slightly lower than those of the hashing baseline at hash-bits 64.

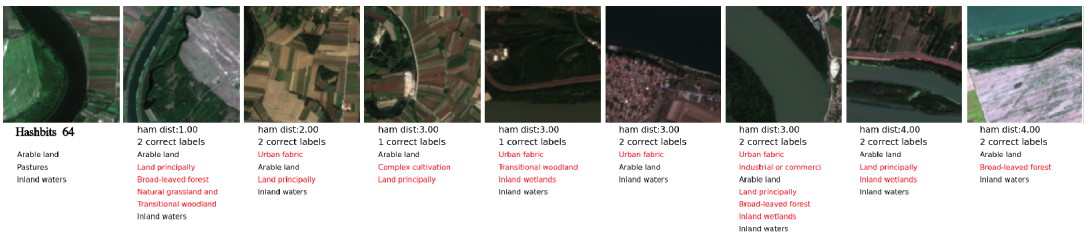
Table 5.3: Comparison of retrieval performance between joint training and hashing baseline

Hashbits	Precision		Recall	
	joint	baseline	joint	baseline
32	**0.72**	**0.72**	**0.70**	**0.70**
64	0.73	0.75	0.69	0.71

Fig.5.7 compares the retrieved images of the joint training and hashing baseline at hashbits 64. The query image is the same one used in Section 5.1.2, which contains three almost equally proportioned land types: pastures, inland waters and arable land. The retrieved images from joint training contained large areas of inland waters but very small portions of arable land, while those from hashing baseline focus both on arable land and inland waters, which leads to a higher precision/recall score than joint training.



(a) Hashbits 64, precision 0.73, recall 0.69, CNN compression part (0.63bpp, PSNR 48,7dB)



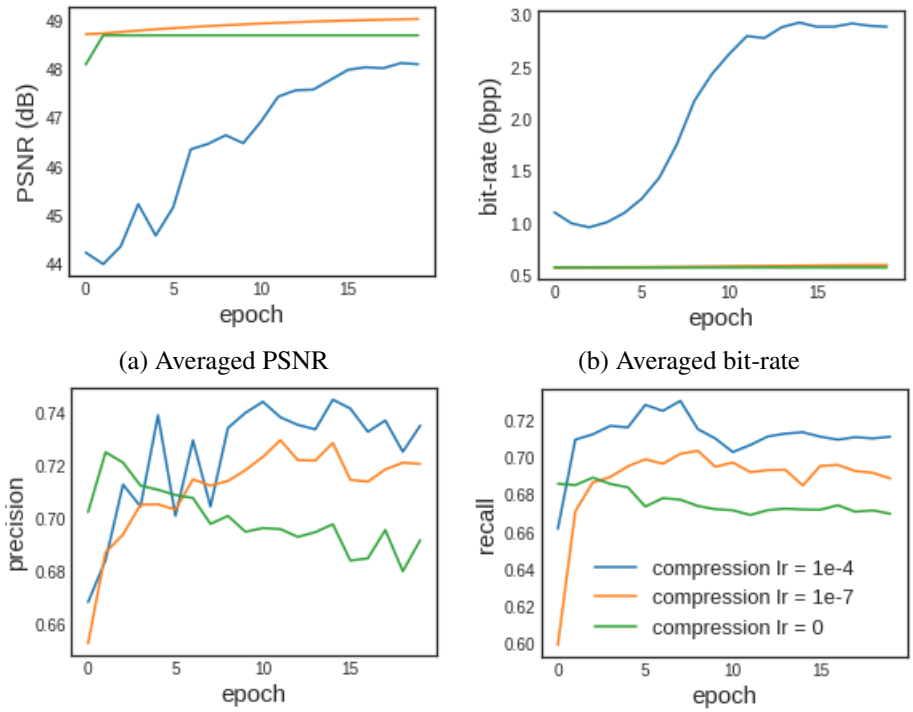
(b) Hashbits 64, precision 0.75, recall 0.71, CNN hashing baseline

Figure 5.7: Comparison of retrieved images between joint training and hashing baseline

5.1.3 Performance Analysis of the Hashing Part

Performance analysis on the learning rate of the compression part

Fig.5.8 shows the joint training results when the learning rate of the compression part was set to different values. PSNR and bit-rate are compression metrics, which averaged on the validation set. Precision and recall are hashing metrics at hashbits 64, which were calculated by retrieving 8 most similar images from the test set for each query image in the validation set.



(a) Averaged PSNR

(b) Averaged bit-rate

(c) Averaged precision (hashbits 64)

(d) Averaged recall (hashbits 64)

Figure 5.8: Joint training results with different learning rates of the compression part

- $\eta_{compression}$ =0 (Green line)
  The compression metrics kept stable. However, the hashing metrics were the lowest. The compression part was frozen when the learning rate was set to 0. This indicates that the hashing part can not learn well when it is not able to adjust the shared image features which extracted from the compression part.

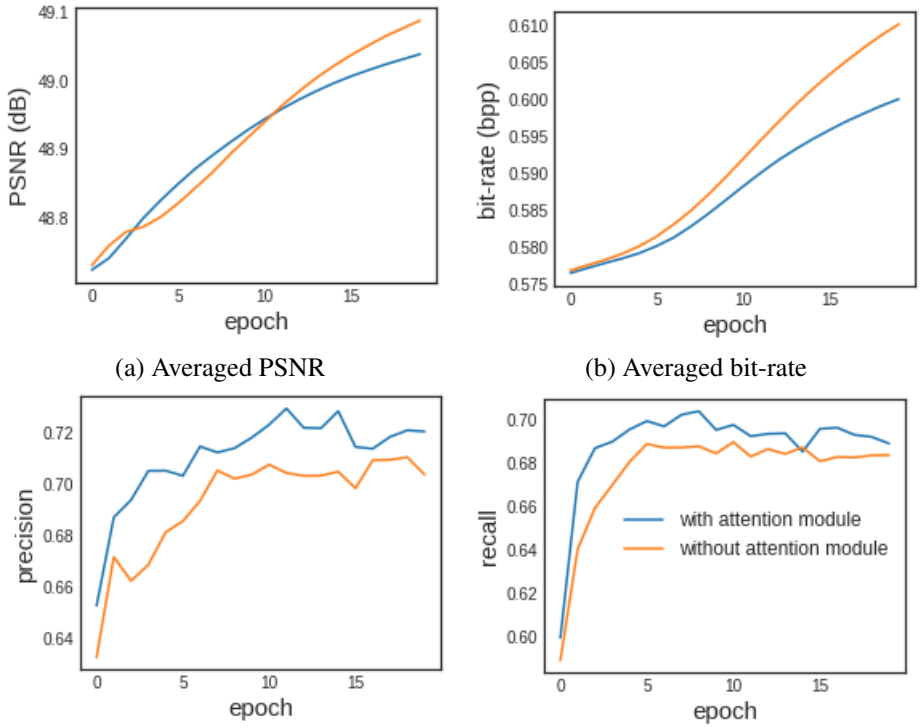
- $\eta_{compression} = \eta_{hashing} =$ 1e-4 (Blue line)
  The compression performance was degraded from the pre-trained rate-distortion point. However, the hashing metrics were the highest, which were very close to those of hashing baseline at hashbits 64 (precision 0.75, recall 0.71). Since a lower PSNR of the compression part corresponds to better retrieval performance in the hashing part, we can get the

conclusion that the hashing part does not require high reconstruction ability on the image features as the compression part does.

- $\eta_{compression}$=1e-7 (Orange line)
  The compression metrics varied little from the pre-trained rate-distortion point. The hashing metrics were higher than those when the compression part was frozen, and were close to those when the learning rate of compression part was 1e-4. We can get the conclusion that, a very small learning rate of the compression part, which allows the hashing part to adjust shared image features, can result in an acceptable compromise on the hashing performance with very little impact on the compression performance of the pre-trained compression part.

Performance analysis on the attention module in the hashing part

Fig.5.9 show joint training results when the attention module in the hashing part was kept or removed. The blue line represents joint training with attention module in the hashing part, and the orange line represents without attention module.



(a) Averaged PSNR

(b) Averaged bit-rate

(c) Averaged precision (hashbits 64)

(d) Averaged recall (hashbits 64)

Figure 5.9: Joint training results with or without the attention module in the hashing part

The compression performance was not changed that much by adding an attention module to the hashing part. The retrieval performance was improved comparing to that without the attention

module, though it was slightly lower than that from the hashing baseline as showed in Table 5.4.

Table 5.4: Comparison of retrieval performance
when joint training with or without attention module in hashing part

Hashbits 64	Precision	Recall
Hashing baseline	0.75	0.71
With attention module	0.73	0.69
Without attention module	0.71	0.68

Fig.5.10 compares retrieval results at hashbits 64 when joint training with and without attention module in the hashing part. The retrieved images in Fig.5.10(a) were structurally more similar to the query image than those in Fig.5.10(b), especially when comparing the retrieved images with the smallest hamming distances to the query image. From the retrieved results, it looks like the attention module can help the hashing part to learn more task-specific information which is beneficial for discriminating different images in the hashing space.

(a) With attention module, hashbits 64, precision 0.73, recall 0.69
CNN compression part (0.63bpp, PSNR 48.7dB)

(b) Without attention module, hashbits 64, precision 0.71, recall 0.68
CNN compression part (0.63bpp, PSNR 48.7dB)

Figure 5.10: Comparison of retrieved images when joint training with or without attention module in the hashing part

Performance analysis on different compression parts

Table 5.5 shows the retrieval performance when joint training with different compression parts. It is easy to find that the retrieval performance of joint training did not change that much when pre-trained CNN/RNN compression models at different bit-rates were loaded to the compression part.

Table 5.5: Comparison of retrieval performance when joint training with different compression parts

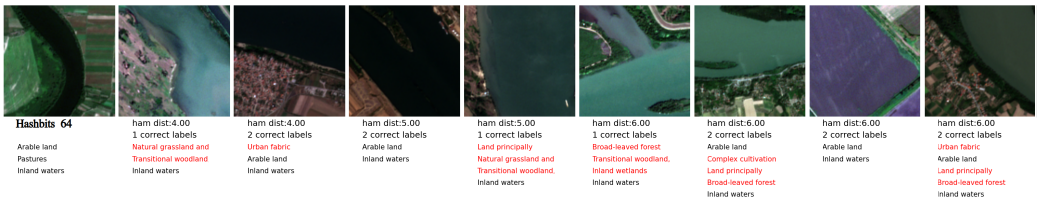
(a) CNN framework, hashbits 64

CNN compression part	Precision	Recall
Bpp 0.63, PSNR 48.7	0.73	0.69
Bpp 0.79, PSNR 49.7	0.73	0.68
Bpp 1.03, PSNR 50.5	0.73	0.68

(b) RNN framework, hashbits 64

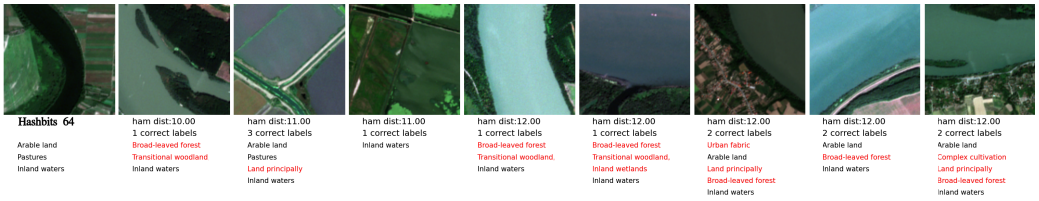
RNN compression part	Precision	Recall
Bpp 0.51, PSNR 47.3	0.71	0.70
Bpp 0.67, PSNR 48.5	0.73	0.70
Bpp 0.84, PSNR 49.4	0.71	0.69

Fig.5.11 compares retrieved images at hashbits 64 when joint training with different pre-trained compression parts. The comparison shows that the retrieved images of joint training did not change that much when the compression part were loaded at different bit-rates, though the hashing part trained with CNN compression parts retrieved more structurally similar images than that trained with RNN compression parts.



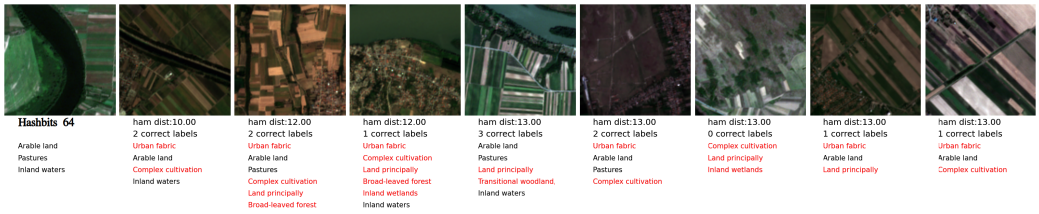
(a) CNN compression part (0.63bpp, PSNR 48.7dB), hashbits 64, precision 0.73, recall 0.69

(b) CNN compression part (0.79bpp, PSNR 49.7dB), hashbits 64, precision 0.73, recall 0.68

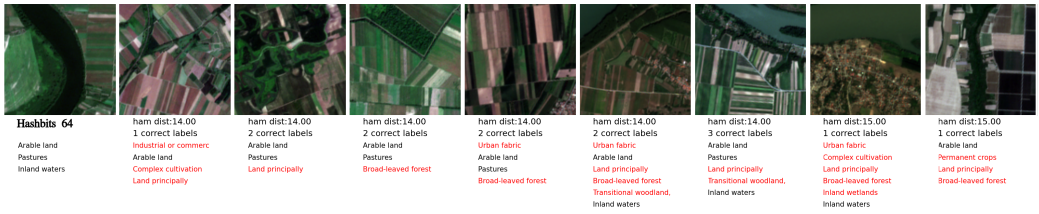


(c) CNN compression part (1.03bpp, PSNR 50.5dB), hashbits 64, precision 0.73, recall 0.68

(d) RNN compression part (0.51bpp, PSNR 47.3dB), hashbits 64, precision 0.71, recall 0.70



(e) RNN compression part (0.67bpp, PSNR 48.5dB), hashbits 64, precision 0.73, recall 0.70



(f) RNN compression part (0.84bpp, PSNR 49.4dB), hashbits 64, precision 0.71, recall 0.69

Figure 5.11: Comparison of retrieved images when joint training with different compression parts

Performance analysis on the location of the hashing part

In this performance study, first we fixed the compression part, put the hashing module at the decoder side, and checked the retrieval performance at different hashbits. Then we kept the hashbits at 64, and checked the retrieval performance when joint training with different pre-trained compression parts. Table 5.7 shows the retrieval performance when the hashing part was moved from the encoder part to the decoder part. There were very small changes in the precision/recall scores especially at hashbits 64, regardless of the network type, rate-distortion points of the pre-trained compression part or the location of the hashing part.

Table 5.7: Comparison of retrieval performance when joint training with hashing module at encoder or decoder side

(a) CNN compression part (Bpp 0.63, PSNR 48.7)

Hash bits	Precision		Recall	
	encoder	decoder	encoder	decoder
16	0.68	0.70	0.67	0.68
32	0.72	0.70	0.69	0.68
64	**0.73**	**0.72**	**0.69**	**0.69**

(b) RNN compression part (Bpp 0.67, PSNR 49)

Hashbits	Precision		Recall	
	encoder	decoder	encoder	decoder
16	0.68	0.65	0.67	0.65
32	0.72	0.69	0.70	0.69
64	**0.73**	**0.73**	**0.70**	**0.70**

(c) CNN compression part, hashbits 64

CNN compression part	Precision		Recall	
	encoder	decoder	encoder	decoder
Bpp 0.63, PSNR 48.7	**0.73**	**0.72**	**0.69**	**0.69**
Bpp 0.79, PSNR 49.7	**0.73**	**0.73**	**0.69**	**0.69**
Bpp 1.03, PSNR 50.5	**0.73**	**0.72**	**0.68**	**0.68**

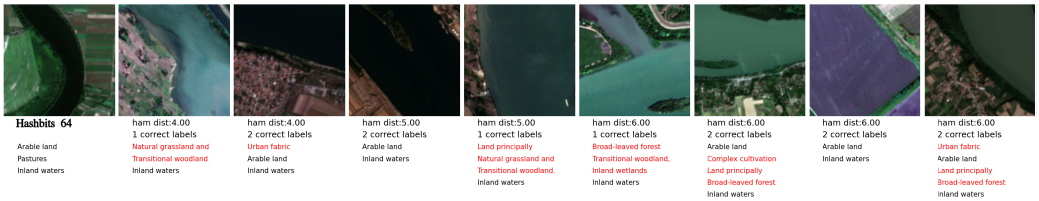
(d) RNN compression part, hashbits 64

RNN compression part	Precision		Recall	
	encoder	decoder	encoder	decoder
Bpp 0.51, PSNR 47.3	0.71	0.71	0.70	0.70
Bpp 0.67, PSNR 48.5	0.73	0.73	0.70	0.70
Bpp 0.84, PSNR 49.4	0.71	0.73	0.69	0.70

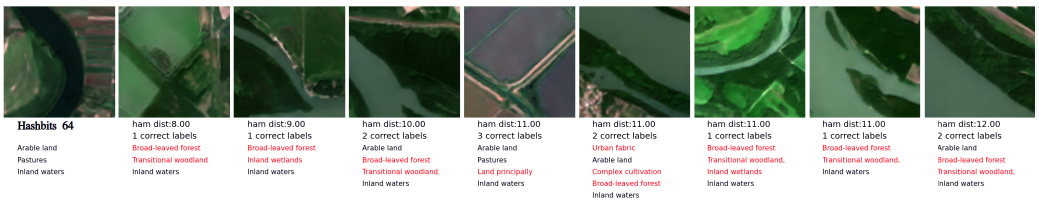
When the hashing part is at the encoder side, during the training and inference, hash codes are trained and generated from continuous latents which are extracted from the image data. When the hashing part is at the decoder side, during the training, it's trained on approximated quantized latents which extracted from the image data, during the inference, hash codes are generated from latents recovered from the bitstream. In other words, the retrieval is carried out in the image domain when the hashing part is at the encoder side, and it is carried out in the bitstream domain when the hashing part is at the decoder side.

Fig.5.12 shows the comparison of retrieved images in the image domain and in the bitstream domain. The results show that retrieved images were similar no matter the retrieval was done in the image domain or the bitstream domain, which indicates that the quantization of image features does not impact the hashing performance.

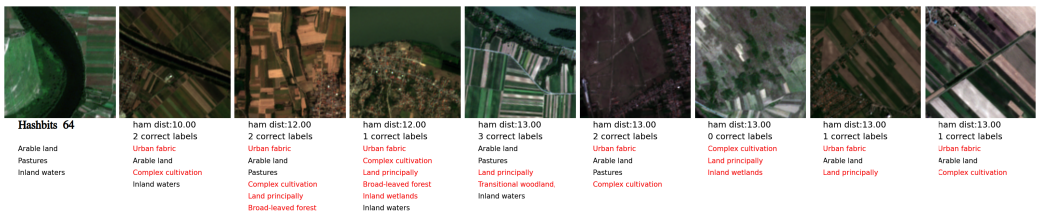
Fig.5.13 shows the comparison of retrieved images in the bitstream domain when joint training with different compression parts. The results show that retrieved images from bitstreams were quite similar when the compression part was loaded with pre-trained CNN/RNN parameters at different bit-rates, though the hashing part trained with CNN compression parts retrieved more structurally similar images than that trained with RNN compression parts.



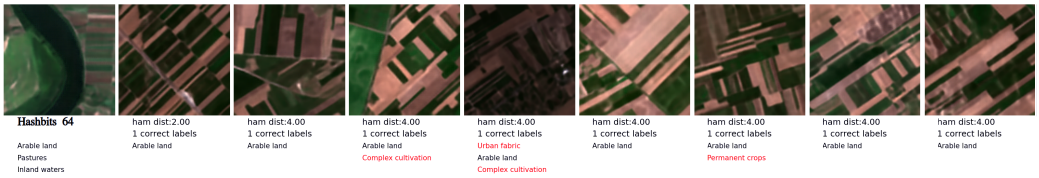
(a) Retrieved from images, hashbits 64, precision 0.73, recall 0.69
CNN compression part (0.63bpp, PSNR 48.7dB)



(b) Retrieved from bitstream, hashbits 64, precision 0.72, recall 0.69
CNN compression part (0.63bpp, PSNR 48.7dB)

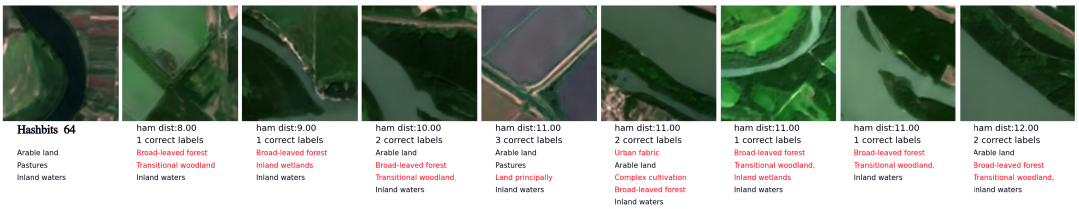


(c) Retrieved from images, hashbits 64, precision 0.73, recall 0.70
RNN compression part (0.67bpp, PSNR 48.5dB)



(d) Retrieved from bitstream, hashbits 64, precision 0.73, recall 0.70
RNN compression part (0.67bpp, PSNR 48.5dB)

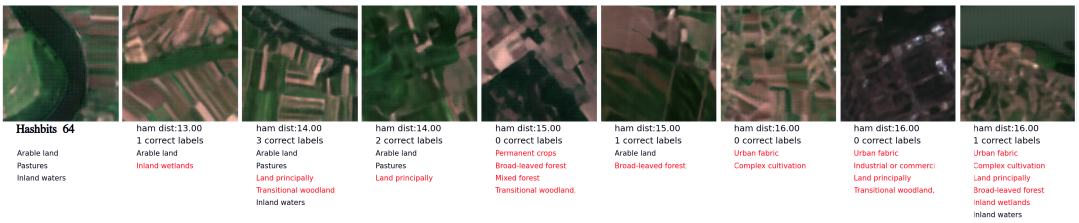
Figure 5.12: Comparison of retrieved images in the image domain and in the bitstream domain



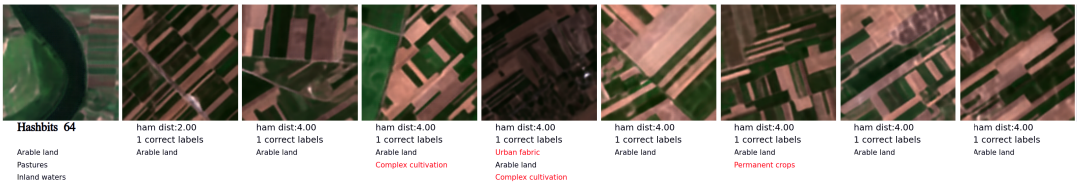
(a) CNN compression part (0.63bpp, PSNR 48,7dB), hashbits 64, precision 0.72, recall 0.69

(b) CNN compression part (0.79bpp, PSNR 49.7dB), hashbits 64, precision 0.73, recall 0.68

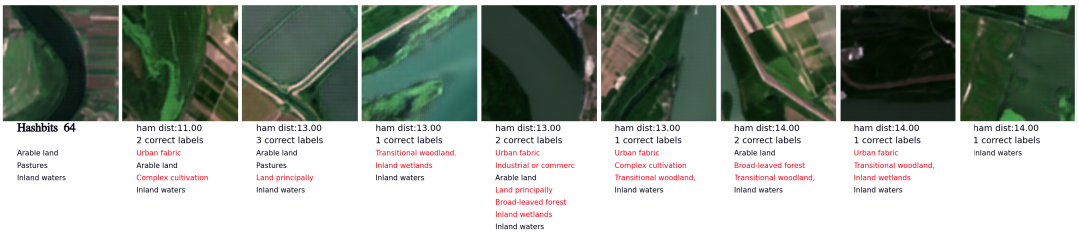
(c) CNN compression part (1.03bpp, PSNR 50.5dB), hashbits 64, precision 0.72, recall 0.68



(d) RNN compression part (0.51bpp, PSNR 47.3dB), hashbits 64, precision 0.71, recall 0.70



(e) RNN compression part (0.67bpp, PSNR 48.5dB), hashbits 64, precision 0.73, recall 0.70



(f) RNN compression part (0.84bpp, PSNR 49.4dB), hashbits 64, precision 0.71, recall 0.70

Figure 5.13: Comparison of retrieved images in the bitstream domain when joint training with different compression parts

6 Conclusion

With the advancement in space exploration, very large volumes of RS data are archived on a daily basis. This necessitates the need for compression and hashing for efficient storage, transmission, and retrieval from large-scale RS archives. Compression and hashing are treated as separate problems in literature and worked on independently. A joint framework would rather save computational time and provide a solution for both the problems at a single pass.

This thesis addressed the very problem of joining the two tasks into a multi-task framework which simultaneously generates compressed bitstream and hash codes from the input RS images. It also provided effective optimization strategies for both compression and hashing tasks during the training, so that the multi-task framework can get competitive compression and hashing performance when compared to the corresponding single-task baseline. An additional advantage of this proposed framework is that it avoided the need for any hyperparameter search when training the joint model or the separate parts involved in the framework.

Since the proposed framework consisted of two parts: compression with an autoencoder and hashing with shallow CNN, separate optimization schemes were proposed to minimize the interference between these two parts. The first stage was to optimize the CNN compression part with MGDA [16] to get a wide range of RD trade-off points. This approach avoided training the network multiple times from scratch for different bit-rates as done in the literature. The proposed compression part with the optimization algorithm was compared against the state-of-the-art RNN based compression and JPEG 2000. When evaluating rate-distortion performance, the CNN based compression outperformed the RNN based compression and JPEG 2000. When evaluating compression/decompression time, CNN based compression was faster than RNN based compression, but significantly slower than JPEG 2000.

In order to select proper hash coding function and optimization algorithm for the hashing part, a hashing baseline, which consisted of the same autoencoder as that in the proposed framework, was designed and tested in terms of retrieval performance. PCGrad [70] was selected to optimize the hashing loss, and Greedy Hash [57] was selected for efficient hash coding in the hashing part.

The hashing part of the proposed framework was trained on the image features extracted from the pre-trained compression part. It was observed that the initial retrieval performance of the multi-task framework was much lower than that of the hashing network trained separately. This is due to the fact that the hashing part was directly trained on image features provided by the compression part, which contained a lot of details that were beneficial for image reconstruction but were not that suitable for image retrieval. In order to narrow this gap of hashing performance between the multi-task framework and the hashing baseline, a very small learning rate was set to the compression part, so that the shared image features can be fine-tuned by the hashing part. Also, an attention module was inserted in front of the hashing network so that the hashing part can extract more task-specific information from the shared image features. After adding up these two improvements, the retrieval performance of the multi-task framework was observed to be at

par with that of the hashing baseline.

The thesis further explored the impact of the compression part on the retrieval performance of the multi-task framework. It was observed that when different pre-trained compression models were loaded to the compression part, the multi-task framework got similar retrieval scores and retrieved images. Hence making the proposed framework unaltered with the change in the compression part.

Finally, the thesis also analyzed the impact of placing the hashing part at the encoder and decoder side of the compression part. It was observed here that when the hashing part was trained at the decoder side, and hash codes were generated from latents that were recovered from the bitstream, the multi-task framework still got similar retrieval scores and retrieved images when compared to that trained with the hashing part at the encoder side. In other words, the proposed framework got very similar retrieval performance in the image domain and the compressed domain. There are extra benefits when the retrieval can be done in the compressed domain, i.e, when the bandwidth is limited, only the bitstream and meta information are required to be sent to the receiver. The retrieval at the receiver side can be carried out on hash codes which are generated from the bitstream.

Directions for Future Work

The compression part used in the framework is presently made of 2D convolution layers. This does not exploit the spectral redundancy in the RS data to the full extent. The compression performance on RS images can be improved by adding 3D convolution layers into the feature extraction modules so that the image features can be extracted in both spatial and spectral dimensions, which would further reduce the redundancy in the latent and lead to fewer bit-rates for entropy coding.

The distortion loss of the compression part is made of SSIM [65], which focuses on preserving the spatial structural similarity between the input image and the reconstructed image. However, for multi-spectral RS images, the reconstruction on the spectral dimension is equally important to that on the spatial dimension. Thus, the reconstruction quality of RS images could be further improved by adding a distortion loss that focuses on preserving the spectral similarity between input and reconstructed image.

The present hashing part is a deep supervised method, which requires a lot of image labels during the training. Since it is impractical to get so many labels for large-scale RS archives, the hashing part can be improved to a semi-supervised deep hashing method, which uses a few ground-truth labels and pseudo labels predicted by the classification layer for hash code learning.

Bibliography

[1] Glen P. Abousleman, Michael Marcellin, and Bobby R. Hunt. "Compression of hyperspectral imagery using the 3-D DCT and hybrid DPCM/DCT". In: *IEEE Transactions on Geoscience and Remote Sensing* 33.1 (1995), pp. 26–34. DOI: 10.1109/36.368225.

[2] Estanislau Augé et al. "Performance impact of parameter tuning on the CCSDS-123 lossless multi- and hyperspectral image compression standard". In: *Journal of Applied Remote Sensing* 7.1 (2013), pp. 1–16. DOI: 10.1117/1.jrs.7.074594.

[3] Johannes Ball, Valero Laparra, and Eero P. Simoncelli. "End-to-end optimization of nonlinear transform codes for perceptual quality". In: *2016 Picture Coding Symposium (PCS)* (2016), pp. 1–5.

[4] Johannes Ballé et al. "Variational image compression with a scale hyperprior". In: *International Conference on Learning Representations* (2018).

[5] Daniel Báscones, Carlos González, and Daniel Mozos. "Hyperspectral Image Compression Using Vector Quantization, PCA and JPEG2000". In: *Remote Sensing* 10.6 (2018). ISSN: 2072-4292. DOI: 10.3390/rs10060907. URL: https://www.mdpi.com/2072-4292/10/6/907.

[6] Saikat Basu et al. "DeepSat: A Learning Framework for Satellite Imagery". In: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems* (2015), pp. 1–10.

[7] Jean Bégaint et al. "CompressAI: a PyTorch library and evaluation platform for end-to-end compression research". In: *arXiv preprint arXiv:2011.03029* (2020).

[8] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *CoRR* abs/1308.3432 (2013). arXiv: 1308.3432. URL: http://arxiv.org/abs/1308.3432.

[9] Tong Chen et al. "End-to-End Learnt Image Compression via Non-Local Attention Optimization and Improved Context Modeling". In: *IEEE Transactions on Image Processing* 30 (2021), pp. 3179–3191.

[10] Zhao Chen et al. "GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks". In: *ICML* 80 (2018), pp. 794–803.

[11] Zhengxue Cheng et al. "Learned Image Compression With Discretized Gaussian Mixture Likelihoods and Attention Modules". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 7936–7945. DOI: 10.1109/CVPR42600.2020.00796.

[12] Zhengxue Cheng et al. “Learned Image Compression with Discretized Gaussian Mixture Likelihoods and Attention Modules”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).

[13] Tat-Seng Chua et al. “NUS-WIDE: a real-world web image database from National University of Singapore”. In: *Proceedings of the ACM International Conference on Image and Video Retrieval*. CIVR ’09 (), p. 9.

[14] Minnen David et al. “Spatially adaptive image compression using a tiled deep network”. In: *2017 IEEE International Conference on Image Processing (ICIP)* (2017), pp. 2796–2800. DOI: 10.1109/ICIP.2017.8296792.

[15] Begüm Demir and Lorenzo Bruzzone. “Hashing-Based Scalable Remote Sensing Image Search and Retrieval in Large Archives”. In: *IEEE Transactions on Geoscience and Remote Sensing* 54.2 (2016), pp. 892–904. DOI: 10.1109/TGRS.2015.2469138.

[16] Jean-Antoine Désidéri. “Multiple-gradient descent algorithm (MGDA) for multiobjective optimization”. In: *Comptes Rendus Mathematique* 350.5 (2012), pp. 313–318. ISSN: 1631-073X. DOI: https://doi.org/10.1016/j.crma.2012.03.014. URL: https://www.sciencedirect.com/science/article/pii/S1631073X12000738.

[17] Pier Luigi Dragotti, Giovanni Poggi, and Arturo R.P.Ragozini. “Compression of multispectral images by three-dimensional SPIHT algorithm”. In: *IEEE Transactions on Geoscience and Remote Sensing* 38.1 (2000), pp. 416–428.

[18] Qian Du and James E. Fowler. “Hyperspectral Image Compression Using JPEG2000 and Principal Component Analysis”. In: *IEEE Geoscience and Remote Sensing Letters* 4.2 (2007), pp. 201–205. DOI: 10.1109/LGRS.2006.888109.

[19] Ting Gong et al. “A Comparison of Loss Weighting Strategies for Multi task Learning in Deep Neural Networks”. In: *IEEE Access* 7 (2019), pp. 141627–141632. DOI: 10.1109/ACCESS.2019.2943604.

[20] Michelle Guo et al. “Dynamic Task Prioritization for Multitask Learning”. In: *ECCV* 11220 (2018), pp. 282–299.

[21] Pengwei Hao and Qingyun Shi. “Reversible integer KLT for progressive-to-lossless compression of multiple component images”. In: *Proceedings 2003 International Conference on Image Processing* 1 (2003), pp. I–633. DOI: 10.1109/ICIP.2003.1247041.

[22] Yueyu Hu, Wenhan Yang, and Jiaying Liu. “Coarse-to-Fine Hyper-Prior Modeling for Learned Image Compression”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.07 (2020), pp. 11013–11020.

[23] Khawar Islam et al. “Image Compression with Recurrent Neural Network and Generalized Divisive Normalization”. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2021), pp. 1875–1879.

[24] Adrián Javaloy and Isabel Valera. “Rotograd: Dynamic Gradient Homogenization for Multi-Task Learning”. In: *arXiv preprint arXiv:2103.02631* (2021).

[25] Qing-Yuan Jiang, Xue Cui, and Wu-Jun Li. "Deep Discrete Supervised Hashing". In: *IEEE Transactions on Image Processing* 27.12 (2018), pp. 5996–6009. DOI: 10.1109/TIP.2018.2864894.

[26] Azam Karami, Mehran Yazdi, and Grégoire Mercier. "Compression of Hyperspectral Images Using Discerete Wavelet Transform and Tucker Decomposition". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 5.2 (2012), pp. 444–450. DOI: 10.1109/JSTARS.2012.2189200.

[27] Pieter Kempeneers and Pierre Soille. "Optimizing Sentinel-2 image selection in a Big Data context". In: *Big Earth Data* 1 (Dec. 2017), pp. 145–158.

[28] Alex Kendall, Yarin Gal, and Roberto Cipolla. "Multi-task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 7482–7491.

[29] Fanqiang Kong et al. "A residual network framework based on weighted feature channels for multispectral image compression". In: *Ad Hoc Networks* 107 (2020), p. 102272. ISSN: 1570-8705.

[30] Fanqiang Kong et al. "Spectral-Spatial Feature Partitioned Extraction Based on CNN for Multispectral Image Compression". In: *Remote Sensing* 13.1 (2021), pp. 2072–4292.

[31] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *Master's thesis, University of Toronto* (May 2012).

[32] Hanjiang Lai et al. "Simultaneous feature learning and hash coding with deep neural networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 219–234.

[33] Jooyoung Lee et al. "Extended End-to-End optimized Image Compression Method based on a Context-Adaptive Entropy Model". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (2019).

[34] Jin Li and Zilong Liu. "Multispectral Transforms Using Convolution Neural Networks for Remote Sensing Multispectral Image Compression". In: *Remote Sensing* 11.7 (2019). ISSN: 2072-4292.

[35] Mu Li et al. "Learning Convolutional Networks for Content-Weighted Image Compression". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 3214–3223.

[36] Peng Li and Peng Ren. "Partial Randomness Hashing for Large-Scale Remote Sensing Image Retrieval". In: *IEEE Geoscience and Remote Sensing Letters* 14.3 (2017), pp. 464–468. DOI: 10.1109/LGRS.2017.2651056.

[37] Qi Li et al. "Deep Supervised Discrete Hashing". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017), pp. 2479–2488.

[38] Wu-Jun Li, Sheng Wang, and Wang-Cheng Kang. "Feature Learning Based Deep Supervised Hashing with Pairwise Labels". In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence* (2016), pp. 1711–1717.

[39] Yansheng Li et al. “Large-Scale Remote Sensing Image Retrieval by Deep Hashing Neural Networks”. In: *IEEE Transactions on Geoscience and Remote Sensing* 56 (2018), pp. 950–965.

[40] Yunqiang Li and Jan van Gemert. “Deep Unsupervised Image Hashing by Maximizing Bit Entropy”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.3 (2021), pp. 2002–2010.

[41] Xi Lin et al. “Pareto Multi-Task Learning”. In: *Thirty-third Conference on Neural Information Processing Systems (NeurIPS)* (2019), pp. 12037–12047.

[42] Haomiao Liu et al. “Deep Supervised Hashing for Fast Image Retrieval”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2064–2072.

[43] Shengchao Liu, Yingyu Liang, and Anthony Gitter. “Loss-Balanced Task Weighting to Reduce Negative Transfer in Multi-Task Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33 (2019), pp. 9977–9978.

[44] Shikun Liu, Edward Johns, and A. Davison. “End-To-End Multi-Task Learning With Attention”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 1871–1880.

[45] Xiao Luo et al. “A Survey on Deep Hashing Methods”. In: *CoRR* abs/2003.03369 (2020). URL: https://arxiv.org/abs/2003.03369.

[46] Fabian Mentzer et al. “Conditional Probability Models for Deep Image Compression”. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 4394–4402.

[47] David Minnen, Johannes Ballé, and George Toderici. “Joint Autoregressive and Hierarchical Priors for Learned Image Compression”. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018), pp. 10794–10803.

[48] “MLRSNet: A multi-label high spatial resolution remote sensing dataset for semantic scene understanding”. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 169 (2020), pp. 337–350. ISSN: 0924-2716.

[49] Johnston Nick et al. “Improved Lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2018), pp. 4385–4393.

[50] *Planet Satellite Imagery and Archive*. https://www.planet.com/products/planet-imagery/. Accessed: 2021-09-30.

[51] Tapani Raiko et al. “Techniques for Learning Binary Stochastic Feedforward Neural Networks”. In: *International Conference on Learning Representations* (2015), pp. 1–10.

[52] Thomas Reato, Begüm Demir, and Lorenzo Bruzzone. “An Unsupervised Multicode Hashing Method for Accurate and Scalable Remote Sensing Image Retrieval”. In: *IEEE Geoscience and Remote Sensing Letters* 16.2 (2019), pp. 276–280. DOI: 10.1109/LGRS.2018.2870686.

[53] Subhankar Roy et al. "Deep Metric and Hash-Code Learning for Content-Based Retrieval of Remote Sensing Images". In: *IEEE International Geoscience and Remote Sensing Symposium* (2018), pp. 4539–4542.

[54] Ozan Sener and Vladlen Koltun. "Multi-Task Learning as Multi-Objective Optimization". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018), pp. 525–536.

[55] Fumin Shen et al. "Deep Asymmetric Pairwise Hashing". In: *Proceedings of the 25th ACM International Conference on Multimedia* (2017), pp. 1522–1530.

[56] Weiwei Song, Shutao Li, and Jón Atli Benediktsson. "Deep Hashing Learning for Visual and Semantic Retrieval of Remote Sensing Images". In: *IEEE Transactions on Geoscience and Remote Sensing* 59 (2021), pp. 9661–9672.

[57] Shupeng Su et al. "Greedy Hash: Towards Fast Optimization for Accurate Hash Coding in CNN". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (2018), pp. 806–815.

[58] Gencer Sumbul et al. "BigEarthNet: A Large-Scale Benchmark Archive for Remote Sensing Image Understanding". In: *IEEE International Geoscience and Remote Sensing Symposium, Yokohama, Japan* (2019).

[59] Gencer Sumbul et al. "BigEarthNet-MM: A Large-Scale, Multimodal, Multilabel Benchmark Archive for Remote Sensing Image Classification and Retrieval [Software and Data Sets]". In: *IEEE Geoscience and Remote Sensing Magazine* 9.3 (2021), pp. 174–180. DOI: 10.1109/MGRS.2021.3089174.

[60] Wim Sweldens. "The Lifting Scheme: A Custom-Design Construction of Biorthogonal Wavelets". In: *Applied and Computational Harmonic Analysis* 3.2 (1996), pp. 186–200. ISSN: 1063-5203. DOI: https://doi.org/10.1006/acha.1996.0015. URL: https://www.sciencedirect.com/science/article/pii/S1063520396900159.

[61] Dang-Khoa Le Tan, Thanh-Toan Do, and Ngai-Man Cheung. "Learning to hash with binary deep neural network". In: *European Conference on Computer Vision. Springer* (2016), pp. 219–234.

[62] Lucas Theis et al. "Lossy Image Compression with Compressive Autoencoders". In: *5th International Conference on Learning Representations, ICLR* (2017).

[63] George Toderici et al. "Variable Rate Image Compression with Recurrent Neural Networks". In: *4th International Conference on Learning Representations,ICLR 2016* (2016).

[64] Xiaofang Wang, Yi Shi, and Kris M. Kitani. "Deep Supervised Hashing with Triplet Labels". In: *Asian Conference on Computer Vision* 10111 (2016), pp. 70–84.

[65] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612. DOI: 10.1109/TIP.2003.819861.

[66] Zirui Wang et al. "Gradient Vaccine: Investigating and Improving Multi-task Optimization in Massively Multilingual Models". In: *International Conference on Learning Representations* (2021).

[67] Rongkai Xia et al. “Supervised Hashing for Image Retrieval via Image Representation Learning”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence* (2014), pp. 2156–2162.

[68] Chen Yaxiong and Lu Xiaoqiang. “Deep discrete hashing with pairwise correlation learning”. In: *Neurocomputing* 385 (2020), pp. 111–121.

[69] Shawn Newsam Yi Yang. “Bag-Of-Visual-Words and Spatial Extensions for Land-Use Classification”. In: *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS)* (2010).

[70] Tianhe Yu et al. “Gradient Surgery for Multi-Task Learning”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5824–5836.

[71] Qingyu Zhang, Dong Liu, and Houqiang Li. “Deep network-based image coding for simultaneous compression and retrieval”. In: *2017 IEEE International Conference on Image Processing (ICIP)* (2017), pp. 405–409.

[72] Zheng Zhang et al. “Improved Deep Hashing With Soft Pairwise Similarity for Multi-Label Image Retrieval”. In: *IEEE Transactions on Multimedia* 22.2 (2020), pp. 540–553.