

Capsule Networks for Hyperspectral Image Classification: From Theory to Practice

Zeno Dhaene

Student number: 01600506

Supervisor: Prof. dr. ir. Aleksandra Pizurica

Counsellors: Xian Li, Nina Žižakic, Dr. Shaoguang Huang

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science Engineering

Academic year 2020-2021

Acknowledgment

With the end of this thesis also comes the end of my time as a university student, a five-year period that I will cherish for many decades to come. Over the last two semesters, I delved into the exciting world of hyperspectral image classification, a topic that is now engraved in my brain and undoubtedly will be for many more years. Now that this period of hard work has come to an end, I would like to thank the people that have been instrumental in guiding and advising me in creating this thesis.

First and foremost, I would like to thank my counsellors Nina Žižakić, Dr. Shaoguang Huang and Xian Li for the biweekly meetings, listening to my questions and giving valuable answers. They truly lifted this thesis to a higher level. I would also like to thank Prof. dr. ir. Aleksandra Pizurica for the valuable feedback, and for allowing and supporting me in taking a more practical approach in this thesis.

In a year characterized by working from home, it is more than fair that I thank my mom, Ruth Raes, for providing me with the comforts of home, and my sister, Elodie Dhaene for listening to my very detailed explanations of the topic that was on my mind for 8 months straight. I would also like to thank my dear friend Maxime Deforche, for listening to my trials and tribulations during our regular walks outside.

Thank you all very much,
Zeno

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation.

May 22, 2021

Capsule Networks for Hyperspectral Image Classification: From Theory to Practice

Zeno Dhaene

Supervisors: Prof. dr. ir. Aleksandra Pizurica, Nina Žižakić, Dr. Shaoguang Huang, Xian Li

Abstract—Understanding the Earth and seeing how it evolves is an important field of study with applications in climate research and land usage. Hyperspectral image classification is the process of assigning a geological label to each pixel of images captured by dozens or hundreds of sensors on aircrafts or satellites, and has greatly benefited from the rise of deep learning techniques. In this thesis, I propose a new multi-stream architecture based on capsule networks, a supervised deep learning technique that models hierarchical relationships of pixels in spatial regions. This new architecture combines the spectral features captured by the sensors with the information of the spatial neighborhood in a 3D model. Apart from the theoretical modelling of a classifier, I develop a user-friendly software tool for classifying unlabeled hyperspectral images that can be used for both practical and research purposes. In this web application, hyperspectral images and architectures with corresponding hyperparameters are combined to create a classification map of the terrain.

Keywords— hyperspectral images, hyperspectral image classification, capsule networks, classification tools, web applications, remote sensing

I. INTRODUCTION

THE assignment of a geological class to an area on the globe can be useful information for many applications. For example, the agricultural productivity of an area may be linked to the area covered by a specific crop, to determine soil fertility and optimal sowing strategies [3]. In climate research, we can monitor the area covered by lakes, deserts, and glaciers to better predict how climate change will affect the Earth in the years to come. Gathering these statistics is a difficult process, and involves looking at aerial and satellite images. However, the information contained in these RGB images is limited, and lacks detailed information on distinguishing different crop types.

Hyperspectral images can provide this required additional information by capturing more than the three traditional spectral bands red (wavelengths 564–580 nm), green (wavelengths 534–545 nm) and blue (wavelengths 420–440 nm), but measure dozens or even hundreds of spectral wavelengths ranging from 400 nm to far beyond the capabilities of human vision, up to 2500 nm. Each sensor therefore captures light in a specific way, leading some sensors to see the difference between two crop types, and other sensors able to discern water from land. Combining the output of these sensors creates a data cube with two spatial dimensions - representing the relative location of the pixels - and one spectral dimension with the sensor values of the different spectral bands. A single pixel in the dataset covers a certain area, determined by the resolution of the dataset. The sensor values that were captured over that area is called the spectral signature of the area.

In Figure 1c, we can see the ground truth of the Salinas dataset, an academic dataset of a part of the Salinas valley in California, one of the most productive agricultural regions in

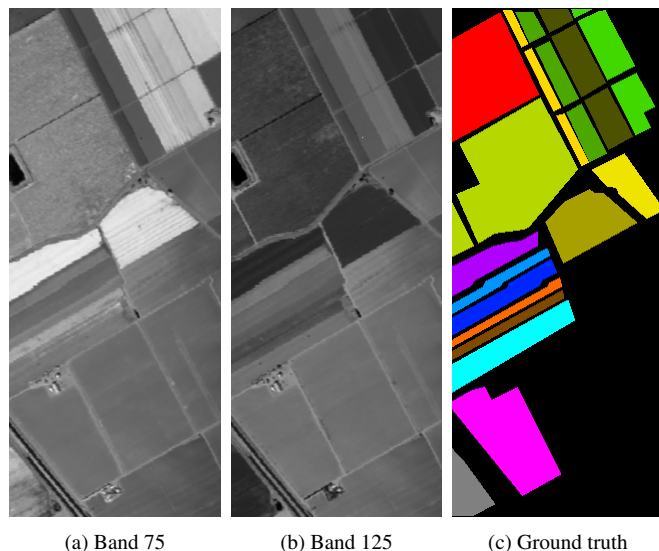


Fig. 1: Spectral bands 75 and 125, and the ground truth map of the Salinas dataset

California. The different colors indicate different crop types, ranging from celery and lettuce fields to vineyards. In this particular dataset, which measures 512×217 pixels, has a resolution of 3.7 metres. Therefore, each pixel covers an area of $3.7m \times 3.7m = 13.69m^2$, with the entire dataset covering approximately $1.52 km^2$. Each pixel in the hyperspectral image has a spectral signature consisting of 224 values. An example of a normalized spectral signature is provided in Figure 2. Normalization of the different spectral bands is necessary, as the variance within the bands can range from 10^1 to 10^7 .

II. RELATED WORK

A. Hyperspectral image classification

In supervised hyperspectral image classification, a hyperspectral image is classified by feeding a limited amount of labeled training samples to a machine learning architecture which can be used to predict the geological class for the remaining pixels in the dataset. The network architecture can either be 1-dimensional, by taking only the spectral values of the target pixel as input, or multidimensional. 1-dimensional architectures like the deep belief network by Chen et al. [4] is an example of such a network that only uses spectral information for the currently classified pixel. They note that their proposed architecture, which uses spatial information as well, performs significantly better.

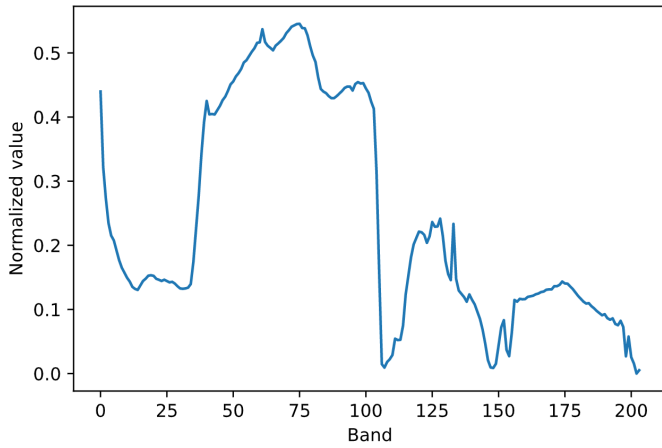


Fig. 2: Spectral signature

In both 2 and 3 dimensions, the spatial neighbourhood of the target pixel is taken into account. For 2D architectures, often PCA is used to select the most relevant spectral slices over the pixels in the spatial neighborhood. An example is 2D-CNN [6], which uses PCA to reduce the dimensionality of the hyperspectral image from three dimensions to 2 dimensions. Computationally, this delivers faster and sometimes more accurate models compared to 3D architectures where all spectral bands of all pixels in the neighborhood are included in the input data. 3D architectures like multi-scale wavelet 3D convolutional neural network (MW-3D-CNN) [7] and multi-scale 3D deep convolutional neural network (M3D-DCNN) [8] often use multiple streams or scales to reduce the level of overfitting that accompanies higher-dimensional data.

Hybrid solutions are available as well: M3D-DCNN [5] is a multi-scale deep convolutional neural network proposed for HSI classification, which can jointly learn both 2D multi-scale spatial features and 1D spectral features. Additionally, there are a number of architectures that provide the entire dataset as training sample to the network. This can be done in a Graph Convolutional Network like in LWCNN [9] and TGRS-CNN [10], which exploit spatial information by representing the pixels as a connected graph, where two pixels are connected if they are similar in spectral signature while not necessarily being adjacent to each other.

Apart from supervised techniques, there are semi-supervised unsupervised techniques for transfer learning, in an attempt to classify unlabeled hyperspectral images using previously obtained models. These deep transfer learning models like multiscale spectral-spatial networks (MSSN) [11] and the Deep Domain Adaptation architecture (DDA-Net) [12] utilize modules in subsequent stages to train an architecture that can be used for transfer learning. Finally, there are architectures that use more than hyperspectral data alone, like the Multimodal Deep Learning Meets Remote Sensing framework (MDL-RS) [13] which uses multiple modalities like heightmaps to classify the dataset.

B. Capsule Networks

Capsule networks were proposed only recently [1] and are seen as an extension of traditional convolutional neural networks (CNN). They exploit hierarchical features, meaning that the relative position of two detected features is also important. In the domain of general image classification, we might draw the parallel with detecting face, where the existence of two eyes and one mouth is not enough to detect a face. In contrast to CNN, where the output of a neuron is a scalar, capsule layers output a vector instead. The neurons in CNN are called capsules in capsule networks. The norm of the output vector represents the confidence of the capsule that a certain learned feature is present in the convolutional window. The direction of the vector describes its instantiating parameters. This describes the hue or brightness of a color, or the rotation of a feature. In practice, these instantiating parameters are not as intuitive since they are learned from the data.

Usually, a capsule network consists of two layers: a primary and a secondary layer. Applied to general image classification, the primary layer would detect basic features, such as eyes and ears, while the secondary capsule layer would combine the output vectors of the primary layer to create more complex features like a face. In hyperspectral image classification too, this hierarchical structure of features can be detected, by combining spectral and spatial features together into the predicted geological class.

III. PROPOSED MULTI-STREAM CAPSULE ARCHITECTURE

A difficulty with hyperspectral images is that datasets are usually not labeled. This creates the need of some sort of pre-processing stage, either manually or automatically through processes like clustering, where training samples and their respective classes are initialized. Therefore, it is useful to create an architecture that can give accurate classification results with relatively little training data. Additionally, a 3-dimensional architecture contains the most information about the target pixel, as the spatial neighborhood of the pixel has been shown to be very important for accurate classification results and PCA is not applied. However, computational complexity for 3D training samples is high and suffers from the curse of dimensionality, degrading classification performance.

The final difficulty that will be addressed in the proposed architectures is the influence of the neighborhood size on the classification results. The neighborhood size must be chosen large enough, so that outliers in the spectral signature of the target pixel or its immediate neighbours don't influence the prediction. However, a neighborhood that is chosen too large will result in lost details, as information is mixed over a big area. This might cause very narrow geological classes to disappear.

To remedy these limitations, I propose a new 3D, multi-stream architecture based on capsule networks. The architecture takes training samples with dimensions $n \times n \times d$, where n (odd) is the highest neighborhood size and d is the number of spectral bands in the dataset. The center column with dimension

$1 \times 1 \times d$ denotes the spectral values of the target pixel. For the training process, 50 labeled samples per class are used, which is approximately 1.5% of the labeled pixels in the Salinas dataset. This number is chosen empirically, and delivers a balance between under- and overfitting, taking the problem of unlabeled datasets into account. The training process starts in three distinct streams, where each stream takes in a smaller neighborhood radius around the target pixel. This way, the benefits of having detailed information about edges between geological classes is preserved while also having the robustness of large neighborhood sizes. The balance between the larger and smaller streams is learned by the architecture.

The 3-dimensional training samples are fed through the network in batches. First, a convolutional filter is applied to reduce the dimensionality of the data cube. Afterwards, the 3D data is fed into the primary capsule layer, where the primary features are learned. The squash function alters the output vectors to have a length between 0 and 1. These squashed vectors are then concatenated into a single stream, and are provided as features to the secondary capsule layer, which has k capsules, with k the number of labeled classes in the ground truth of the dataset. Each output vector of the k capsules is again squashed, making the norm of the vectors interpretable as the confidence of the network that the target pixel belongs to that class. The output vector with the highest norm is chosen as the predicted class of the network.

IV. WEB APPLICATION

Current state-of-the-art models achieve an accuracy in excess of 99%, which is also a score obtained by the proposed architecture on some datasets. These performance numbers are good enough to be applied in practice, where we can compare classification performance in an academical context to a practical context. A lot of hyperspectral images are available openly and for free through various satellite programs of NASA and ESA. These datasets are completely unlabeled, creating the need of a manual preprocessing step.

I present an application that can be used both by remote sensing researchers and machine learning experts in the field to classify unlabeled hyperspectral images. The application can generate a complete classification map for any hyperspectral image, along with the area coverage for each class. To achieve this, the application, which is deployed as a web application, requires 4 inputs: (1) the unlabeled hyperspectral image, (2) manually assigned training samples - indicated through the user interface, (3) an architecture - which can be uploaded by the researcher and used by the expert, and (4) the hyperparameters for the architecture. The output is a classification map, the area per class indicated in pixels or km^2 if the resolution was provided, and a confusion map. This confusion map can be used to see whether hyperparameters or training samples need to be adjusted.

The web application is written in a way that models are enqueued on a server and trained one after the other, which enables multiple users to use the system at the same time without needing powerful hardware, with the additional benefit that a lot

of models can be enqueued and are run automatically without the need for supervision. The source code of this application is available on GitHub [2].

V. RESULTS

The results in this section were obtained by running the proposed multi-stream architecture on both labeled datasets and unlabeled datasets. For the labeled datasets, the performance of a model trained with manually indicated classes is compared to the ground truth of the dataset. Per dataset, the hyperparameters of the model are fixed regardless of whether the model is trained with manually labeled or ground truth training samples. Per class, 50 labeled samples were used in the training set and 50 different labeled samples were used in the validation set. The multi-stream model consists of three streams, with neighborhood sizes 3×3 , 7×7 , and 11×11 .

A. Classifying academical HSI

In Table I, we compare the single-stream architecture with the proposed multi-stream architecture. It is clear that there is an improvement in classification performance by going from a single stream to three streams. The difference lies between 1 and 4 percent with regards to overall accuracy and average accuracy, as well as an increase of up to 0.04 for Cohen's Kappa metric.

Dataset	OA	AA	$\kappa \times 100$
Salinas: single-stream	83.44	92.08	83.88
Salinas: multi-stream	87.15	94.51	86.42
Indian Pines: single-stream	65.70	81.19	60.94
Indian Pines: multi-stream	70.52	84.02	66.22
PaviaU: single-stream	97.31	96.91	96.31
PaviaU: multi-stream	98.25	98.13	98.09

TABLE I: Overall Accuracy (OA), Average Accuracy (AA), and Cohen's Kappa metric comparison between a single stream and multiple streams. The kappa metric measures how much better the classifier performs in comparison to a random classifier.

Now we look at the impact of manually indicated pixels compared to using the ground truth for training samples: in Figure 3a, the manually defined labels are indicated. Training samples for the manual training result are chosen from these manually indicated class areas. In Figure 3b, the ground truth for the Salinas dataset is shown, which serves as a baseline for the results achieved in the model that used training samples from the ground truth data (Figure 3c) and the model that used manual initialization (Figure 3d).

With an overall accuracy of 82.82% and an average accuracy of 89.61%, the classification performance of the model that uses manually indicated training samples performs worse than the model that used the ground truth training samples, which achieved an overall accuracy of 88.80% and an average accuracy of 94.80% compared to the ground truth data. Note that in both figures, the pixels that did not have a class in the ground truth data have been colored black, although they too were classified. This performance loss is to be expected, as the set of

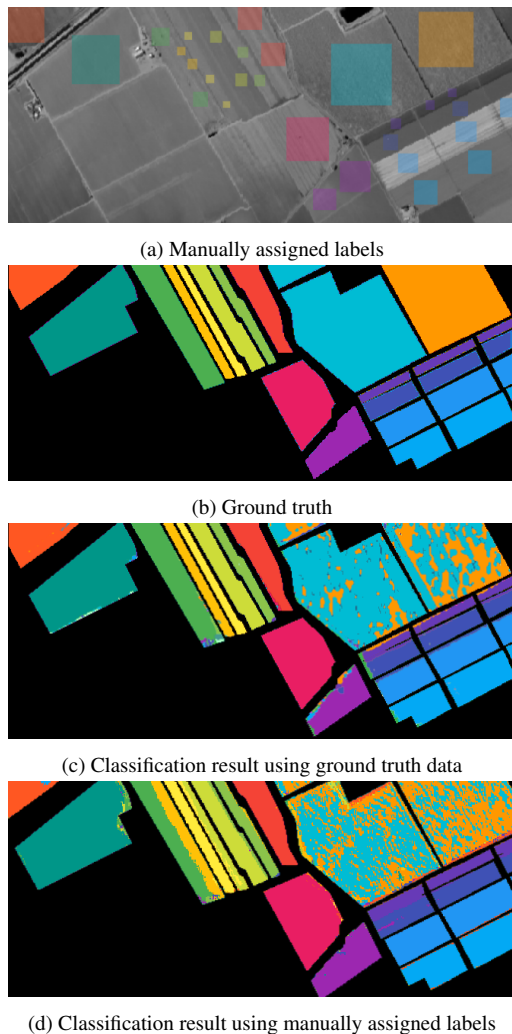


Fig. 3: Influence of ground truth vs manual labeling on classification performance on the Salinas dataset

available training samples is less diverse when using manually indicated classes.

B. Unlabeled satellite dataset

In Figure 4a, we can see a satellite image of the island of Moroni, near the coast of Mozambique. This hyperspectral image was captured by the Earth Observation 1 satellite, which uses the AVIRIS sensor to capture hyperspectral data. The resolution of this dataset is quite poor, although there are 2 big classes to be discerned: water and land. In Figure 4b, a grayscale visualization of a spectral slice is shown with 6 manually placed class areas. The classification result shown in Figure 4c, and is shown semi-transparent over the map view in Figure 4d, shows that the architecture is clearly capable of making the distinction between water and land.

VI. CONCLUSION

The goal of this thesis was to create a novel architecture for hyperspectral image classification using a capsule network architecture, and while it does not yet fully compete with the state of the art, I am confident that a follow-up article can create an

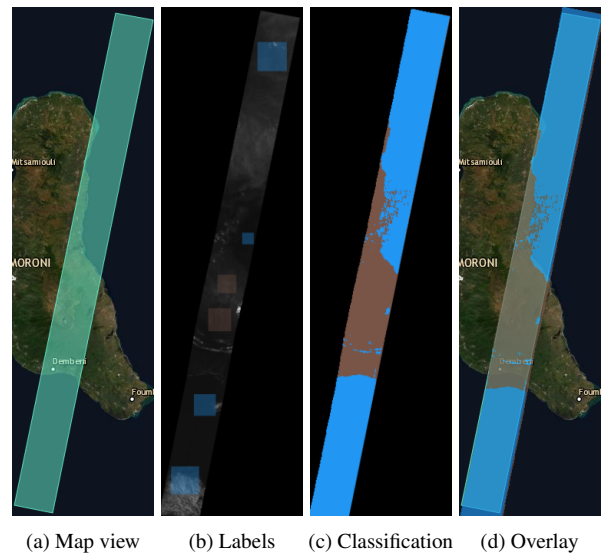


Fig. 4: Classification result on a manually labeled satellite dataset

architecture based on this 3D multi-stream capsule network architecture that can deliver good results. The other goal, applying the proposed architecture and other architectures to real-world datasets has been achieved by creating an application that is capable of training an unlabeled input dataset with the chosen architecture and chosen parameters.

REFERENCES

- [1] G. E. Hinton, S. Sabour, N. Frosst, *Dynamic Routing Between Capsules* <http://arxiv.org/abs/1710.09829>, 2017
- [2] Z. Dhaene, *Hyperspectral Classification Webapplication Source Code*, <https://github.com/zenodhaene/HyperspectralClassification>
- [3] J. Meola, M. T. Eismann, R. L. Moses, J. N. Ash *Application of Model-Based Change Detection to Airborne VNIR/SWIR Hyperspectral Imagery*, IEEE Transactions on Geoscience and Remote Sensing, vol. 50, no. 10, pp. 3693–3706, 2012
- [4] C. Yushi, Z. Xing, J. Xiuping *Spectral–Spatial Classification of Hyperspectral Data Based on Deep Belief Network*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 8, no. 6, pp. 2381-2392, 2015
- [5] H. Mingyi, L. Bo, C. Huahui *Multi-scale 3D deep convolutional neural network for hyperspectral image classification*, in Proc. IEEE International Conference on Image Processing (ICIP), pp. 3904-3908, 2017
- [6] X. Liu, H. Wang, Y. Meng and M. Fu *Classification of Hyperspectral Image by CNN Based on Shadow Area Enhancement Through Dynamic Stochastic Resonance*, in Proc. IEEE Access, vol. 7, pp. 134862-134870, 2019
- [7] Y. Jingxiang, Z. Yong-Qiang, C. J. Cheung-Wai, X. Liang, *A Multi-Scale Wavelet 3D-CNN for Hyperspectral Image Super-Resolution*, Remote Sensing, vol. 11, no. 13, article 1557, 2019
- [8] H. Mingyi, L. Bo, C. Huahui, *Multi-scale 3D deep convolutional neural network for hyperspectral image classification*, in Proc. 2017 IEEE International Conference on Image Processing (ICIP), pp. 3904-3908, 2017
- [9] J. Sen, L. Zhijie, X. Meng, H. Qiang, Z. Jun, J. Xiuping, L. Qingquan *A Lightweight Convolutional Neural Network for Hyperspectral Image Classification*, IEEE Transactions on Geoscience and Remote Sensing, vol. 59, no. 5, pp. 4150-4163, 2021
- [10] M. Lichao, L. Xiaoqiang, L. Xuelong, Z. Xiao Xiang *Nonlocal Graph Convolutional Networks for Hyperspectral Image Classification*, IEEE Transactions on Geoscience and Remote Sensing, vol. 58, no. 12, pp. 8246-8257, 2020
- [11] Z. Chongxiao, Z. Junping, W. Sifan, Z. Ye *Cross-Scene Deep Transfer Learning With Spectral Feature Adaptation for Hyperspectral Image Classification*, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 13, pp. 2861-2873, 2020
- [12] M. Xiaorui, M. Xuerong, W. Jie, L. Xiaokai, W. Hongyu, Y. Baocai *Cross-Data Set Hyperspectral Image Classification Based on Deep Domain Adap-*

- tation, IEEE Transactions on Geoscience and Remote Sensing, vol. 57, no. 12, pp. 10164-10174, 2019
- [13] H. Danfeng, G. Lianru, Y. Naoto, Y. Jing, C. Jocelyn, D. Qian, Z. Bing *More Diverse Means Better: Multimodal Deep Learning Meets Remote-Sensing Imagery Classification*, IEEE Transactions on Geoscience and Remote Sensing, vol. 59, no. 5, pp. 4340-4354, 2021

Contents

List of Figures	13
List of Tables	15
1 Introduction	16
1.1 Own contributions	17
2 An introduction to hyperspectral image classification	19
2.1 Hyperspectral image datasets	20
2.1.1 Salinas dataset	20
2.1.2 Indian Pines dataset	22
2.1.3 Pavia University dataset	22
2.1.4 An Earth Observation 1 Satellite dataset	24
2.2 Spatial neighborhood in training samples	24
2.3 Supervised learning	25
2.4 Limitations of theoretical models	29
3 Exploring capsule networks	31
3.1 Introducing capsule neural networks	31
3.1.1 Origin & idea	32
3.1.2 Primary and secondary capsule layers	32
3.1.3 Routing by agreement & the squashing function	33

<i>CONTENTS</i>	11
3.1.4 A simple architecture	35
3.2 Relevance to hyperspectral image classification	36
3.3 Experiments	37
3.4 Limitations	38
4 Proposed architecture: a multi-stream capsule network	41
4.1 Motivation	41
4.2 Architecture	43
4.3 Results	44
4.3.1 Salinas dataset	45
4.3.2 Indian Pines	45
4.3.3 Pavia University	47
4.4 Comparison with state of the art	47
5 Applying theory in practice: a web application	52
5.1 Goal of the application	53
5.2 Feature overview	55
5.2.1 Dashboard	55
5.2.2 Uploading a new dataset	57
5.2.3 Detailed dataset page	57
5.2.4 Uploaded datasets	57
5.2.5 Creating a new model	61
5.2.6 Model result overview page	64
5.2.7 Detailed model result	64
5.2.8 Earth Observation 1 satellite datasets	67
5.3 Software architecture	67
5.3.1 Frontend	68
5.3.2 Database	68

5.3.3	Backend	69
5.4	Experimental results on academic datasets	70
5.4.1	Indian Pines dataset	71
5.4.2	Salinas dataset	71
5.5	Experimental results on satellite datasets	71
6	Future work	73
6.1	Improving the multi-stream architecture	73
6.2	Future development on the web application	73
7	Conclusion	75
	Bibliography	76

List of Figures

2.1	Grayscale representation of a single spectral band and the ground truth of the Salinas dataset	21
2.2	Different slices from the Salinas dataset	22
2.3	Grayscale representation of a single spectral band and the ground truth of the Indian Pines dataset	23
2.4	Variations in the different spectral bands of the Indian Pines dataset	23
2.5	Normalized spectral values of a single pixel	24
2.6	Grayscale representation of a single spectral band and the ground truth of the Pavia University dataset	25
2.7	Grayscale representation of a single spectral band and the map view of the EO1 dataset .	26
3.1	Instance vectors: a core idea of capsule networks for hierarchical relationships	32
3.2	Routing by agreement: detecting objects from shapes	33
3.3	Routing by agreement: combining shapes into objects	34
3.4	The squash function used in routing by agreement	35
3.5	A simple capsule network architecture	36
3.6	A simple single-stream capsule network architecture	37
3.7	Classification maps for single-stream capsule network architecture	39
4.1	Impact of neighborhood size on classifying the Salinas dataset	42
4.2	Schematic overview of the proposed multi-stream architecture	43
4.3	Influence of increasing neighborhood of the multi-stream architecture on the classification performance	45
4.4	Comparing single-stream and multi-stream classification maps	46

4.5	Classification maps for the Salinas dataset	51
4.6	Classification maps for the Pavia University dataset	51
5.1	Confusion matrix for a partially inaccurate classification result (Salinas dataset)	54
5.2	The dashboard page	56
5.3	Uploading a new dataset	58
5.4	Detailed dataset page	59
5.5	The overview of uploaded datasets	60
5.6	Model creation page	62
5.7	Architecture selection page	63
5.8	The model overview page	65
5.9	The model result page	66
5.10	Software architecture of the web application	67
5.11	Ground truth, labels, and generated classification map for the Indian Pines dataset	71
5.12	Ground truth, labels, and generated classification map for the Salinas dataset	72
5.13	Classification maps for the satellite dataset	72

List of Tables

3.1	Single stream capsule network performance with regards to overall accuracy (OA), average accuracy (AA) and Cohen's kappa coefficient (κ)	38
4.1	Proposed architecture performance comparison on the Salinas dataset with regards to overall accuracy (OA), average accuracy (AA) and Cohen's kappa statistic (κ)	47
4.2	Proposed architecture performance comparison on the Indian Pines dataset with regards to overall accuracy (OA), average accuracy (AA) and Cohen's kappa statistic (κ)	48
4.3	Proposed architecture performance comparison on the Pavia University dataset with regards to overall accuracy (OA), average accuracy (AA) and Cohen's kappa statistic (κ)	49
4.4	Comparison of classification accuracies on the Salinas hyperspectral image with regards to overall accuracy (OA), average accuracy (AA) and Cohen's kappa statistic (κ)	49
4.5	Comparison of classification accuracies on the Pavia University hyperspectral image with regards to overall accuracy (OA), average accuracy (AA) and Cohen's kappa statistic (κ)	50

1

Introduction

The human species is driven by exploration. In the early ages exploration was driven by necessity, roaming the Earth for natural resources in the time of hunters and gatherers. From the 15th century to the 18th century, during the age of discovery, exploration was driven by the search for silver and gold, as well as to find a new route for the silk and spice trade. Nowadays, we can find humans from the desolate north pole to the crowded cities of East Asia. Exploration now is driven by the desire to understand our planet, from researching how the earth once looked like to seeing how it changes from year to year.

With climate change being one of the greatest challenges of the 21st century, the need to predict how our world will change over the coming decades has become an important field of academic research. Using satellite images, we can already see how glaciers around the world are shrinking at a staggering pace [1]. Capturing the bigger picture however, where we try to measure the area occupied by ice, desert, rain forest, cities, and other land types, can still be improved upon as they are often based on satellite images [2].

Hyperspectral image classification is the process of assigning each pixel of a hyperspectral image to a certain class, where the classes represent the different geological features present in the image. Examples of such classes are mountains, ocean, and farmland, but with hyperspectral images we can go much further: cornfield, river, road, and so on. If this classification process could be automated, we would be able to solve a number of problems much faster than we can currently, such as urban development [3], land change monitoring [4], resource management [5], and many more.

Hyperspectral images can provide a solution for the lack of accuracy in current classification methods that use traditional RGB images. Hyperspectral sensors measure reflected light in the same way as regular (RGB) cameras, but capture light on a wider range of spectral wavelengths. RGB images contain 3

bands: red (wavelengths 564–580 nm), green (wavelengths 534–545 nm), and blue (wavelengths 420–440 nm). Hyperspectral images contain dozens or even hundreds of bands, often beyond the capabilities of human vision. Hyperspectral images will be discussed in more detail in Chapter 2.

In this thesis, I focus on HSI classification based on capsule networks. These emerging models have already been successfully applied in some demanding tasks in computer vision, but have not yet been explored thoroughly in remote sensing. Compared to the common deep learning architectures such as convolutional neural networks (CNNs), capsule networks can better represent hierarchical representations in the data. Their power comes from an efficient vector representation of features, referred to as activity vectors. This efficient feature representation captures also spectral-spatial features from HSI data better than the more traditional deep learning models. Capsule networks and how they work will be discussed in detail in Chapter 3.

At the beginning of the thesis, I familiarised myself the problem by researching the latest work in hyperspectral image classification. Doing this also taught me a lot about the hyperspectral images themselves: what the data looks like, where the difficulties lie, and how classification performance is measured. My research was focused on supervised learning, as capsule networks are a supervised learning technique. It quickly became apparent that using as little training samples as possible in the supervised learning training process is very important, as practical datasets often have little or no labeled pixels. Getting unlabeled data however is not a problem: efforts by NASA and ESA have made a huge number of hyperspectral images available to the public. The summary of this research is given in Chapter 2, where I go over the problem statement, the academical and practical datasets that are available, an overview of the state of the art, and the limitations of current theoretical models.

With this research, I attempted to create my first deep learning model using convolutional neural networks (CNN). With my first model, it was time to discover the capsule networks technique. This recent technique is described in Chapter 3, where I go over the inner workings. In also provide a simple single-stream architecture for HSI classification, along with experimental results and limitations of this architecture.

1.1 Own contributions

In Chapter 4, I propose a new architecture that will eliminate some of the limitations of the single-stream architecture. The proposed architecture consists of multiple input streams, which extract different features based on the surrounding pixels of the target pixel. These features are concatenated and jointly trained to give the final prediction. The details of this model, along with experimental results and a comparison with the state of the art, are also included and discussed in this chapter.

While researching the state of the art and creating my own models for HSI classification, it became apparent that there is no tool to help the deep learning process for both machine learning researchers and those that want to apply the research in practice, without the need for a computer science background. This is important if we want to apply state-of-the-art architectures in practice. As running python code is not something we can expect from the end user, I started working on an application to ease the training and classification process. With this application, the user can upload a hyperspectral image. They can then either use the corresponding ground truth map if the dataset is labeled, or manually specify

classes to enable supervised learning. The user is then able to choose a deep learning architecture, configure its parameters, and run the classification process. I believe this tool could prove useful for both academic researchers looking to try out and compare their architectures against other architectures, and less research-focused institutions and businesses to solve the hyperspectral classification problem on their own dataset, with architectures provided by the academic community. This application is discussed in Chapter 5.

In Chapter 6, avenues for future work and development on both the proposed multi-stream architecture as well as the web application for HSI classification are provided. Finally, I conclude in Chapter 7, where I look back on what has been achieved in this thesis.

2

An introduction to hyperspectral image classification

As is the case with many research problems, understanding what the problem entails is a very important first step. It requires research in literature as well as practical experimentation. My thesis subject focuses in particular on two subjects: hyperspectral image classification (HSI classification), and capsule neural networks or capsule networks in short.

In Section 2.1 of this overview chapter, I will take a look at some of the frequently used datasets when it comes to evaluating model performance for HSI classification. These datasets are used throughout this thesis, and serve as a useful benchmarking tool for both the multi-scale capsule network architecture I created (cf. Chapter 3) as well as in a more practical setting (cf. Chapter 5).

The HSI classification problem is not new [6] and there is a significant interest of the scientific community towards the problem. Before taking a look at some state-of-the-art supervised learning techniques and their characteristics in Section 2.2, we will look at what the training data looks like and how the spectral information is combined with the spatial neighborhood of the targetted pixel in Section 2.3.

Limiting the amount of labeled training data is very important for applying theoretical research to practical scenarios. The difference between academical and practical datasets is that the latter often have no labeled training samples, which creates a gap between theoretical performance and practical performance. To close this gap, models that use limited training samples have become an important branch of HSI classification research. What this means in practice and how limited training data influences performance is discussed in section 2.4.

2.1 Hyperspectral image datasets

If we take a regular image, for example an aerial view of the fields of a farmer, we distinguish two dimensions. We call these spatial dimensions, since they give an idea of how two pixels (or places) are located with respect to each other. In hyperspectral images, we add a third dimension: the spectral dimension.

In the HSI classification problem, we want to classify each pixel of the image: some pixels may be part of a road, others represent a river or a pond. When we look at a grayscale picture, we might be able to discern a building by its straight edges, a river by curvature, and roads by discovering a network of connected lines with roundabouts and intersections. It might be more difficult to tell a lake apart from a forest, or a field from a large garden.

In colored images, we can remedy some of these uncertainties: rivers, lakes, and ponds can be distinguished by a blue or green color. Forests get a colored texture, fields get different colors depending on the crop that grows on it. In fact, colored images already have 3 spectral bands: red, green, and blue. These spectral bands provide us humans and computer models with more information. It makes classification easier.

Hyperspectral images capture spectral information not just in a wider range of wavelength, but also with a finer spectral resolution. The rich spectral information in HSI allows a better discrimination between different materials. This yields even more information that a computer can interpret, where our human eyes can no longer detect that information. This makes it possible to distinguish different forest types, different crop types, and even how long ago the crops were planted. This is very useful information for a wide range of applications, including but not limited to sea ice monitoring [7], soil burn due to forest fires [8], desertification [9], urban planning [10], soil fertility [11], and many more.

2.1.1 Salinas dataset

As a first example, we take the Salinas hyperspectral dataset. This academical dataset was captured by the AVIRIS sensor [12] mounted on a plane flying over the Salinas valley, California. It contains 224 spectral bands, which capture light frequencies from 400nm to 2500nm. For reference: the human eye can only perceive light frequencies ranging from 380nm to 700nm. The Salinas dataset consists of 204 spectral bands. 20 bands are water absorption bands, and since there are no water bodies in this scene, these were removed from the dataset.

In Figure 2.1a, a grayscale visualization of the middle band (band 102) is shown. It was created by first normalizing the sensor value of the pixels between 0 and 1, which produces this image. This image represents one slice of the data cube, over the spectral dimension. In Figure 2.2, some grayscale representations of different spectral bands are depicted.

If we take a look at the fields in the top right of the image, which are shown almost white in spectral band 75 (Figure 2.2b), we can see that they are clearly different from the rest of the image. It is harder to see differences between those fields, however. To tell them apart, we can use band 20 (Figure 2.2a) which shows a lot more variation between those fields.

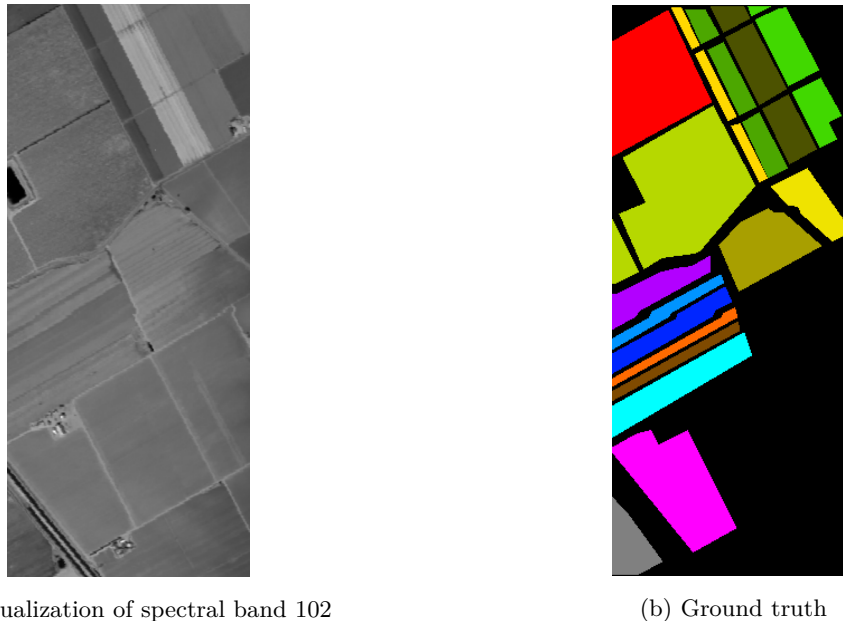


Figure 2.1: Grayscale representation of a single spectral band and the ground truth of the Salinas dataset

The HSI classification problem we are trying to solve attempts to assign a class to each pixel in the dataset. Some examples of the 16 classes from the Salinas dataset are 'Celery', 'Stubble', and 'Fallow'. Also in this image: 4 lettuce fields that were planted 1 week apart, from 4 weeks old to 7 weeks old. Each of those fields have a different class assigned to them.

Due to the Salinas dataset being used for academical purposes, it is labeled. If we color in the class labels that are provided with the dataset, we get the image in Figure 2.1b. Black represents unlabeled classes. The task of the hyperspectral classification model will be to take a subset of these labeled samples and learn from them to classify the remaining labeled pixels to their respective classes.

The resolution of the dataset is provided as well. For hyperspectral images, the resolution is usually given in metres per pixel. The Salinas dataset has a spatial resolution of 3.7 meters, which means that every pixel represents an area of $3.7m * 3.7m = 13.69m^2$. The Salinas dataset consists of $512 * 217$ pixels, making it span an area of $1.52 km^2$.

We now know that a hyperspectral image consists of pixels that each represent an area of an aerial picture. Instead of the usual red, green, and blue color channels, we have up to hundreds of spectral bands. Each band captures the reflected light in a different way, generating n float values, where n is the number of spectral bands. Each pixel can be assigned a label based on the geographical features of that area: examples are forest, a specific type of crop, or a soil type. The HSI classification problem consists of taking the spectral data as input, and providing a class label as output. What we don't know, is what this actual data looks like. This will be demonstrated in the next dataset, the Indian Pines dataset.

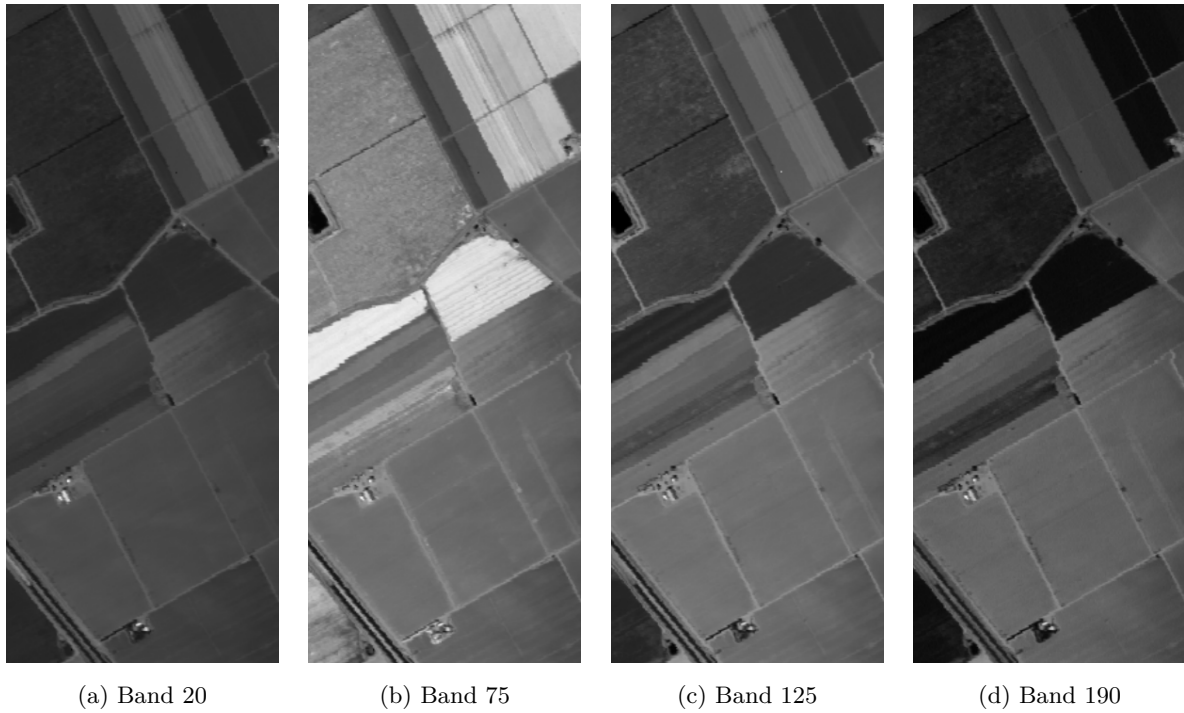


Figure 2.2: Different slices from the Salinas dataset

2.1.2 Indian Pines dataset

The Indian Pines dataset is one of the oldest hyperspectral datasets in existence. It was captured in 1992 by the AVIRIS sensor over one of the Indian Pine Test Sites in Indiana [13]. Similar to the Salinas dataset, 24 water absorption bands were removed bringing the total number of spectral bands to 200. The dataset is square, measuring 145×145 pixels, each with a resolution of $20m^2$. In the ground truth data, there are 16 labeled classes.

In Figure 2.3a, the grayscale visualization of band 120 of the Indian Pines dataset is shown. The spatial resolution of Indian Pines is 20 m/pixel, which is much lower than Salinas. The ground truth map is shown in Figure 2.3b.

In the original dataset, the values of the spectral bands are not normalized. This leads to varying degrees of variation in the spectral bands (Figure 2.4). To inspect the spectral signature of a single pixel (and the 200 spectral values that belong to that pixel), it is necessary to normalize over all spectral bands. The result is shown in Figure 2.5.

2.1.3 Pavia University dataset

A third academic dataset, which is used multiple times to evaluate classification performance throughout this thesis is the Pavia University dataset, over Italy. In contrast to the other two academical datasets, this dataset focuses on urban areas instead of agricultural lands. The grayscale representation of a single spectral band and the ground truth of this dataset can be seen in Figure 2.6a and Figure 2.6b, respectively.

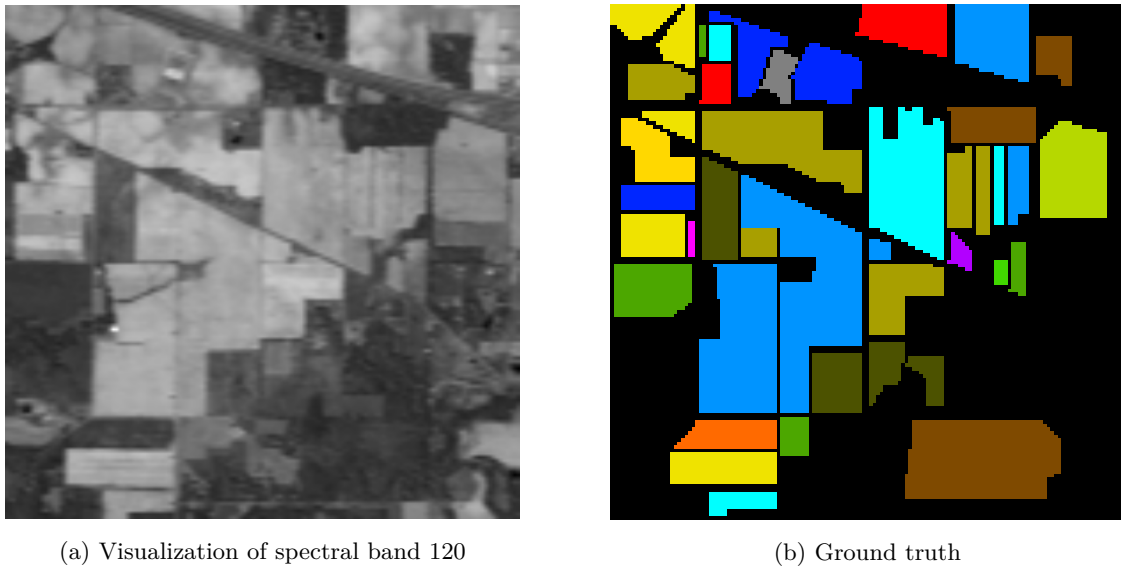


Figure 2.3: Grayscale representation of a single spectral band and the ground truth of the Indian Pines dataset

Immediately it is clear that it differs from the other datasets. There are no longer large areas belonging to the same class, making the local neighborhood of the target pixel more variable.

As a big local neighborhood is essential for achieving high classification accuracy [14], this variable landscape can introduce problems during training. This is offset partially by the relatively high resolution of the Pavia University dataset, where each pixel covers just $1.3m^2$, as opposed to $13.69m^2$ (Salinas dataset), $400m^2$ (Indian Pines dataset), and $900m^2$ (Hyperion-1 sensor for satellites).

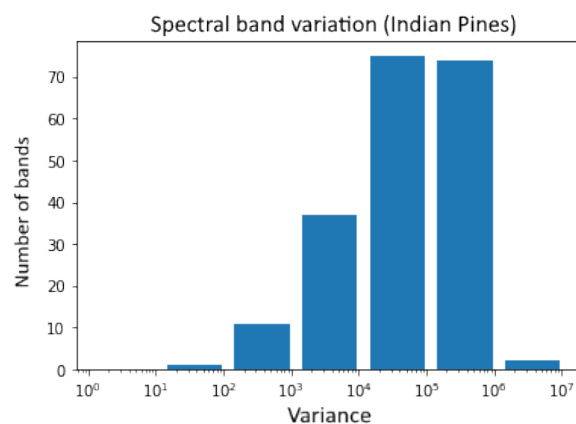


Figure 2.4: Variations in the different spectral bands of the Indian Pines dataset

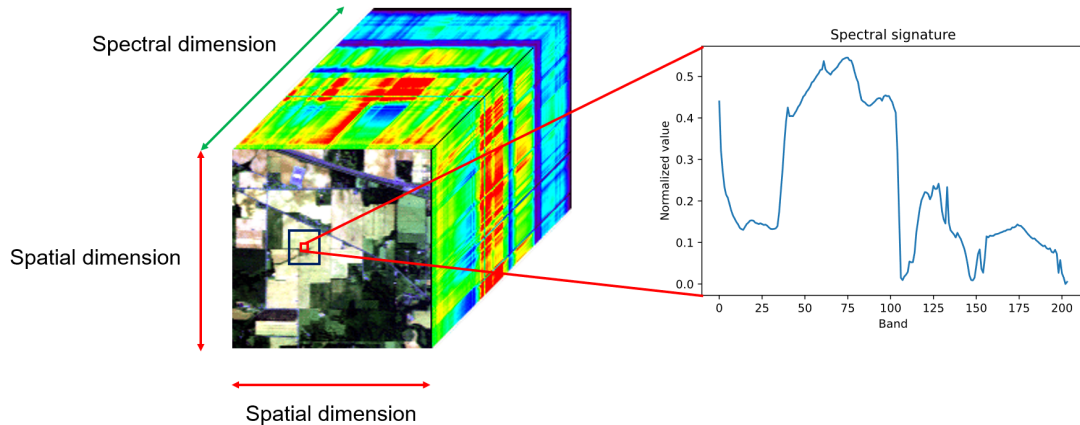


Figure 2.5: Normalized spectral values of a single pixel

2.1.4 An Earth Observation 1 Satellite dataset

So far, we have explored some datasets that were captured specifically because of the challenge they pose with their many different terrain features. In practice however, the datasets are often of lower resolution and consist of fewer classification classes. Additionally, the data is often unlabeled, making supervised learning not as straight-forward anymore. In this section, we explore a satellite image as an example of a dataset that we might encounter in practice.

In Figure 2.7, one of the countless satellite datasets already openly available is shown. It originates from the <https://earthexplorer.usgs.gov/> website, which enables registered users to download hyperspectral images captured by the Earth Observation 1 satellites, using the Hyperion 1 sensor. This particular dataset from 2017 features 242 spectral bands and consists of 3261 x 911 pixels. The spatial resolution is quite low, as each pixel covers $900m^2$.

It features the island of Moroni, off the coast of Mozambique. Due to its poor resolution, making out specific terrain features is difficult if not impossible. We can discern two clearly distinct classes: land and water.

2.2 Spatial neighborhood in training samples

Training any (supervised) deep learning model requires training data. For HSI classification, there are patch-based methods [15], adjacency graphs [16], and even methods that take the entire dataset as one single input [17]. Patch-based methods will be explored in more detail below, as it is used throughout my thesis and lies at the basis of my multi-scale capsule network architecture.

The most simple model for classifying hyperspectral images would take in a feature vector of size n , where n is the number of spectral bands, along with a classification label. The training sample is then essentially the collection of all spectral values belonging to a single pixel. The Indian Pines dataset has 10,069 labeled pixels, meaning that there would be as many training samples. Using a convolutional

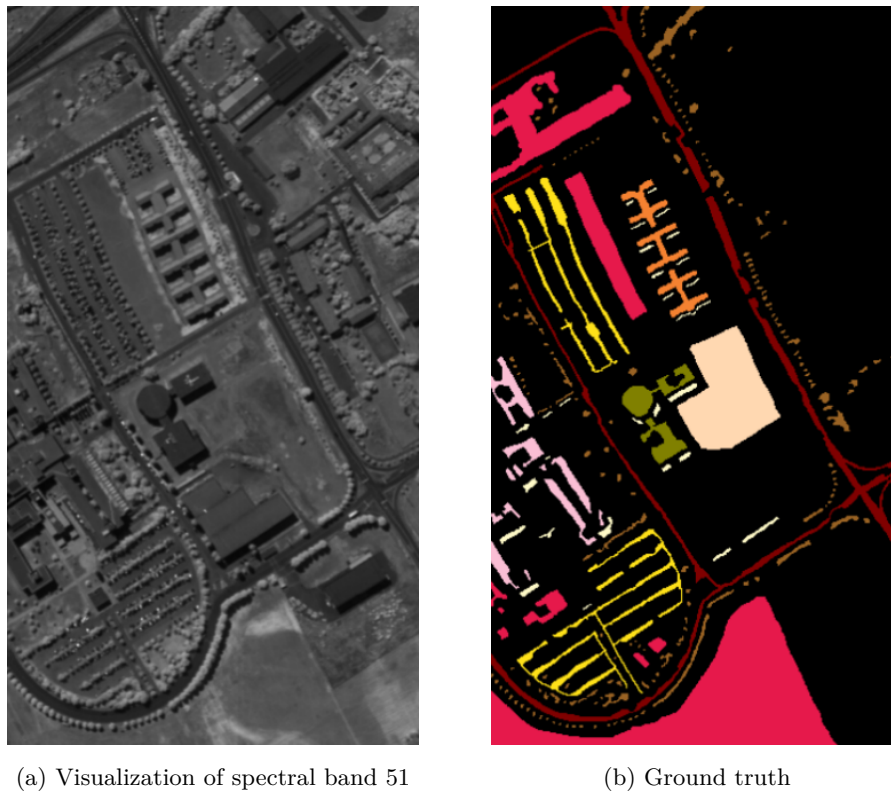


Figure 2.6: Grayscale representation of a single spectral band and the ground truth of the Pavia University dataset

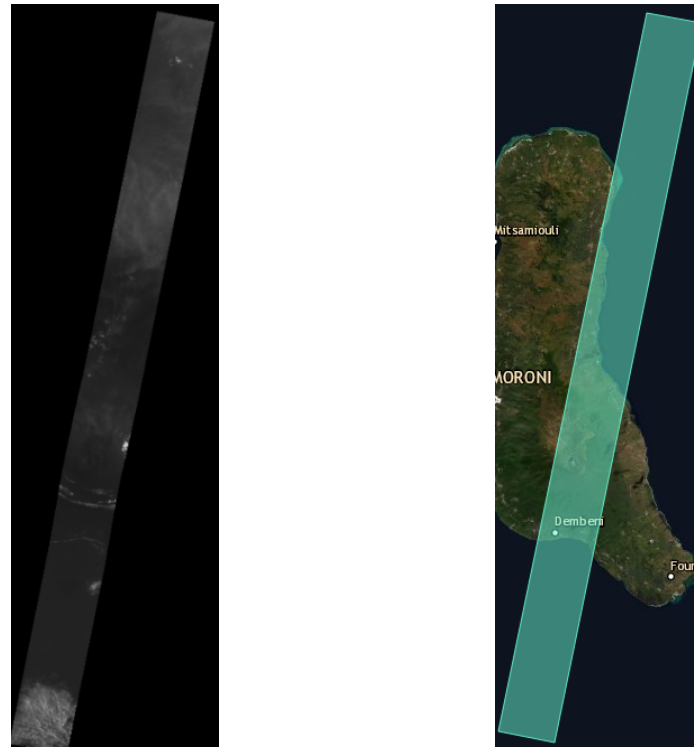
neural network [18] [19], we can train a deep learning model to classify the hyperspectral image.

The models that use only spectral features are called spectral-feature networks. Including the spatial dimension in the training samples improves the classification results [20]. In these spectral-spatial-feature networks, the model looks not only at the values of the sensor for that specific area, but also at the surrounding area. This helps to generalize the model and improves the classification performance.

The size of the neighborhood is an important hyperparameter: if the value is too low, the resulting classification map appears pixelated and has many outliers. If the value is chosen too large, subtle details are missed, which is especially bad if the class area is narrow. An example of the two extremes can be found in Figure 4.1. A possible solution to combine the benefits of both large and small patch sizes is provided in the multi-scale capsule network architecture I created for this thesis, and is explained in Chapter 4.

2.3 Supervised learning

With the number of published papers on hyperspectral images increasing more year after year [20], it comes as no surprise that a lot of techniques have been thrown at the classification problem. In this section, I will outline some key techniques that have been used specifically for a supervised approach.



(a) Visualization of spectral band 15

(b) Map view of the dataset

Figure 2.7: Grayscale representation of a single spectral band and the map view of the EO1 dataset

With recent efforts to take hyperspectral images of the globe, such as the Sentinel-2 satellite mission of ESA [21] launched in 2015 and the Earth Observation 1 mission by NASA [22] launched in 2000, unlabeled hyperspectral image data has become more available. While unsupervised techniques to solve this problem are very interesting and worth researching, they suffer from lackluster performance and lie beyond the scope of the thesis.

1) Convolutional Neural Network

CNN's are some of the most used deep learning techniques in image recognition. Their ability to see patterns was a big leap forward for all kinds of image classification techniques. In HSI classification, we don't focus on a large dataset of different pictures, but rather on one picture. A lightweight CNN (LWCNN) [16] applies a dimensionality reduction technique called Spatial Spectral Shroedinger Eigenmaps (SSSE) to create an adjacency graph from the data cube. If two pixels are in each others spatial neighborhood, the weight of the edge between the two is set to 1, and 0 otherwise.

A kernel that measures the similarity between adjacent pixels further calculates the weight of the edges. This way, feature extraction can be done by calculating the eigenvectors of the matrix. The result is a datacube that has the same spatial dimensions as the original image, with a 3rd dimension consisting of the number of features chosen. The number of features is much higher than the number of spectral bands, which in combination with a limited sample size results in overfitting. To avoid this, distributed source coding is run a few times to aggregate the features.

The way to cope with scarce training data is interesting. The performance results are very good for this architecture, although an extremely small amount of training data was used, which is precisely what LWCNN performs well in. In practical scenarios we can use a little more data, as practical datasets will either have no labeled samples at all or a few hundred pixels assigned to each class.

2) Graph Convolutional Network

Graph Convolutional Networks (GCN) [17] also work with a graph-like structure. The idea now however is to use both labeled and unlabeled data, to create a more robust result. Rather than cutting out patches from the image, the image is provided in its entirety to the graph-based architecture, which calculates the non-local graph over the image.

Because the architecture is not targeting specific pixels, but rather the entire image, the resulting classification map is way smoother than pixel-based classification methods. It yields favorable results, although the computational complexity might prevent this model from being used in practice.

This technique has a good idea: if supervised learning can be applied, it is likely that only a fraction of the data is available as labeled training data. Including unlabeled training samples can not only improve classification performance on the current dataset, it could also be used to cross the gap between two datasets captured by the same sensor: the second dataset might not be labeled at all, and will benefit from the learned classes from the first dataset, aided by their own unlabeled samples to offset noise like clouds. However, this technique suffers from high computational complexity.

3) Deep Domain Adaptation

The idea to re-use learned architectures to other dataset was the starting point for the Deep Domain Adaptation architecture (DDA-Net) [23]. In this architecture, three modules are used to train the network: the domain alignment module, the task allocation module, and the domain adaptation module.

In the domain adaptation module, two branches are used: the source branch (of the original dataset) and the target branch (of the dataset that has no or little labeled training samples). This unsupervised module will learn spatial-spectral features so that the output of the two branches is an equal amount of features. Then, the similarity between any pair of outputs is learned, which results in the similarity coefficient.

These coefficients are then used in the task allocation module which learns the features on the source domain. This module minimizes the similarity loss of the domain alignment module. Finally, the learned classification model is fed to the domain adaptation module which transfers the learned architecture to the target domain.

While very interesting, the paper that accompanies this architecture calculates their performance results by splitting an academical dataset in two: one side is left labeled, and the other one is left unlabeled. While this certainly is a valid technique - and the only one possible on these academical datasets since they are all captured by different sensors - it would be interesting to see this research applied in practice and see how it performs. That being said, the domain of transfer learning is certainly an interesting one for HSI classification.

4) Cross-Scene Deep Transfer Learning

Transfer learning can take many forms: a learned architecture can be applied to unlabeled datasets captured by the same sensor, or by different sensors. The last one creates additional problems, which is exactly what the Multiscale Spectral-Spatial Network (MSSN) [24] attempts to solve. It consists of the spectral feature adaptation module and the MSSN-based deep learning module.

The spectral feature adaptation module is applied first and calculates the distribution of both source and target domains, where the target domain is largely unlabeled. It calculates the projection of the source domain onto the target domain, and applies PCA on both the source and target domain features.

This semi-supervised technique, along with a couple others, works with two separate streams. This concept is what inspired my own proposed architecture in Chapter 4, although most two-stream architectures split spatial and spectral features. In my proposed architecture, the multiple streams consist of spatial and spectral features across different spatial neighborhoods. The technique used in this section approaches the HSI classification problem from a probabilistic point of view. That makes it interesting from a mathematical stance, and it calculates its performance on two different datasets, rather than splitting one dataset up in two halves, which arguably is a more interesting problem statement.

5) Multi-Modal Learning

Just like a height map combined with an RGB-image is data with multiple modalities, hyperspectral images have multiple modalities in that every spectral band captures different light wavelengths. More modalities like height maps can still be introduced, however, which is exactly what the Multimodal Deep Learning Meets Remote Sensing (MDL-RS) [25] framework presents.

Using this principle, grass on a roof can be distinguished from grass in a park, and snowy mountain tops can be distinguished from the flats that lay might next to those peaks, and are also covered in snow. Another motivation for multiple modalities is the problem of cloud coverage in image acquisition, as to make the model more robust to bad weather conditions. MDL-RS extracts hierarchical features, mixes them and fuses them in the fusion network.

With satellites already being able to measure height of the terrain, it makes sense that this data, as well as weather data, is readily included in hyperspectral datasets. Multiple modalities can only improve classification performance. Additionally, the ability of these multiple modalities to provide context for the classification process might prove useful.

6) Multi-Scale Models

Architectures that use multiple streams or scales have proven very impactful on my thesis. It is therefore only right that I include one of them in my discussion of the state of the art. The Multi-Scale Capsule Network uses multiple types of capsules to detect types of features. These "types" are rather abstract, as is often the case with deep learning, and prove useful both for increasing classification performance as well as for decreasing training time. It is worth noting that the accompanying paper [26] experiments with the FashionMNIST and CIFAR10 image datasets, rather than hyperspectral datasets.

The idea of multiple scales has spoken to me from the start of my thesis, and has proven very helpful in creating my own proposed architecture. As these multiple streams provide a 'different look' on the

data, it stands to reason that they might give additional information, therefore increasing classification performance.

2.4 Limitations of theoretical models

There are a couple of problems that arise when applying theoretical models to real, unlabeled datasets. These problems are the challenges I had to overcome when building my application. Solutions for these problems are proposed in the application chapter, while the problems themselves are discussed below.

A major problem has already been discussed: the lack of unlabeled training samples. If supervised learning is to be applied to these practical datasets, a way of labeling at least some pixels will need to be devised. No matter how carefully this is done, we won't have the quality of labeled samples that we did with the academical datasets. After all, the ground truth of the academical datasets is seen as the ideal solution for the problem.

The training samples in many of the papers on HSI classification are picked at random, making the training data very robust. Pixels on the edge of two fields, as well as pixels of the same class but from different parts of the image are much more likely to be included in the training samples, even if the number of training samples is purposefully kept low. If all pixels come from a small, labeled area however, the model might have to discover unlabeled areas of the image with the same features. This is something that is often not focused on in academical research currently.

With large datasets, like those taken by satellites, the training time often becomes way smaller compared to the classification stage. This is strong motivation to keep the number of parameters low. Additionally, the patch size must remain reasonably small. If an architecture has to use patch sizes of 27×27 [27], the number of input parameters quickly spins out of control. This can be remedied by applying PCA, among other techniques. Additionally, it makes it harder to classify large batches of pixels, as the memory footprint experiences a quadratic growth. Luckily, this problem can be overcome to an extent since distributed computing is highly applicable to the classification stage.

Resolution is a problem as well. A very interesting research topic would be to generate a road network based on hyperspectral information. While this might be possible with the Salinas dataset, for example, where each pixel represents an area of $13.69m^2$, this is impossible to do with satellite hyperspectral images, which still have a resolution of $900m^2$ (EO-1 Hyperion [22]) or more.

Finally, it would be interesting if we could re-use the same model for different datasets. This might work for some datasets, but not for others. Especially with two different sensors or a different number of spectral bands, different models will need to be trained. A model that is trained on dataset A, and applied to dataset B which has a geographical class that is not present in dataset A, might deliver unpredictable results. The field of transfer learning [28] [23] is a very important research direction for HSI classification, where much work is still to be done.

A criticism on hyperspectral image classification research is that it attempts to keep improve results on academical datasets that are already very high. Research resources could better be invested into doing something with the research, applying it to real problems with real, unlabeled datasets. In Chapter 5

of this thesis, I discuss the web application I developed, which allows the user to load in hyperspectral datasets and classify them using an architecture of their choosing. The application's precise goal as well as its features are discussed in detail in Chapter 5.

3

Exploring capsule networks

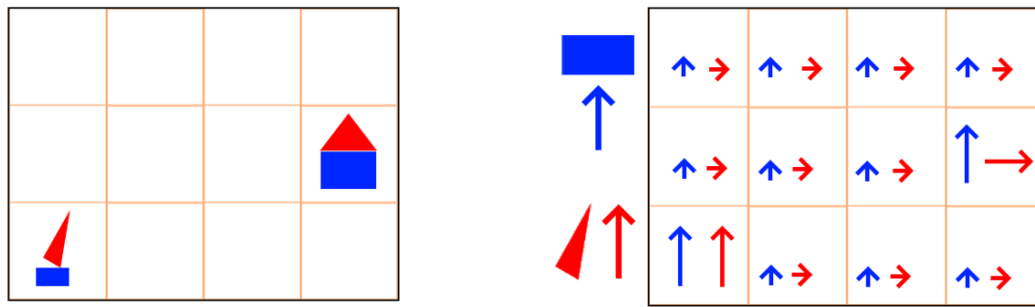
In this chapter, I will focus on a particular deep learning technique: capsule neural networks, or capsule networks for short. This technique is closely related to convolutional neural networks, which has been around for a lot longer than capsule networks. While I have expanded the thesis with an application (see Chapter 5) which supports multiple architectures, capsule networks are the basis for a new architecture I am proposing, which incorporates multi-scale spatial information by developing a multi-stream based architecture.

3.1 Introducing capsule neural networks

In the first section, the origin and the idea behind capsule networks is outlined. The idea of instantiating vectors, which is one of the core ideas behind capsule networks, is quite abstract. To make the concept more approachable for the reader, I provide a toy example.

In Section 3.1.2, we dive deeper into the architectural properties of a capsule network based architecture. Using primary and secondary capsule layers, the basis for these architectures is provided. Two concepts of capsule networks deserve special attention, and are explained in Section 3.1.3. These two concepts, routing by agreement and the squashing function are respectively algorithmic and mathematical solutions used by capsule network architectures.

In the final Section 3.1.4, we combine the concepts of the capsule network architecture into a simple model. Inspiration for this model was gained from a paper by Li et al. [29], and deserves special attention since it provided me with essential knowledge about capsule networks and hyperspectral image classification



(a) A raster with a boat and a house

(b) Instance vectors for rectangles and triangles

Figure 3.1: Instance vectors: a core idea of capsule networks for hierarchical relationships

early on in my thesis.

3.1.1 Origin & idea

The idea behind capsule networks is to add structures to convolutional neural networks in an attempt to better model hierarchical relationships. The idea for this algorithm comes from neuroscience research [30]. In 2000, Hinton proposed a technique for image segmentation and recognition [31] for the MNIST dataset using parse trees [32]. In 2017, Hinton’s paper that sparked interest into capsule networks was published [33].

Despite its young age, capsule networks have been used with success on various problems. New architectures are often tested on the MNIST dataset, where it was shown that it could compete with state-of-the-art deep classification methods [34] [35]. Without comparing itself to other architectures, a multi-lane capsule network architecture [36] managed to classify the Fashion-MNIST dataset [37] and reconstruct the training samples using the decoder network. In a Kaggle competition about identifying aggression and toxicity in comments, capsule networks performed as well as other deep learning techniques [38].

3.1.2 Primary and secondary capsule layers

Pooling layers in CNN are used to introduce rotational invariance with respect to small rotations and translations, and for dimensionality reduction. This can be useful in some cases: a traffic cone that has been thrown over should still be recognized as a traffic cone. In other cases, this is less applicable: a face consists of two eyes, a nose, a mouth, two ears, but the positions of these individual parts with respect to each other matters too. To visualize this, we can look at a simple example. In Figure 3.1a we can see a raster with a boat and a house in two of the raster squares.

The capsule architecture in its simplest form consists of a primary layer and a secondary layer. The first layer will recognize shapes and patterns. In the example of our face, the facial features would be

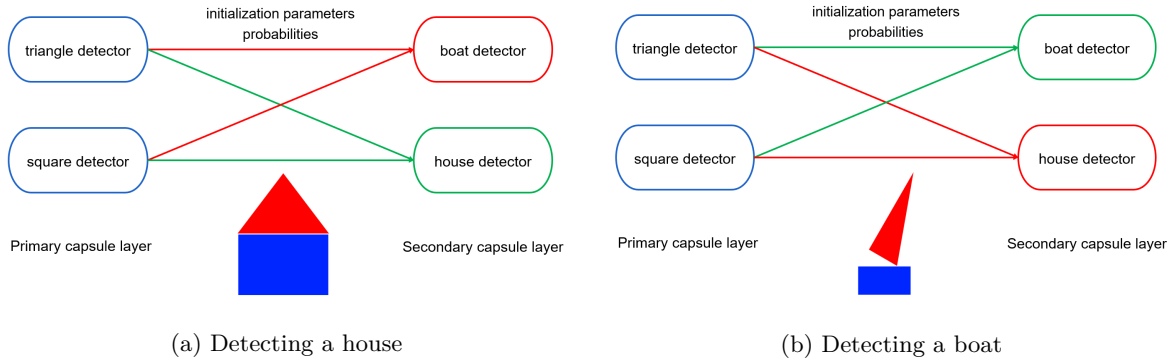


Figure 3.2: Routing by agreement: detecting objects from shapes

recognized by the first layer. Rather than outputting a value that represents whether a shape is present in the current window or not, the primary capsule network layer outputs a vector. The length of this vector is the probability that this shape is present in the current window, while the direction and sense of the vector indicate the instantiating parameters of the shape.

Some examples of these instantiating parameters are rotation and size for shapes, or hue for color. While these are parameters that we can understand easily, in reality these are learned from the data, and might not be so straight-forward. The primary layer is thus already a black box, which might warrant further research around explainability [39].

A second capsule network combines these outputted vectors, and decides whether the specific combination of shapes with respective instance vectors should be detected as a particular object. In our simple example, where the second capsule layer delivers the outputs for classification, the number of capsules in the secondary capsule layer is equal to the number of classes. It should be noted that the output of the secondary capsule layer is again a set of vectors: each with the probability that the object is detected in the current window, and instantiating parameters of the object. In Figure 3.2 we can see this combination process in action. This process is called routing by agreement and is a cornerstone of capsule networks.

3.1.3 Routing by agreement & the squashing function

The process of routing by agreement is shown in more detail in Figure 3.3. Pseudo code for the algorithm is shown in Listing 1. This routing algorithm runs for r iterations, where r is the routing parameter. It calculates the parameters b_{ij} , which are initialized as the log prior probabilities that capsule i from the previous layer l should be coupled to capsule j on layer $l + 1$. These parameters b_{ij} are called coupling coefficients. In our simple example in Section 3.1.2, the primary layer is layer 0 and the secondary layer is layer 1.

The initial coupling coefficients are refined in the iterative routing process by measuring the agreement $a_{ij} = v_j \cdot \hat{u}_{i,j}$, with $\hat{u}_{i,j}$ the prediction vectors indicating the prediction made by capsule i based on the outputs of capsule j .

Routing by agreement

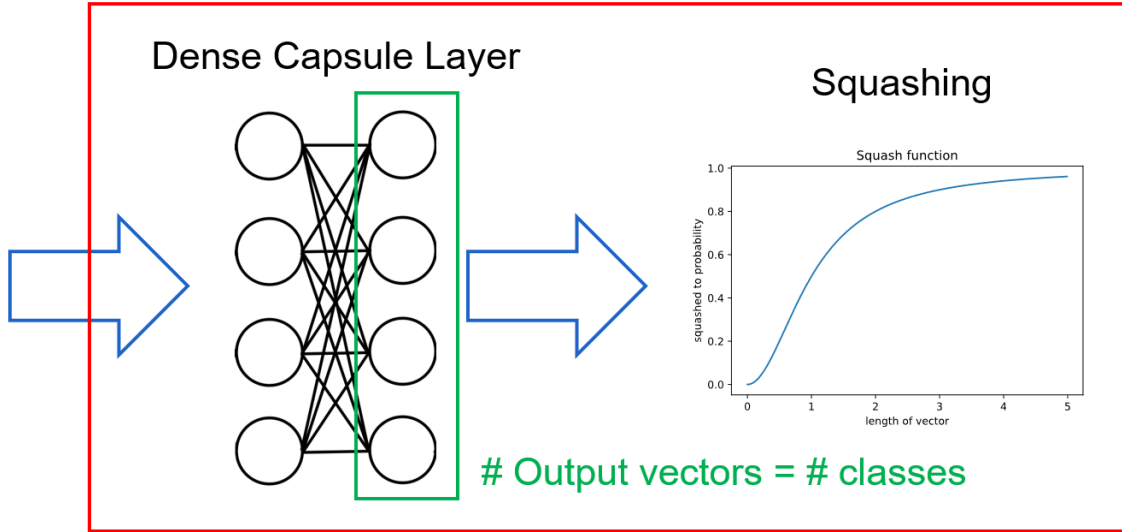


Figure 3.3: Routing by agreement: combining shapes into objects

```

function routing( $\hat{u}_{j,i}$ ,  $r$ ,  $l$ )
  for all capsules  $i$  in layer  $l$  and all capsules  $j$  in layer  $l+1$ 
     $b_{ij} = 0$ 
  loop  $r$  times
    for all capsules  $i$  in layer  $l$ 
       $c_i = \text{softmax}(b_i)$ 

    for all capsules  $j$  in layer  $l+1$ 
       $s_j = \sum_i c_{ij} \hat{u}_{j,i}$ 
       $v_j = \text{squash}(s_j)$ 

    for all capsules  $i$  in layer  $l$  and all capsules  $j$  in layer  $l+1$ 
       $b_{ij} = b_{ij} + \hat{u}_{j,i} \cdot v_j$ 

```

Listing 1: Routing by agreement in pseudo code

Before the coupling coefficients are refined, the inputs of capsule j are squashed. The squashing function is a non-linear function to ensure that short vectors get shrunk to almost zero length while long vectors are mapped to values close to 1. The squashing function is shown in Formula 3.1 and plotted in Figure 3.4.

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \cdot \frac{s_j}{\|s_j\|} \quad (3.1)$$

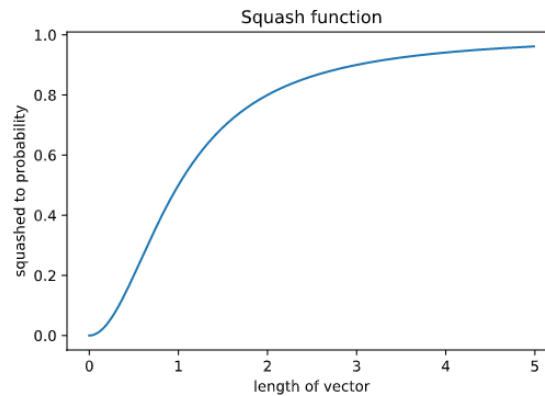


Figure 3.4: The squash function used in routing by agreement

3.1.4 A simple architecture

A simple architecture is shown in Figure 3.5. In the example, the Indian Pines dataset is taken as input for the model. Patches are cut out around the labeled training samples, as can be seen in Figure 2.5. In this Figure, the training sample subject is the red square, while its neighborhood (the patch) is shown in dark blue. The spectral signature shown is that of the target pixel, the spectral signature of all pixels in the neighborhood is also part of the input data.

The first operation is a regular convolutional layer. We will not go into the details of the parameters that should be used for this layer, these will be discussed in Section 3.3. The convolutional layer introduces extra features by applying different filters to the input. Since the input is 3D, this is a 3D-convolutional layer.

Next up is the primary capsule layer. This layer will detect the 'shapes' in our data. Because we are no longer talking about our simple example with the boat and the house, this is less intuitive. The output of this primary capsule layer is then squashed to prepare it for the routing by agreement algorithm. The routing algorithm will decide how capsules from the primary capsule layer are connected to the secondary capsule layer. The number of capsules in this primary layer is a hyperparameter.

After routing by agreement, the secondary capsule layer outputs a vector based on the input that was provided by the primary capsule layer. As a reminder, the length of the vector indicates the probability that the class X is predicted for the target input. While the number of primary capsules can be chosen freely, this last capsule layer has n capsules, with n the number of classes in the dataset.

To make the output interpretable, we apply the squashing function again to the outputs of the secondary capsule layer, which produces vector lengths in the interval $[0, 1]$. Optionally, a decoder network is added. This will take the output of the secondary capsule layer and attempt to reconstruct the input. This sometimes makes the coupling coefficients more robust.

In a practical setting, we would choose the output with the largest length as the predicted class.

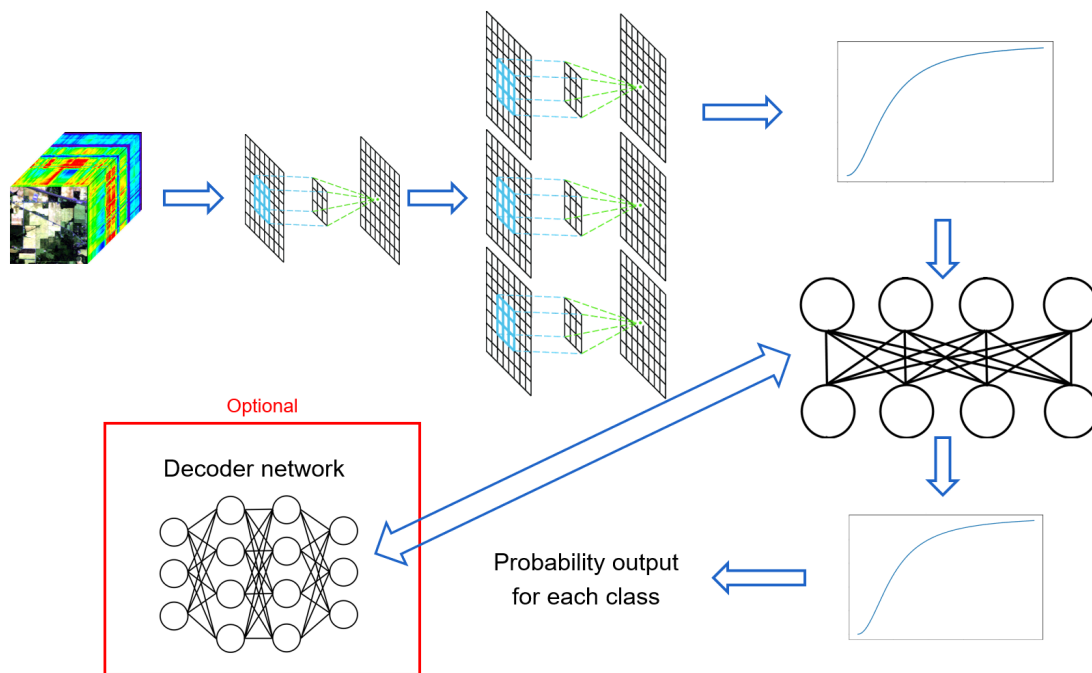


Figure 3.5: A simple capsule network architecture

3.2 Relevance to hyperspectral image classification

In hyperspectral image classification, both spatial and spectral domain are essential to provide good classification results. Capsule networks can be an improvement on convolutional neural networks, as these networks perform poorly due to their lack of spatial awareness [15]. This can be remedied to a degree by reducing the spatial dimension using pooling layers, although this comes at the cost of information loss. Additionally, CNN fail to detect equivariences in training samples that lie far away from each other, and are negatively influenced by rotations in the features. By representing features as vectors instead of scalars, capsule networks are able to detect rotations, and discover hierarchical relationships between features.

HSI classification can be performed in one dimension, two dimensions, or all three dimensions [40]. In 1 dimension, the spatial dimension is ignored, and the spectral features of the training sample are used for the classification process. The solution in 2 dimensions remedies this by focusing on the spatial information. In these models, PCA is often used to extract the spectral bands that deliver the most information. From these spectral planes, patches are cut out around the target pixel and are fed into the 2D architecture. Using a 2D architecture instead of a 1D architecture can give an immediate improvement of 9 and 10 percent for CNN and CapsNet respectively [41].

Finally, HSI classification can be done in three dimensions. These methods extract small overlapping 3D patches around the target pixel, and feeding it to a 3D classification architecture. This introduces the curse of dimensionality, a term coined by Richard Bellman in 1966 [42]. Where the introduction of the spectral domain in the feature space is considered essential for HSI classification, the shift towards 3-dimensional feature space can cause a lot of overfitting. Still, 3D CNN networks [43] and 3D capsule

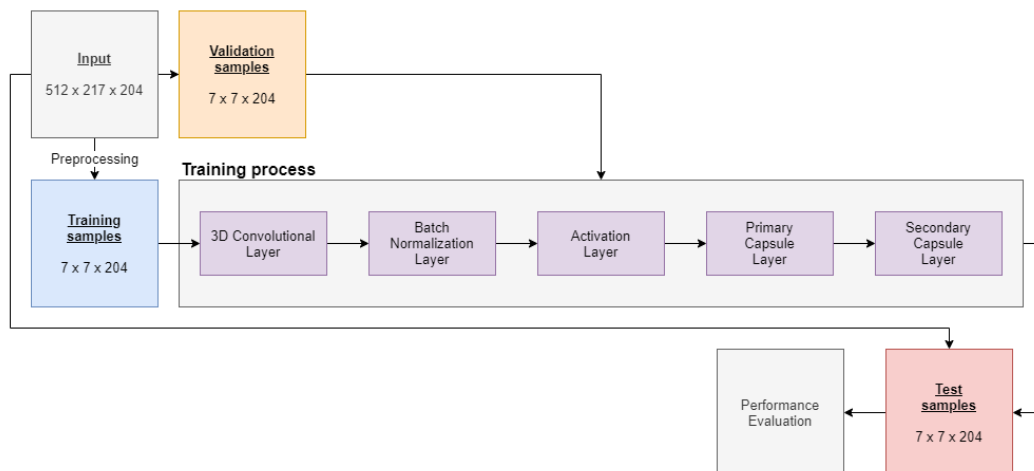


Figure 3.6: A simple single-stream capsule network architecture

network architectures [27] exist. In the architecture that I propose in Chapter 4, I will be exploring a multi-stream 3D capsule network architecture that incorporates multi-scale spatial information to classify hyperspectral images.

3.3 Experiments

In this section, I provide experimental results of a very simple, single stream capsule network for the hyperspectral image classification problem for 3 datasets: the Indian Pines dataset, the Salinas dataset, and the Pavia University dataset. All experiments ran for 50 epochs and used the ground truth as input, with 100 samples per class selected randomly: 50 for training and the other 50 for validation. If a class contains less than 100 labeled samples, it was split equally over the two sets, and were upsampled to 50 through picking random samples multiple times. The architecture can be viewed schematically in Figure 3.6

The performance of these models serves as a baseline for what is possible with capsule networks. The hyperparameters for all models were determined after an extensive parameter search, where the best performance was searched for one to three hyperparameters at a time. The hyperparameters for the single-stream architecture are listed below. In total, there are 212,388 trainable parameters in this architecture.

- **Patch radius:** the spatial radius of each input patch. A patch radius of 3 means that the input patch has the following shape: $(7, 7, 204)$, with the third dimension being the spectral dimension.
- **Batch size:** fixed to 10 for all models
- **Learning rate:** 0.001 for all models
- **Learning rate decay:** 0.9 for all models

- **3D convolutional layer:** 64 filters, a kernel with shape (3, 3, 5) and a stride of 1 over all dimensions. The padding is set to 'valid'
- **Primary capsule layer:** 12 filters, a kernel with shape (3, 3, 3), an output capsule dimension of 2, and a stride of 1 over all dimensions. The padding is set to 'valid'
- **Secondary capsule layer:** the number of secondary capsules is equal to the number of classes in the ground truth data, with output vectors consisting of 2 scalars. The number of channels was found to be 6. A kernel with shape (3, 3, 3) is used with a stride of 1 over all dimensions, with padding set to 'valid'

Dataset	Indian Pines	Salinas	Pavia University
OA (%)	65.70	86.79	90.87
AA (%)	81.19	92.73	89.71
κ (x 100)	60.94	85.29	87.91
Class			
1	100.0	99.83	86.51
2	40.81	99.92	98.05
3	55.59	70.90	84.56
4	84.67	100.0	97.23
5	93.77	98.72	100.0
6	97.16	99.87	80.83
7	100.0	99.71	91.54
8	96.56	68.74	70.46
9	100.0	97.25	98.23
10	59.40	91.49	-
11	54.41	97.33	-
12	52.86	95.15	-
13	100.0	99.62	-
14	89.27	98.51	-
15	74.30	68.48	-
16	100.0	98.21	-

Table 3.1: Single stream capsule network performance with regards to overall accuracy (OA), average accuracy (AA) and Cohen’s kappa coefficient (κ)

3.4 Limitations

As can be seen in the classification maps (Figure 3.7) and in the performance overview (table 3.1), there is much room for improvement. For a full comparison between this simple architecture, the proposed multi-stream architecture, and other state-of-the-art models, I refer to the 'results' section of Chapter 4.

To see the effect of the neighborhood size, I ran the single-stream architecture again on the Salinas dataset. In Figure 4.1a, you can see the result for a model that only uses the spectral values of the



(a) Indian Pines - classification map



(b) Indian pines - ground truth



(c) Salinas - classification map



(d) Salinas - ground truth



(e) Pavia University - classification map



(f) Pavia University - ground truth

Figure 3.7: Classification maps for single-stream capsule network architecture

target pixel and the 8 surrounding pixels. It has an overall accuracy of 80.02% and an average accuracy of 89.68%. In Figure 4.1b, you can see the same dataset, now trained with a neighborhood radius of 6, where each training sample therefore has a shape of (13, 13) around the target pixel. It has an overall accuracy of 87.93% and an average accuracy of 94.58%.

With only 212,388 trainable parameters, the training process is very fast. For the Indian Pines and Salinas datasets, that have 16 labeled classes each, the training time per epoch is 5.6 seconds for all 800 training samples. Inferencing the entire dataset however, has the same time complexity as training 1 epoch. This makes that the time it takes to calculate the performance and create a classification map is often much higher than training the model.

4

Proposed architecture: a multi-stream capsule network

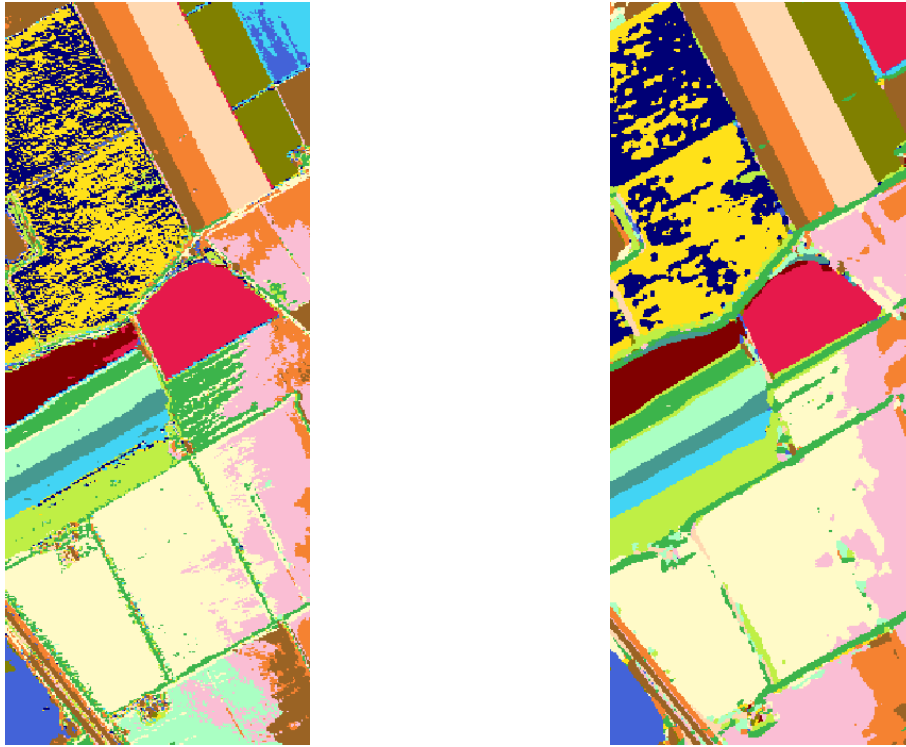
In this chapter, I propose a new architecture for the hyperspectral image classification problem. In Section 4.1, I will discuss the motivation for this new architecture, and how it differs from the simple single-stream architecture used for my early experiments discussed in Section 3.3.

In Section 4.2 the schematic representation of the architecture is explained, which is followed by experimental results for both datasets with ground truth as without in Section 4.3.

4.1 Motivation

Research has shown that the spatial neighborhood of the target pixel that we are trying to classify is extremely important [14]. Without the spatial neighborhood, there is often too little training data for a robust model, which leads to overfitting on pixels that have outliers in their spectral signature. When the spatial neighborhood is included in the training sample, the spectral values of the target pixel are smoothed by mixing its value with those of its neighbours. We know this mixing process as a convolutional layer. This layer has 3 dimensions: (x, y, s) with x and y the length and width of the neighborhood region, respectively. These are hyperparameters, meaning that their value is chosen by the user. The parameter s denotes the number of spectral bands, and is typically equal to the number of spectral bands in the original dataset. In this 3D bar (typically a cube) the center pixel is the pixel which we are trying to classify.

In Figure 4.1, we can see the effect of a large neighborhood versus a small neighborhood. On the left, a small neighborhood (3x3) leads to a pixelated classification result, while a larger neighborhood



(a) Result of training with 3x3 neighborhood

(b) Result of training with 13x13 neighborhood

Figure 4.1: Impact of neighborhood size on classifying the Salinas dataset

(13x13) leads to smoother edges and less noise inside a clearly distinguishable class. The benefit of small neighborhoods is that the preprocessing stage takes less time, and very narrow classes are easily detected. The results are not robust, however, as classification errors are more prone due to spectral outliers. This is resolved partially by increasing the neighborhood size, and we can see this in the overall accuracy of the two models. For the small neighborhood, the overall and average accuracy are 80.02% and 89.68% respectively, compared to 87.93% and 94.58% respectively for the large neighborhood. The downside of large neighborhood sizes are longer training and inference times, and a longer preprocessing stage. When the neighborhood is chosen too large, narrow classes are not detected.

Combining the ability to detect narrow classes with the ability to generalize better over different training samples should deliver a better model. This is what I propose in the architecture that I created. The term "multi-stream" or "multi-scale" networks is not new, and has been used in several papers about hyperspectral image classification [44] [45] [46], and has even been used with capsule networks for image recognition in MSCaps [47]. My proposed architecture differs from the first two in that it treats spatial and spectral features together instead of separately, and that it takes different patch sizes as input for the different streams.

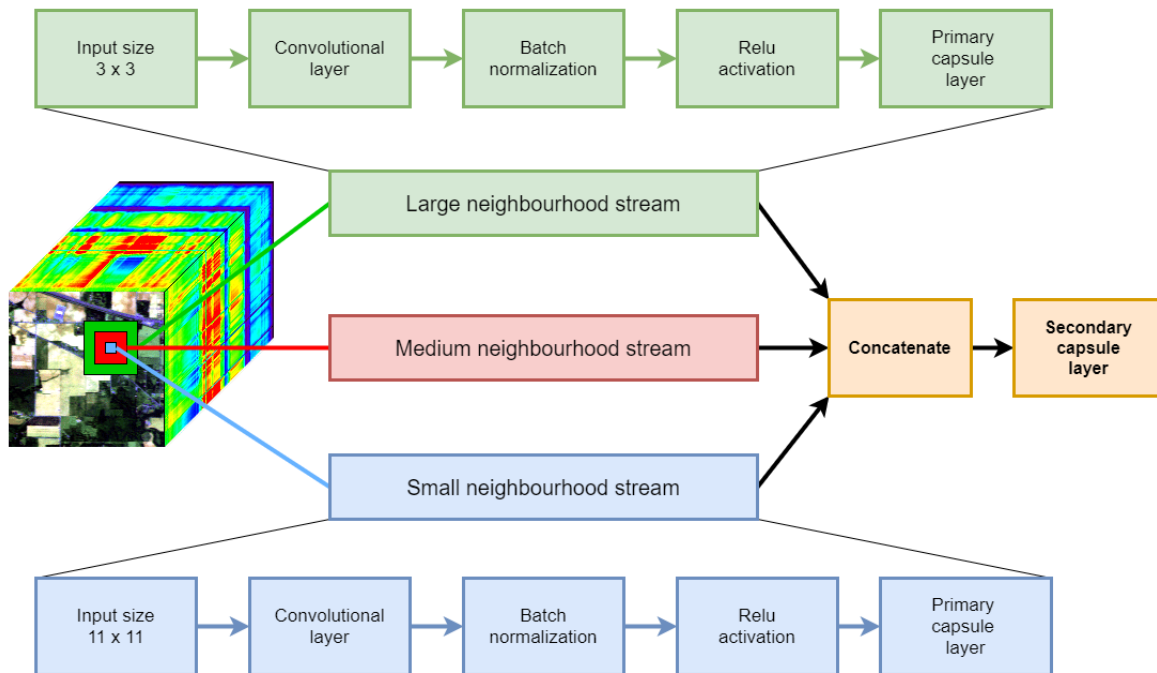


Figure 4.2: Schematic overview of the proposed multi-stream architecture

4.2 Architecture

In our simple capsule network model for hyperspectral image classification, the input training sample consists of the target pixel and its neighborhood pixels. All spectral bands of all pixels in the training sample are included, making the training sample a subcube. The number of pixels to include in the training sample is a hyperparameter, determined by the radius of the neighborhood. A radius of for example 5 would lead to a training sample with dimension $(11, 11, s)$ with s the number of spectral bands in the hyperspectral image.

In the proposed multi-stream architecture shown in Figure 4.2, we recognize the architecture of the single-stream architecture. The first stage of the training process, however, up until the primary capsule layer, is replicated three times, for three different patch sizes. The training sample is therefore exactly the same as in the single-stream model, with a neighborhood radius equal to the radius of the large neighborhood stream.

The number of streams is a hyperparameter, as is the neighborhood radius for each stream. Inside of each stream, the input data is fed into a convolutional layer with a number of filters, a kernel size and a certain stride for the convolutional window. These are all hyperparameters. Through experimentation it is clear that increasing the convolutional window size over the spectral dimension benefits the classification performance. Increasing the stride over the spectral dimension does not lead to immediate performance loss and should therefore be as large as possible without decreasing classification performance to shrink the number of parameters. Increasing the window size over the spatial dimension beyond 2 proves to have a negative effect on performance, as does increasing the stride over the spatial dimensions.

The output of this 3D convolutional layer is normalized, and the rectified linear unit (ReLU) function is used to activate features with high outputs. These are required steps in the capsule network architecture, before feeding the data to the primary capsule layer. The primary capsule layer is essentially another 3D convolutional layer, with the difference that the output data is now reshaped into vectors, which we call capsules. The hyperparameters for the convolutional part follow the same principles of the input convolutional layer, with the exception of the number of filters: this parameter is called the number of channels of the capsule layer, and indicates the number of capsule types. While the number of filters in the input convolutional layer can be chosen to be 64 or 128 or even more, the number of channels in the primary capsule layer is best kept low to avoid a large increase in model parameters. 6 or 12 was found to be sufficient to deliver a good classification result. The reshape layer transforms the set of scalar features into vectors. Increasing the dimension of these capsules (the number of scalars in each output vector) beyond 2 delivers no increase in classification accuracy and heavily influences computation time.

The output of the different streams are vectors, and differ in only 1 dimension. This dimension denotes the number of feature vectors, and the number of features depends on the chosen hyperparameters of the two 3D-convolutional layers. I found that choosing the hyperparameters such that all layers have an approximately equal amount of features delivers optimal performance. Because they only differ in one dimension, these vector features can be concatenated into a single feature vector set. This set is fed into the secondary capsule layer, which will output the predicted class for the input sample.

The secondary capsule layer consists of n capsules, which are activated or deactivated based on the outputs of the input feature vectors. In this layer, n denotes the number of classes that supervised learning is applied to, and will therefore output n vectors of dimension d whose length determine the likelihood that the sample belongs to that particular class. The dimension parameter d is not necessarily the same as the dimension of the outputted feature vectors of the primary capsule layer, but I have found that increasing this parameter beyond 2 also negatively influences computation time, without increasing classification accuracy. The routing algorithm shown in Listing 1 is run a times, where a is a hyperparameter.

The final and most important hyperparameters are the number of streams, and the neighborhood radii for each of the streams. I have found 3 streams to be a good compromise between maximizing classification accuracy while keeping computational time low. While experimenting, I evaluated all possible combinations of radius sizes 1, 3, 5, 7, 9, and 11 for an architecture with 3 streams, where no two streams have the same neighborhood size. In Figure 4.3, the effect of increasing the spatial neighborhoods of the streams can be seen. As 3D data is hard to visualize, I opted to add the dimensions of the three streams and order them from small to large. It is clear that the smallest model, with neighborhood sizes 3x3, 7x7, and 11x11 for the three streams, performs the best.

4.3 Results

In this section, I will compare the single-stream architecture with my proposed multi-stream architecture. The datasets that were used in this comparison are the Salinas dataset (Section 2.1.1), the Indian Pines dataset (Section 2.1.2), and the Pavia University dataset (Section 2.1.3). For all results, the hyperparameters of their respective architectures were tuned to deliver optimal performance. The number of training samples is the same for models on the same dataset.

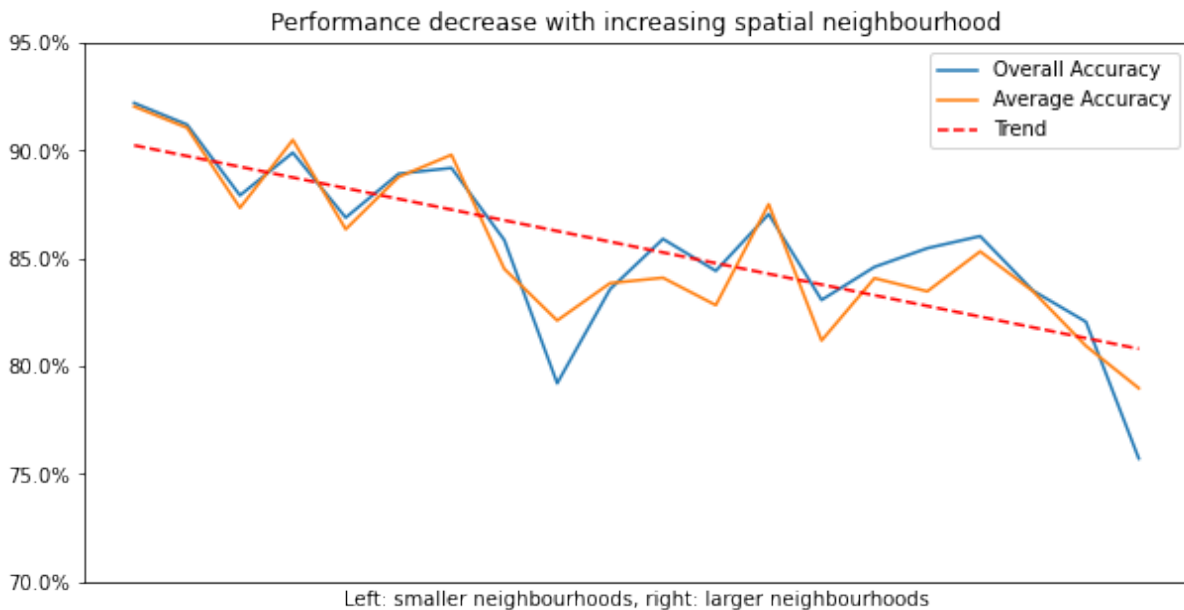


Figure 4.3: Influence of increasing neighborhood of the multi-stream architecture on the classification performance

4.3.1 Salinas dataset

For the Salinas dataset, 50 training samples and 50 validation samples were randomly chosen for each of the 16 labeled classes in the ground truth data, or about 1.5% of the labeled pixels. For the single-stream architecture, a neighborhood size of 7×7 was used. For the multi-stream architecture, 3 streams with neighborhood sizes 3×3 , 7×7 , and 11×11 were chosen.

The overall accuracy on the Salinas dataset increases from 83.44% to 87.15% when transitioning from the single-stream architecture to the multi-stream architecture. Similarly, the average accuracy increases from 92.08% to 94.51%. We see that although classification performance is better, the architecture still is not able to distinguish classes 8 and 15.

4.3.2 Indian Pines

The Indian Pines dataset suffers from a poor spatial resolution, which makes classification more difficult. For this dataset, again 50 training samples and 50 validation samples were randomly chosen for each of the 16 labeled classes in the ground truth data, or about 7.8% of the labeled pixels. For the single-stream architecture, a neighborhood size of 7×7 was used. For the multi-stream architecture, 3 streams with neighborhood sizes 3×3 , 7×7 , 11×11 were chosen.

The overall accuracy on the Indian Pines dataset increases from 65.70% to 70.52% when transitioning from the single-stream architecture to the multi-stream architecture. Similarly, the average accuracy increases from 81.19% to 84.02%.

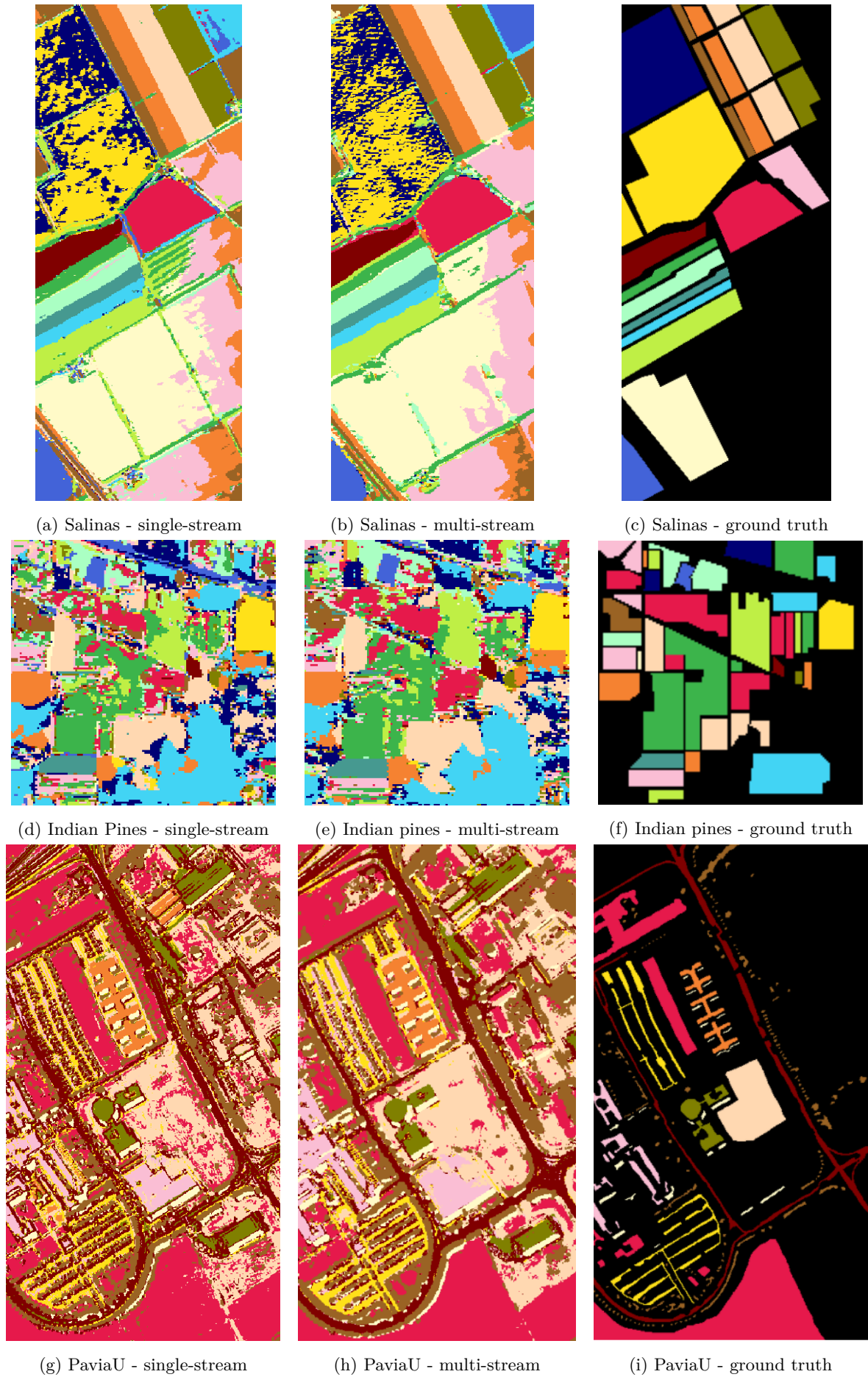


Figure 4.4: Comparing single-stream and multi-stream classification maps

Architecture	Single-stream	Multi-stream
OA (%)	83.44	87.15
AA (%)	92.08	94.51
$\kappa \times 100$	83.88	86.42
1	98.68	99.32
2	99.06	99.97
3	85.93	94.76
4	100.0	100.0
5	99.46	99.34
6	100.0	99.97
7	99.77	99.53
8	54.56	67.19
9	97.78	97.24
10	89.81	92.09
11	96.16	99.34
12	94.64	100.0
13	99.50	100.0
14	97.77	99.78
15	63.47	64.98
16	96.77	98.59

Table 4.1: Proposed architecture performance comparison on the Salinas dataset with regards to overall accuracy (OA), average accuracy (AA) and Cohen’s kappa statistic (κ)

4.3.3 Pavia University

For the Pavia University dataset, 150 training samples and 150 validation samples were randomly chosen for each of the 9 labeled classes in the ground truth data, or about 3.2% of the labeled pixels. For the single-stream architecture, a neighborhood size of 7x7 was used. For the multi-stream architecture, 3 streams with neighborhood sizes 3x3, 7x7, and 11x11 were chosen.

The overall accuracy on the Pavia University dataset increases from 97.31% to 98.25% when transitioning from the single-stream architecture to the multi-stream architecture. Due to the classification performance already being quite high, a 1% increase is certainly no insignificant result. Similarly, the average accuracy increases from 96.91% to 98.13%.

4.4 Comparison with state of the art

Now that the single-stream and multi-stream variant are compared, their performance can be compared to various state-of-the-art architectures. The performance metrics and classification maps shown in tables 4.4 and 4.5 originate from the paper ”Deep Feature Fusion via Two-Stream Convolutional Neural Network for Hyperspectral Image Classification” [44], and are used with permission of the authors. All architectures were trained using 50 labeled training samples, 10% of which were randomly chosen for

Architecture	Single-stream	Multi-stream
OA (%)	65.70	70.52
AA (%)	81.19	84.02
$\kappa \times 100$	60.94	66.22
1	100.0	100.0
2	40.81	52.33
3	55.59	60.51
4	84.69	82.35
5	93.77	96.67
6	97.16	97.35
7	100.0	100.0
8	96.56	97.35
9	100.0	100.0
10	59.40	70.39
11	54.41	58.71
12	52.86	57.27
13	100.0	100.0
14	89.27	88.76
15	74.30	86.67
16	100.0	100.0

Table 4.2: Proposed architecture performance comparison on the Indian Pines dataset with regards to overall accuracy (OA), average accuracy (AA) and Cohen’s kappa statistic (κ)

the validation set. The architectures are compared on the Salinas and Pavia University datasets. Note that creating a competing architecture was not the primary goal of the thesis, this is merely additional information for the reader.

Models included in this comparison are Deep Feature Fusion Network (DFFN) [48], Deformable CNN for HSI classification (DHCNet) [49], CNN combined with Markov Random Fields (CNN-MRF) [50], Multigrained Network (MugNet) [51] and a 2-stream 2-D CNN (Fusion) [44]. These architectures are compared the proposed multi-stream architecture described in this thesis.

Architecture	Single-stream	Multi-stream
OA (%)	97.31	98.25
AA (%)	96.91	98.13
$\kappa \times 100$	96.31	98.09
1	94.37	97.46
2	99.44	99.77
3	91.21	97.62
4	99.34	99.66
5	100.0	100.0
6	95.88	95.75
7	98.45	99.13
8	93.67	93.73
9	100.0	100.0

Table 4.3: Proposed architecture performance comparison on the Pavia University dataset with regards to overall accuracy (OA), average accuracy (AA) and Cohen’s kappa statistic (κ)

Classes	DFFN	DHCNet	CNN-MRF	MugNet	Fusion	Proposed
1	99.81±0.26	99.96±0.06	99.81±0.29	64.91±3.30	100±0	99.80±0.07
2	99.83±0.26	99.97±0.07	97.51±1.24	99.90±0.10	99.95±0.10	99.97±0.02
3	99.95±0.16	99.93±0.22	98.94±1.17	99.63±0.18	99.99±0.02	95.54±1.32
4	98.79±1.41	99.69±0.22	99.30±0.69	99.87±0.28	99.99±0.03	99.90±0.14
5	98.93±1.14	99.72±0.21	97.87±0.60	99.34±0.37	99.50±0.44	98.91±0.38
6	99.75±0.31	99.91±0.17	99.83±0.30	98.98±0.18	99.92±0.27	99.99±0.01
7	99.86±0.14	99.72±0.19	98.67±0.96	99.31±0.63	99.99±0.02	99.34±0.43
8	97.36±1.56	95.09±2.52	76.69±2.49	99.40±0.34	99.61±2.70	68.93±7.01
9	99.78±0.32	99.70±0.37	98.77±0.42	89.23±2.12	99.94±0.19	98.39±0.49
10	99.21±0.55	99.77±0.35	94.88±1.72	99.12±0.92	99.75±0.19	91.78±0.51
11	99.22±0.70	99.62±0.43	99.08±0.78	94.72±1.56	99.78±0.34	98.86±0.73
12	99.57±0.62	99.87±0.17	99.99±0.02	99.29±0.54	99.58±1.15	100.0±0.00
13	99.38±0.84	99.95±0.14	99.75±0.35	99.85±0.22	99.93±0.18	100.0±0.00
14	99.89±0.21	99.70±0.53	98.25±0.82	98.43±0.69	99.91±0.14	99.21±0.76
15	97.50±1.75	98.73±0.97	82.43±3.55	97.80±1.14	99.16±1.05	71.17±10.10
16	99.92±0.26	99.92±0.12	97.44±1.97	94.02±2.22	99.99±0.04	99.17±0.59
AA (%)	99.30±0.19	99.45±0.20	96.18±0.41	95.86±0.16	99.63±0.20	88.56±0.17
OA (%)	98.86±0.24	98.67±0.56	91.66±0.68	79.74±1.60	99.09±0.56	95.05±0.22
$\kappa \times 100$	98.73±0.27	98.52±0.63	90.73±0.75	74.35±1.75	98.99±0.62	87.27±0.16

Table 4.4: Comparison of classification accuracies on the Salinas hyperspectral image with regards to overall accuracy (OA), average accuracy (AA) and Cohen’s kappa statistic (κ)

Classes	DFFN	DHCNet	CNN-MRF	MugNet	Fusion	Proposed
1	96.63±1.99	97.46±1.73	84.54±1.85	84.07±2.09	97.07±2.09	88.39±1.41
2	95.50±2.66	98.52±1.39	89.77±1.41	98.42±0.57	99.16±0.50	95.77±1.91
3	97.73±2.84	98.74±0.88	83.04±1.19	97.33±1.01	99.38±0.90	91.18±0.37
4	92.57±2.51	95.20±1.31	97.13±0.75	97.47±0.61	98.37±0.59	98.31±0.37
5	98.88±1.24	99.45±0.94	99.70±0.37	99.90±0.10	100.0±0.00	100.0±0.00
6	98.29±1.65	99.39±0.67	87.66±1.96	98.00±0.49	99.72±0.46	81.98±4.24
7	99.70±0.43	99.45±0.53	92.73±1.29	99.11±0.43	99.96±0.10	98.07±0.74
8	98.19±1.14	98.14±1.27	79.51±3.36	96.01±1.31	97.96±1.56	90.64±0.98
9	94.96±1.12	96.22±1.89	98.65±0.94	99.29±0.49	99.88±0.17	99.96±0.05
AA (%)	96.94±0.66	98.06±0.44	90.30±0.36	96.62±0.33	99.05±0.33	92.71±0.24
OA (%)	96.35±1.36	98.21±0.63	88.61±0.50	95.89±0.39	98.82±0.40	93.76±0.36
$\kappa \times 100$	95.19±1.77	97.62±0.83	85.11±0.60	94.53±0.51	98.43±0.53	90.49±0.28

Table 4.5: Comparison of classification accuracies on the Pavia University hyperspectral image with regards to overall accuracy (OA), average accuracy (AA) and Cohen’s kappa statistic (κ)

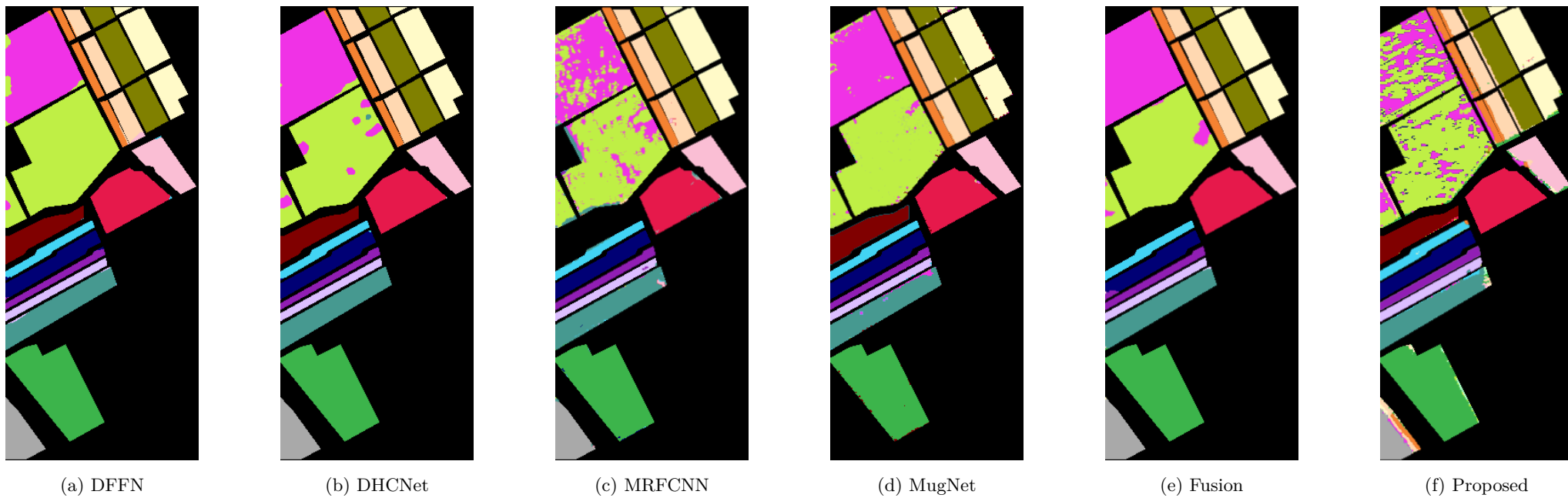


Figure 4.5: Classification maps for the Salinas dataset

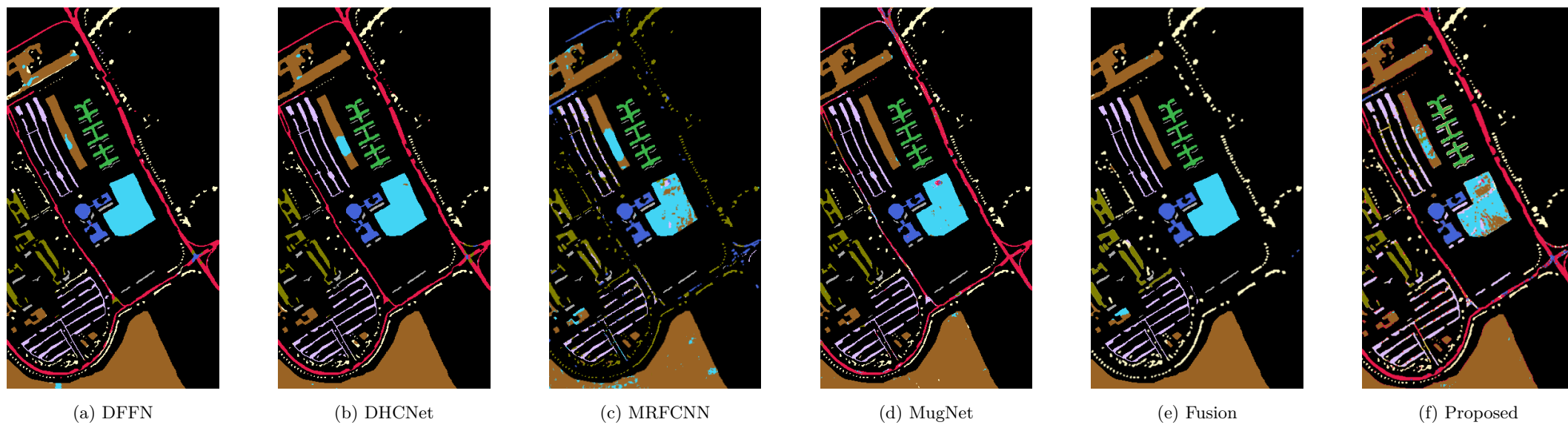


Figure 4.6: Classification maps for the Pavia University dataset

5

Applying theory in practice: a web application

The goal of every academical research is to apply that knowledge to the real world. Research towards hyperspectral image classification is not different in that regard, as there are a lot of practical problems that could benefit from this technique.

Hyperspectral image classification has come quite far, to the point that state-of-the-art models now achieve both an overall and average accuracy of over 99%. When I discovered this during my literature research, I felt motivated to write an application for hyperspectral image classification. The goal of this application is to provide a tool to both academical and more practical use, which is explained further in Section 5.1. This section will serve as context for the thought processes that drove development decisions. A more in-depth feature overview is provided in Chapter 5.2. In Section 5.3, I will outline the software architecture of the application, and provide explanations for the design choices that were made during development.

As the application doesn't rely on ground truth data but rather on user-indicated training data, it is important to see how this influences performance. To this end, I compare the results of the model trained on the ground truth of the Salinas dataset in Section 2.1.1 with the results obtained by the application in Section 5.4. Additionally, in Section 5.5, we see how the application performs on unlabeled data. The source code for this application is available on GitHub [52].

5.1 Goal of the application

Practical hyperspectral datasets become more and more available through various government-run programs like the NASA Airborne Science Team [12], the Earth-Observation 1 satellite missions [22], and its European counterpart, the Sentinel-2 satellites [21]. These datasets are unlabeled, making supervised learning impossible. These practical datasets also differ from academical datasets like the ones discussed in Section 2.1 in that they are very large. These datasets often contain millions of pixels over areas as big as American states, instead of spanning only a couple of square kilometers.

If we want to apply the architectures of HSI classification research to these practical datasets, we need to have some sort of partial manual classification. If a human operator could visualize the dataset, he could easily create relevant classes (eg. forest, grassland, city, crop types, ...), indicate some rough areas where these classes are present in the dataset, and let the machine learning model of their choice do the work.

Other than in Chapter 3, where I mainly focused on classifying hyperspectral images using very little training samples, training time is of less importance compared to inference time. It is important that the preprocessing time per pixel and model complexity is kept low, as to keep the classification time of unlabeled pixels manageable. Luckily the inference process can be sped up drastically using distributed computing, which is discussed in Chapter 6.

The application should be user friendly even for experts without computer science background. Previously obtained results should be easy to retrieve and to visualize, so that the right conclusions can be distilled from the classification results. These results are not limited to classification performance on the manually labeled pixels (where a train-validation-test split should be used). The area covered by the various classes, obtained by multiplying the number of pixels that were classified as that class by the resolution of the dataset, should be calculated for each dataset.

Additionally, faulty input data should be detected. For every classification result, a confusion matrix should be generated. This confusion matrix will indicate where the model is still uncertain. In Figure 5.1, we can see an example of how an inaccurate classification result can be detected. In this example, the pixels with class 7 and 14 are often confused. This example is from the Salinas dataset, where classes 7 (grapes_untrained) and 14 (vineyard_untrained) are very hard to distinguish. This confusion matrix will allow the operator to make changes to the labeled training data, as well as tune the parameters for the model to include more training samples, for example.

Every dataset is different. It might be useful and even necessary to tune the parameters from the models that are present in the application. These models are provided by the user or by the community, and should therefore be easy to incorporate into the application where parameter validation should be available in a modular fashion. We need to take security into account, though. Currently, models are submitted through python code, which requires trust between the user and the author of the model. A solution for this problem is suggested in Chapter 6.

Model result

Performance	
overall accuracy	87.04
average accuracy	93.95
Areas	
Parameters	

Change View

Classification map

Confusion map

Create new model

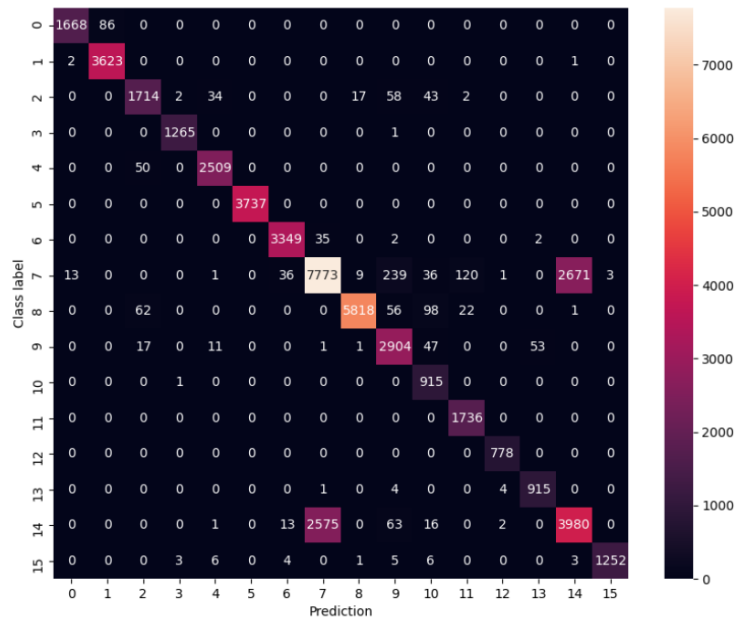


Figure 5.1: Confusion matrix for a partially inaccurate classification result (Salinas dataset)

5.2 Feature overview

In this section, I will go over each page that is present in the web application. Since features are logically divided into these pages, they serve as the ideal backbone for this feature overview section.

5.2.1 Dashboard

The user is greeted in the dashboard (Figure 5.2). On this page, an overview of the application's status is provided. We can see whether the application is currently open for new jobs, and how many jobs are currently in the queue.

On the left, we can see the recent model results, which are models that have recently been completed. Note that with the term 'model' is meant the combination of (1) a dataset, (2) a set of training samples, (3) a deep learning architecture, and (4) the parameters of that architecture. At first glance, the user is able to see a thumbnail for each model result, enabling the user to visually inspect the classification result. Automatically, the overall accuracy is displayed, along with the name of the used dataset and a generated version number for reference later on. Clicking the 'view more' button brings us to the model overview page, which is explained further in Section 5.2.6. Clicking on a particular recent model result will bring the user to a detailed overview of that specific model, discussed in Section 5.2.7.

In the center of the screen, the currently running and queued models can be observed. The layout is largely the same as the recent model results, with the addition of a progress indication for the currently running model, and a queue position for the queued models. Only the top 5 of the queued models is shown, with the total number of jobs in the queue being displayed above this list.

On the right, some important navigational buttons are shown. Uploading a new dataset (5.2.2), viewing uploaded models (5.2.4) and a dedicated page for the Earth Observation 1 satellite datasets (5.2.8) is accessible through these buttons.

Dashboard

Your recent model results



Dataset: Indian Pines
Model: v1.11
Overall Accuracy: 92.23%




Dataset: Indian Pines with ground truth
Model: v29.1
Overall Accuracy: 71.23%



Dataset: Pavia University with ground truth
Model: v28.3
Overall Accuracy: 92.45%



Dataset: Pavia University with ground truth
Model: v28.2
Overall Accuracy: 93.02%



Dataset: Salinas with Ground Truth
Model: v26.3
Overall Accuracy: 88.80%

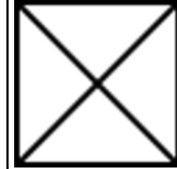
[View more](#)

Server status: online Jobs in queue: 3

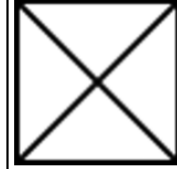
Queued Models



Dataset: Indian Pines
Model: v1.12
Status: running
Epoch: 7/50



Dataset: Indian Pines with ground truth
Model: v29.2
Status: queued
Position in queue: 1



Dataset: Pavia University with ground truth
Model: v28.4
Status: queued
Position in queue: 2

Create a new model

Upload new dataset

Uploaded datasets

Earth Observation 1

Figure 5.2: The dashboard page

5.2.2 Uploading a new dataset

Before a model can be trained, an input dataset is required. Currently, Matlab and TIF files are supported. Each dataset has a required part - the data - and an optional ground truth dataset. In Figure 5.3 the user is uploading the academical Kennedy Space Center dataset. After the user uploaded the data file, the data and its dimensions are automatically detected. Optionally, the user can now also upload the ground truth for this dataset. The API will check if the dimensions of the ground truth data match with the dimensions of the dataset. When the user clicks "Confirm", the detailed dataset page is displayed.

5.2.3 Detailed dataset page

When a user uploads a new dataset, or wants to change an already existing dataset, they are taken to the detailed dataset page. On this page, all information of the data itself is gathered and displayed. For user experience purposes, a dataset can be manually named. This name will show in the model result, and might be useful when a user has two identical datasets: one that uses the ground truth, and one that uses manually specified labeled data.

In the details about the model, the extracted information from the data is shown. Information that can be extracted from the dataset are its dimensions, and how many spectral bands are present in the dataset. Resolution is often not provided in these datasets, as it is unnecessary to evaluate classification performance. However, as this application also targets practical problems, the possibility to manually specify the resolution is provided. This allows the application to calculate the surface area of the various classes in the resulting classification map. If this field is left blank, the surface areas are calculated in pixels instead.

If the user chose to upload a ground truth dataset, it is displayed on the right. A predetermined list of colors is chosen to be contrasting and vibrant. If the user has not uploaded a ground truth map, or chooses not to use the ground truth for specifying training samples, class colors can be specified manually. When the user saves the changes done to the dataset, the dataset becomes visible in the dataset overview.

5.2.4 Uploaded datasets

On this page (Figure 5.5), an overview is provided for the different datasets that have been uploaded to the web application. From here, the user can see an overview of trained models that use this dataset (Section 5.2.6), create a new model for this dataset (Section 5.2.5), or edit the dataset's properties (Section 5.2.3).

Matlab file

Hyperspectral image data

Upload file

Found valid data. Please confirm the information below and press confirm

Key in datafile that contains the data: KSC
Dimension (spatial): 512 x 614
Number of spectral bands: 176

Confirm

Ground truth data (optional)

Upload file

Found valid ground truth data

Figure 5.3: Uploading a new dataset

Dataset

Give this dataset a name

Dataset size

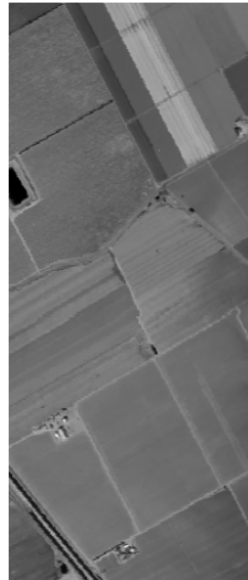
Dimensions: 512 x 217

Spectral bands: 204

Resolution

 m²

False colour image








Ground Truth



Figure 5.4: Detailed dataset page

Your datasets

	<input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Salinas	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Pavia University	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Indian Pines with ground truth	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Salinas with Ground Truth	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>

Common datasets






 Indian Pines	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Salinas	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Pavia Center	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Pavia University	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>
 Kennedy Space Center	<input type="button" value="View models that use this dataset"/> <input type="button" value="Create new model from this dataset"/> <input type="button" value="Edit this dataset"/>

Figure 5.5: The overview of uploaded datasets

5.2.5 Creating a new model

The core functionality of the web application is classifying hyperspectral images. A hyperspectral model consists of four parts: it is trained on a certain dataset (1), using supervised learning. To have supervised learning, we also need labeled training samples (2). The model is trained on an architecture (3) with specified hyperparameters for that specific architecture (4). The first and second inputs are specified in the first window of the model creation process (Figure 5.6), while the third and fourth inputs are shown in the architecture selection page (Figure 5.7).

A new model can be created in two ways: either through the uploaded datasets page (Section 5.2.4) or through the result page of an already trained model (Section 5.2.7), in which case the parameters and selected training samples used for that model are carried to the new model.

Selecting training samples for the supervised learning algorithm can be done manually, or by using the ground truth if this is available for the chosen dataset. In Figure 5.6, the manual process is shown. On the right, all classes present in the dataset are displayed. These classes are manually specified by the user, and are given a color.

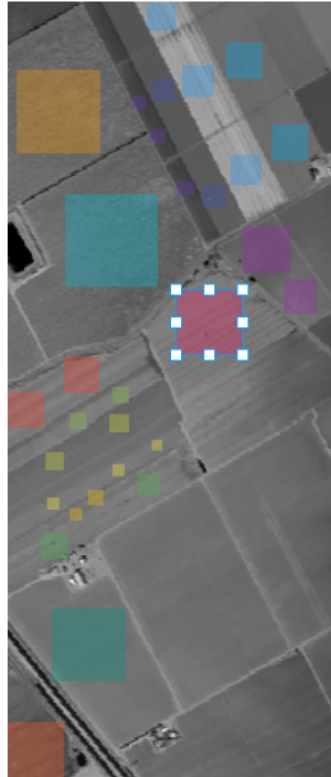
On the right, the grayscale visualization of the dataset is shown. This image is obtained through the backend, by taking a slice from the dataset, normalizing its sensor values, and coloring the result in grayscale. By default, the spectral band in the middle is displayed. The user can choose to display another spectral slice by using the appropriate buttons below the grayscale visualization.

By pressing the "New" button of a specified class, a new rectangle in the color of that class is added to the canvas of the visualization. All pixels beneath this rectangle are assigned as labeled training samples of that class. The user can rescale this rectangle to have the desired dimensions. The actual ground truth data of the Salinas dataset is visualized in Figure 2.1b.

As some colors can become hard to differentiate from each other with many classes, the user can see the assigned class of each rectangle by selecting it. A tooltip is shown with the assigned class, under the button that allows the user to delete it altogether.

When the user is satisfied with their work, they click the "Run model" button, which shows part 2 of the model creation process: selecting the architecture and its parameters (Figure 5.7). On this page, the user can select the architecture that will train the model at the top of the page, and configure its parameters. These parameters are configurable by the architecture, and are displayed dynamically on the page. The architecture creator can choose default values and limits for these parameters. The values are filled in if no parameters were carried from an earlier model, and limits are enforced by the application.

Salinas



Layer (max 203) 102 Change layer Delete selected instance

Selected instance class: Broccoli-green-weeds-2

Classes

1	Broccoli-green-weeds-1 Pixels selected: 1128 / 50	New	Trash
2	Broccoli-green-weeds-2 Pixels selected: 1892 / 50	New	Trash
3	Fallow Pixels selected: 1467 / 50	New	Trash
4	Fallow-rough-plow Pixels selected: 281 / 50	New	Trash
5	Fallow-smooth Pixels selected: 465 / 50	New	Trash
6	Stubble Pixels selected: 1183 / 50	New	Trash
7	Celery Pixels selected: 1152 / 50	New	Trash

Run model Add new class

Figure 5.6: Model creation page

Model Selection

capsule networks svm multiscale capsule_networks

Parameters

patch_radius_stream_0 <input type="range"/> <input type="text" value="1"/>	patch_radius_stream_1 <input type="range"/> <input type="text" value="3"/>	patch_radius_stream_2 <input type="range"/> <input type="text" value="5"/>	spatial_kernel_size_stream_0 <input type="range"/> <input type="text" value="2"/>
spatial_kernel_size_stream_1 <input type="range"/> <input type="text" value="2"/>	spatial_kernel_size_stream_2 <input type="range"/> <input type="text" value="2"/>	dim_capsule_primary_stream <input type="range"/> <input type="text" value="2"/>	nchannels_primary_stream_0 <input type="range"/> <input type="text" value="4"/>
nchannels_primary_stream_1 <input type="range"/> <input type="text" value="4"/>	nchannels_primary_stream_2 <input type="range"/> <input type="text" value="4"/>	spectral_kernel_size <input type="range"/> <input type="text" value="16"/>	training_samples_per_class <input type="range"/> <input type="text" value="50"/>
validation_samples_per_class <input type="range"/> <input type="text" value="50"/>	epochs <input type="range"/> <input type="text" value="10"/>	batch_size <input type="text" value="10"/>	learning_rate <input type="text" value="0.001"/>
learning_rate_decay <input type="text" value="0.9"/>			

Run model

Figure 5.7: Architecture selection page

5.2.6 Model result overview page

In the model overview (Figure 5.8), the user can search through the model results. Several filtering options are added on top: filtering by dataset and by architecture, as well as on a specific version using the 'search version' input box. The results can be sorted by when they were finished, or their classification performance on the test set.

Each model can be clicked on, and brings the user to the detailed model result (see 5.2.7). Again, the name and version of the model is shown, as well as the overall accuracy of the classification performance of the test set. The user can flip through the pages of model results by using the navigation buttons at the bottom.

5.2.7 Detailed model result

Clicking on a recent model in the dashboard or on the model overview page brings the user to the detailed model result page (Figure 5.9). On this page, the user is greeted with the classification map of the entire dataset. This image is obtained by applying the learned model on every part of the dataset, in order. The colors represent the different classes. These are specified when indicating classes manually, or picked automatically if the ground truth was chosen to train the model.

The key metrics of the model are displayed on the left: performance, classification areas and parameters. Clicking on a section will open that section and close the others. In the "Performance" section, the overall and average accuracy are displayed. Additionally, other performance metrics such as the kappa metric are shown if the model architecture calculates it.

Under the "Areas" section, the surface area per class is displayed. This allows the user to quickly inspect how much land a certain class covers, which can be useful to climate research, for example. Finally, under "Parameters", the chosen parameters to train the model are displayed. These parameters were specified when the user enqueued the model.

Using the buttons under "Change View", the user can change the current view. By default, the classification map of the dataset is shown. The user can also display the confusion matrix should they desire, for example to see for which classes the model is unsure. This confusion matrix is calculated on the labeled data that remains after choosing training and validation sets, and is therefore never before seen by the model. Performance metrics are also calculated on the test set only.

Models

Dataset: Architecture: Sort: Search version:

Showing results 71-80. Total results: 174











 <p>Dataset: Pavia University with ground truth Model: v10.1 Overall Accuracy: 75.30%</p>	 <p>Dataset: Salinas with Ground Truth Model: v9.10 Overall Accuracy: 84.61%</p>
 <p>Dataset: Salinas with Ground Truth Model: v9.9 Overall Accuracy: 85.06%</p>	 <p>Dataset: Salinas with Ground Truth Model: v9.8 Overall Accuracy: 87.04%</p>
 <p>Dataset: Salinas with Ground Truth Model: v9.7 Overall Accuracy: 84.26%</p>	 <p>Dataset: Salinas with Ground Truth Model: v9.6 Overall Accuracy: 84.04%</p>
 <p>Dataset: Salinas with Ground Truth Model: v9.5 Overall Accuracy: 85.68%</p>	 <p>Dataset: Salinas with Ground Truth Model: v9.4 Overall Accuracy: 85.87%</p>
 <p>Dataset: Salinas with Ground Truth Model: v9.3 Overall Accuracy: 88.09%</p>	 <p>Dataset: Salinas with Ground Truth Model: v9.2 Overall Accuracy: 84.39%</p>

Figure 5.8: The model overview page

Model result

Performance	
overall accuracy	97.31
average accuracy	96.91

Areas

Parameters

Change View

Classification map

Confusion map

Create new model



Figure 5.9: The model result page

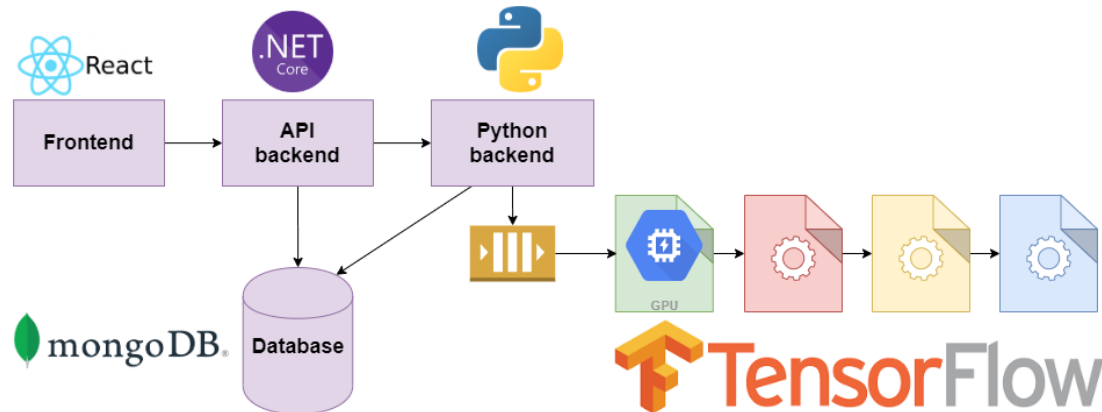


Figure 5.10: Software architecture of the web application

After the user has inspected the classification map and the confusion matrix, they might want to change the parameters of the model and/or add more training samples to the model. The "Create new model" button creates a new instance of the model, where the user can yet again specify the training data and model parameters. For a good user experience, all parameters and manually defined classification areas are carried to that new model.

5.2.8 Earth Observation 1 satellite datasets

Datasets provided by NASA of the Earth Observation 1 satellite can also be uploaded to the application. As the process is quite similar to uploading any other dataset, this process will not be discussed here. Currently, the EarthExplorer website allows the user to download each dataset in 5 different formats. For the application, I have chosen to work with the 'L1T' format, which delivers its data in a zip file. This zip file contains n files, where n is the number of spectral bands in the image. I have found n to be 242 over all available datasets.

The satellite datasets have lower quality and resolution compared to the academical datasets: atmospheric features like clouds are more widespread, as the satellites provide a huge number of datasets completely autonomously. This is different from the academical datasets in that they are gathered at the optimal moment, where little atmospheric interference is present. Atmospheric interference also causes certain spectral bands to be completely without data. This is an additional challenge for the supervised learning process, as the spectral signature is no longer as smooth.

5.3 Software architecture

The software architecture can be seen schematically in Figure 5.10. It provides a high-level overview on how the functionality of the application is divided and interconnected. In this section, I will go over each of the four main components, and explain how models are trained in their own process. The motivation

behind the language and/or framework choices is also provided, along with key libraries if relevant.

5.3.1 Frontend

The frontend is what allows the user to interact with the features that the application provides. Especially for the goal of this application, much functionality is implemented in Python, and is therefore not directly accessible to the user. User experience has a very high priority throughout the development process.

In modern web development, one is required to use a frontend library for creating web applications. Libraries like Angular, React, and Vue, are valuable tools that take care of responsiveness, lifetime management, avoiding code duplication, and so on. Being experienced with both Angular and React, I chose React for its very fast prototyping and easy component approach. In React, each page is a component, which might in turn consist of various components that are reused throughout the application. In my final product as well, the reuse of components can be seen in Figure 5.8 and Figure 5.2, in the model result summary items.

Angular also has a very powerful component system, and with its many community-driven libraries is highly optimized for huge company websites. In the end it came down to learning curve: the learning curve of Angular is, even for experienced developers, very steep. React on the other hand is not as powerful, but has an easier learning curve. Due to the application not really needing the powerful features of Angular, and the limited time constraints, I opted to use React for frontend development.

Two very important react libraries are the KonvaJS [53] and Bootstrap [54] libraries. Bootstrap is a widely known frontend-framework directed at responsive, mobile-first front-end web development. It has responsive CSS and Javascript-based design template for all kinds of HTML elements like buttons, containers, navigation, forms, and many more. It applies a highly customizable style to the web application, making it attractive to the end user.

KonvaJS is a Javascript library that expands the Javascript canvas element. A canvas is used in website where graphical user input is required, for example on a website with draggable elements, drawing functionality, and photo editing in the browser. For my application specifically, it is used when manually labelling terrain classes, as can be seen in Figure 5.6.

5.3.2 Database

In the database, data that needs to persist restarts of the application is stored. A big part of my hyperspectral classification application is the uploaded datasets. These datasets need to be stored by the backend, so that it is accessible to the python model for training. There are some database technologies that are made specifically for file storage, but I was looking for a technology that could both save files, as well as regular data entries. These data entries are metadata for the information saved in the application like model performance and dataset names.

The solution was found in the MongoDB software [55] with the GridFS plugin. MongoDB stores data rows as so-called BSON documents, where each document has a size limit of 16 MB. For hyperspectral

images, even the small academical datasets exceed this limit, which is where GridFS comes in. The GridFS plugin splits an uploaded file into smaller documents, which are then stored in the MongoDB database. With implemented methods to both split and merge these files, it is well suited for the task of storing the hyperspectral datasets.

5.3.3 Backend

The motivation for having a backend is twofold: it serves as an interface to the database, which often contains data that not everyone may view or change, and less often it offloads heavy processes from the client machine. In the case of this application both are required, and are split in two programs. In the first section, I will discuss the part of the backend that is responsible for interacting with the database. In the second section I will focus on offloading the heavy computational processes, mainly training deep learning models, from the client machine.

1) API

While the processing backend is implemented in python, the API that provides interaction between the frontend and the database is implemented using the .NET core framework by Microsoft [56]. This relatively new framework from 2016 is developed as the successor of the .NET framework. In contrast to .NET framework, it can be used for cross-platform development. Since I work on a Windows machine, but want my application to be able to be deployed on Linux as well, this is a hard requirement. For future development as well, .NET core makes more sense since it is designed to work in a microservice environment. More information on this programming technique can be found in Chapter 6.

The main motivation behind not implementing the API backend in python as well so that I can join both applications is simple: I am personally more experienced with .NET core, and I feel like .NET core has more features to make designing an API easier. Additionally, the database-as-code feature of .NET core through the entity framework system is really powerful, so that database technologies can be swapped out without much work.

In the API backend, I employ the Model-View-Controller (MVC) software design pattern: database objects are represented as C# classes. These models can be read from and written to the database through services, which provide specific implementations of features like renaming a dataset. The functionality of the services is accessed through API controllers: these controllers provide validation and transformation for the service method implementations. This way, responsibilities are separated and code can be maximally reused, making development more efficient.

2) Python worker

While as much functionality as possible is written in the API backend, there is functionality that can only be implemented in python. Two examples are reading hyperspectral image datasets and training deep learning models. The academical datasets are published as MATLAB files, which makes reading them without third-party libraries very difficult. While .NET is advancing to become more centered around machine learning, it still has a long way to go. For the python language, in contrast, many libraries focusing on machine learning exist. Reading MATLAB files through the Scipio [57] library is very easy, making it the preferred method for this specific task.

Training the deep learning models for the users of the application is done through whatever library the author chooses. I personally implemented both my proposed method (Section 3) as well as other deep learning architectures using the TensorFlow framework by Google [58]. As the preferred language for machine learning by many developers is still python, I decided that the default processing backend should be written in python. In future development, though, it should be very easy to add an additional processing backend which uses another language.

The processing backend consists of three distinct parts: API endpoints, a master model runner, and slave model trainer processes. The functionality of these three parts will be outlined in the following three paragraphs.

Throughout the application, data visualizations are used. In figures 5.6 and 5.9, the user can respectively see the ground truth of the dataset and the classification map visualized in the frontend. Since reading the hyperspectral files is done through the python backend, it needs to provide endpoints to deliver information directly to the user. This functionality is accessible through API endpoints, which are called from the API backend on the user's behalf. Validating uploaded datasets is done through these endpoints as well, for the same reason.

Every five seconds, the master checks to see whether a new model can be trained. This process runs in the same process as the API endpoints, but as a separate thread. When a model in the queue is ready to be trained, the master model runner will spawn a new slave process in charge of actually training the model, and will monitor that process until it is finished. This prevents a crashing model from bringing down the entire python backend, and separates the two responsibilities. This way, the API remains responsive while the slave process is hard at work training the new model.

Finally, the slave model trainer is responsible for actually training the model. As it runs in a separate process, it is completely isolated from any other program. This way, performance is optimal since no thread scheduling should be done, and a specific amount of resources can be reserved for that particular process. Apart from training the model and classifying the entire dataset using the trained model, this slave process is responsible for updating its progress to the database. This progress can then be displayed by the frontend.

5.4 Experimental results on academic datasets

In this and the next Section, I will show some experimental results from two academical datasets and one satellite dataset in my application. The results here are different from the results discussed in Chapter 4.2 in that they were trained using manually defined training data. In rapid succession, I will show the ground truth (if available), the manually labeled training data, and the generated classification map for the three datasets. Per dataset, I will also briefly discuss the impact of manually specifying training samples on the classification accuracy.

All results were obtained by randomly selecting 50 training samples and 50 validation samples. Each model was trained for 50 epochs, and the parameters are the same across all models. The overall accuracy and average accuracy is calculated on the test set, which consists of all labeled samples not present in the training or validation set.

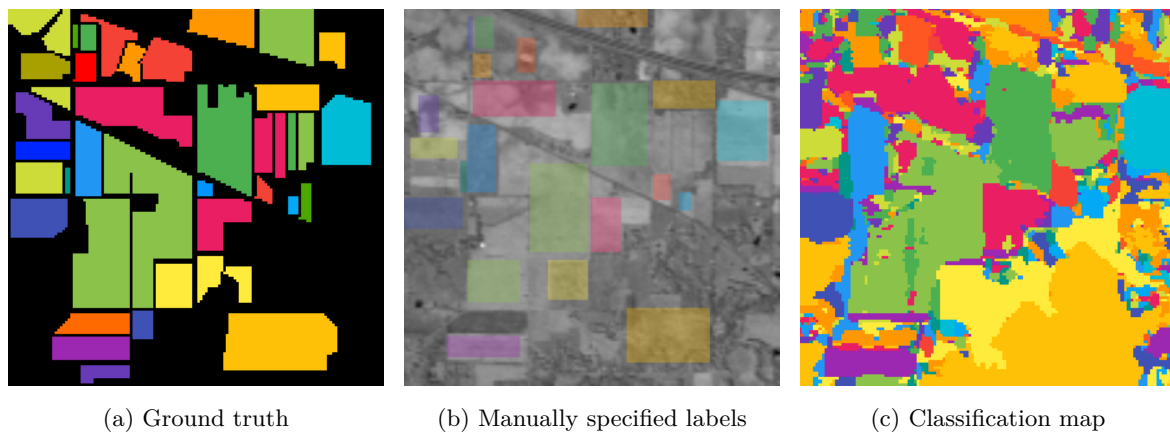


Figure 5.11: Ground truth, labels, and generated classification map for the Indian Pines dataset

5.4.1 Indian Pines dataset

Compared to using the ground truth as input for the training process, the labeled samples indicated in Figure 5.11 provide an overall accuracy of 94.22%, and an average accuracy of 97.47%. When we look at the results on the model that was trained using the ground truth data, we see something weird: the classification scores of 92.23% and 96.53% for overall accuracy and average accuracy are slightly lower compared to the manually labeled model. This means that if done correctly, manually specifying the classes has no or even a positive effect on classification performance.

5.4.2 Salinas dataset

For the Salinas dataset, we achieve an overall accuracy of 82.82%, with an average accuracy of 89.61% when we use manually labeled training samples. Compared to training the ground truth labels, this is quite a bit lower as this model achieves an overall accuracy of 88.80% and an average accuracy of 94.80%.

5.5 Experimental results on satellite datasets

Finally, we arrive at a satellite dataset. This dataset of the island of Moroni near the coast of Mozambique, features two rough geographic features: ocean and land. Due to the poor resolution of these datasets, this is for now as far as we can go.

Again, 50 labeled training samples were used to train the model, although there are millions of pixels in this classification map. These pixels come from the manually specified labels in the application. When we process the result, we obtain the classification map, which is accurately mapping the contours of the island. In Figure 5.13d especially, we can see the overlay of the transparent classification map on the map view to see that it indeed follows its contours. Note that even the small bulge on top of the island is detected, even though there are no training pixels in that area.

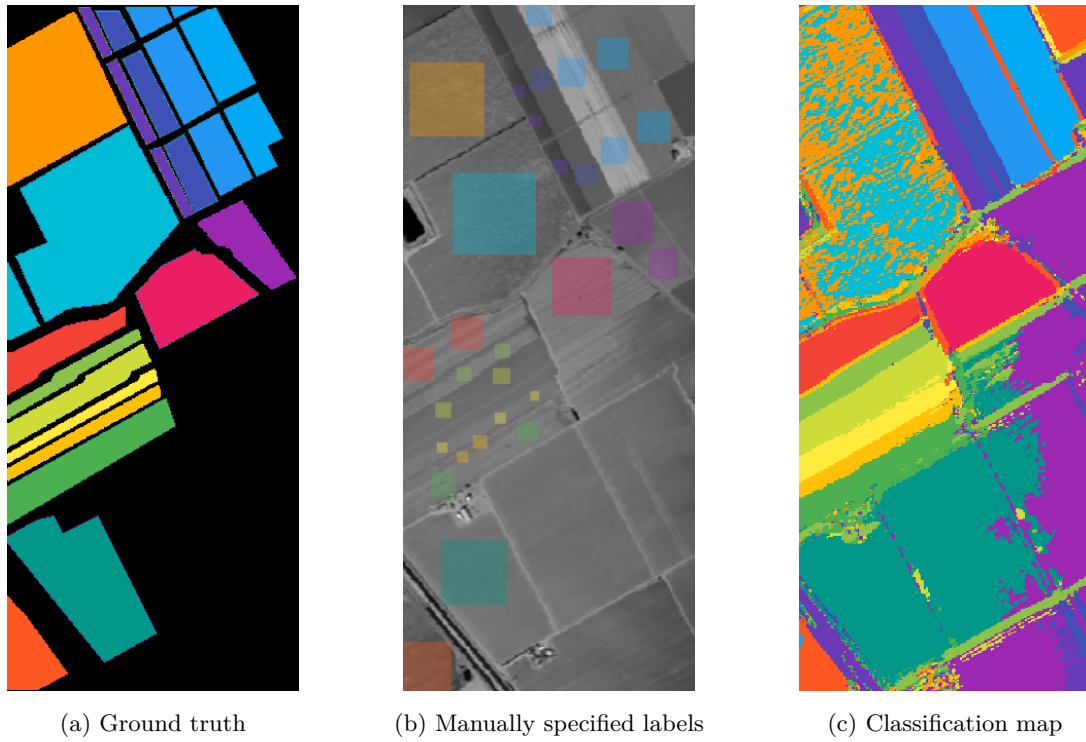


Figure 5.12: Ground truth, labels, and generated classification map for the Salinas dataset

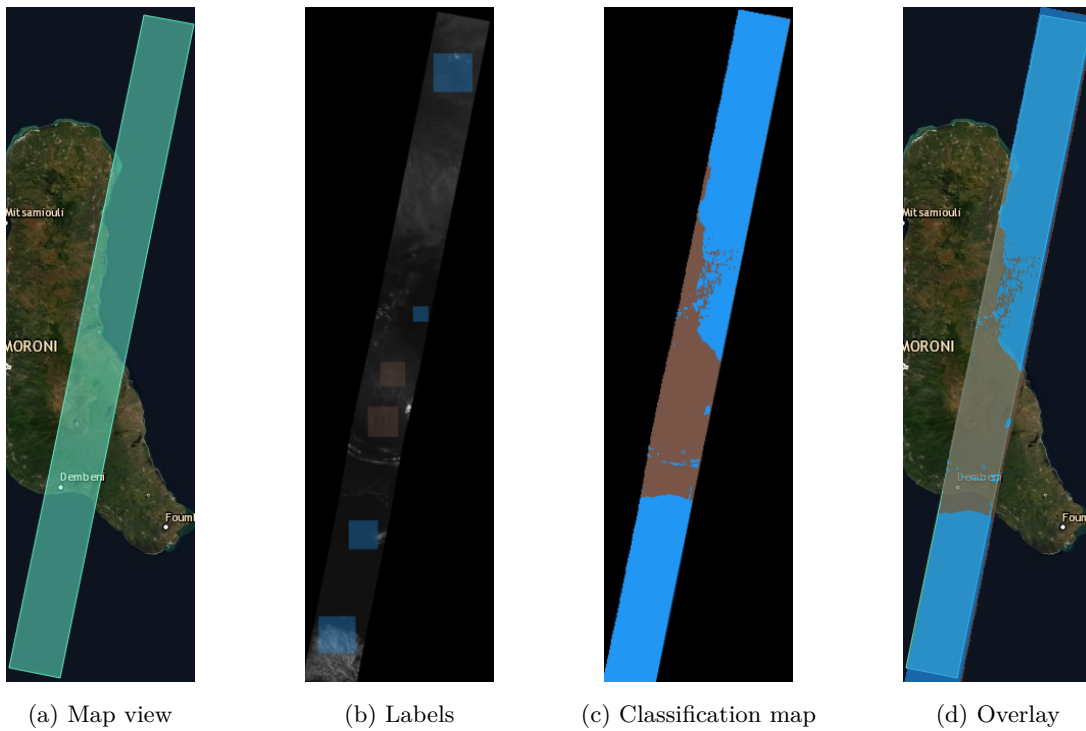


Figure 5.13: Classification maps for the satellite dataset

6

Future work

In this chapter, I will discuss future work for the multi-stream architecture and capsule networks in general in Section 6.1, as well as future development for the web application described in Chapter 5 in Section 6.2.

6.1 Improving the multi-stream architecture

As can be seen in the experimental results, this architecture is not up to par with current state-of-the-art techniques. I believe that capsule networks can become a state-of-the-art deep learning technique for HSI classification. A first possible avenue of future work is simply refining this multi-stream architecture. During the time of my thesis, I primarily focused on three streams, where each stream has an equal amount of output features. This might be suboptimal: the impact of these hyperparameters should be investigated more in-depth.

The describe multi-stream architecture uses 3-dimensional inputs. A variant using 2-dimensional inputs (selecting spectral slices through PCA) and 1-dimensional inputs should also be explored. The impact of normalization on the spectral layers should be investigated.

6.2 Future development on the web application

In this section, I will discuss avenues for further development on the web application.

Uploadable model architectures: adding a new model architecture is done by implementing an interface that will take care of preprocessing, performance evaluation and classification map generation. However to be able to use the model architecture in the web application, the user needs to modify the software by manually uploading their code file to the application, recompiling it and redeploying it. Both for user experience and security (this application could run on internal server infrastructure), it would be better if the user could upload their code file, just like is already possible with datasets. This way, the disadvantages of having to modify the software files is avoided.

Implement more data formats: hyperspectral images are always published in a format. While MATLAB is often used for academical datasets, more practical datasets are often distributed in a custom data format. For example: the Hyperion hyperspectral images from the Earth Observation 1 mission that is implemented in the web application uses a custom format based on a .TXT metadata file and hundreds of geotiff files. As more practical datasets become available, the need for custom data parsers will inevitably rise. Additionally, some dataset providers provide API access to their datasets. Fetching these datasets through the API would improve the user experience and would reduce the storage requirements for the application.

Supporting community-driven development: a major task of any researcher that comes up with a new deep learning architecture is comparing their results against other architectures. Some researchers do not provide code to run their architectures, or provide insufficient documentation on how to run the code, leading to frustration. This forces researchers to reproduce the work of others, taking up valuable research time and possibly introducing bugs or guessed hyperparameters not described in the paper of the target architecture. This inevitably leads to unreproducible classification results, leading to biased research conclusions. Creating a community platform where users can share their architectures using this web application would greatly reduce these problems and would be a big advantage to the research community. Additionally, it allows less experienced end-users to "shop" for an architecture based on their needs to solve real problems.

Distributed computing: the inference step of most architectures on practical datasets takes orders of magnitude more time compared to the training time. This inference step can easily be accelerated through distributed computing: distributing the model and training subsets of pixels would greatly improve computation time.

7

Conclusion

In this thesis, capsule networks were explored for the hyperspectral image classification task, where the goal is to assign a geological class to an area on the globe. To remedy the problems of computational complexity and the trade-off between overfitting and underfitting in current methods, a new 3D multi-stream architecture was proposed based on capsule networks. This architecture has shown that combining multiple streams with varying degrees of detail improves the classification performance considerably. Additionally, the architecture performs well with very little training data, making it fit to be applied in a practical setting.

To explore this practical setting, a web application was created as a bridge between academic research and practical applications. This application allows remote sensing researchers and machine learning experts to work together towards applying hyperspectral image classification on practical datasets like those captured by satellites at NASA and ESA. The software architecture is designed in a way that multiple users can test their architectures and run their models in a queue, removing the need for dedicated hardware for every individual workstation.

To prove that capsule networks are a valid technique for remote sensing applications, and that the results of this research can be used in a practical context, the architecture was compared to the current state of the art in hyperspectral image classification. These results show that while the architecture is not as good as other state-of-the-art architectures for some academical datasets, capsule networks can deliver very good results for practical applications. A follow-up article may enhance this architecture to further increase the performance of the proposed multi-stream architecture on hyperspectral image classification.

Bibliography

- [1] A. M. Milner, K. Khamis, T. J. Battin, J. E. Brittain, N. E. Barrand, L. Füreder, S. Cauvy-Fraunié, G. M. Gíslason, D. Jacobsen, D. M. Hannah, A. J. Hodson, E. Hood, V. Lencioni, J. S. Ólafsson, C. T. Robinson, M. Tranter, and L. E. Brown, “Glacier shrinkage driving global changes in downstream systems,” *PNAS : Proceedings of the National Academy of Sciences of the United States of America*, vol. 114, pp. 9770–9778, 2017.
- [2] W. Li, R. Dong, H. Fu, J. Wang, L. Yu, and P. Gong, “Integrating google earth imagery with landsat data to improve 30-m resolution land cover mapping,” *Remote Sensing of Environment*, vol. 237, p. 111563, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0034425719305838>
- [3] P. Ghamisi, M. Dalla Mura, and J. A. Benediktsson, “A survey on spectral–spatial classification techniques based on attribute profiles,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 5, pp. 2335–2353, May 2015.
- [4] J. Meola, M. T. Eismann, R. L. Moses, and J. N. Ash, “Application of model-based change detection to airborne vnir/swir hyperspectral imagery,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 10, pp. 3693–3706, Oct 2012.
- [5] L. G. Olmanson, P. L. Brezonik, and M. E. Bauer, “Airborne hyperspectral remote sensing to assess spatial distribution of water quality characteristics in large rivers: The mississippi river and its tributaries in minnesota,” *Remote Sensing of Environment*, vol. 130, pp. 254–265, 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0034425712004555>
- [6] M. J. Khan, H. S. Khan, A. Yousaf, K. Khurshid, and A. Abbas, “Modern trends in hyperspectral image analysis: A review,” *IEEE Access*, vol. 6, pp. 14 118–14 129, 2018.
- [7] S. Yuan, C. Liu, and X. Liu, “Practical model of sea ice thickness of bohai sea based on modis data,” *Chinese Geographical Science*, vol. 28, 07 2018.
- [8] P. R. Robichaud, S. A. Lewis, D. Y. Laes, A. T. Hudak, R. F. Kokaly, and J. A. Zamudio, “Postfire soil burn severity mapping with hyperspectral image unmixing,” *Remote Sensing of Environment*, vol. 108, no. 4, pp. 467–480, 2007. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0034425706005025>
- [9] Y. Yue, K. Wang, B. Zhang, Q. Jiao, B. Liu, and M. Zhang, “Remote sensing of fractional cover of vegetation and exposed bedrock for karst rocky desertification assessment,” *Procedia Environmental Sciences*, vol. 13, pp. 847–853, 2012, 18th Biennial ISEM Conference on Ecological

- Modelling for Global Change and Coupled Human and Natural System. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1878029612000795>
- [10] R. Anand, S. Veni, and J. Aravinth, "Big data challenges in airborne hyperspectral image for urban landuse classification," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, 2017, pp. 1808–1814.
- [11] H. Yao, L. Tian, G. Wang, and I. Colonna, "Estimation of soil fertility using collocated cokriging by combining aerial hyperspectral imagery and soil sample data," *Applied Engineering in Agriculture*, vol. 30, pp. 113–121, 01 2014.
- [12] N. California Institute of Technology, "The aviris sensor," 2021. [Online]. Available: <https://aviris.jpl.nasa.gov/>
- [13] P. R. Foundation, "Indian pines dataset," 1994. [Online]. Available: <https://engineering.purdue.edu/~biehl/MultiSpec/hyperspectral.html>
- [14] L. He, J. Li, C. Liu, and S. Li, "Recent advances on spectral–spatial hyperspectral image classification: An overview and new guidelines," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 3, pp. 1579–1597, March 2018.
- [15] M. E. Paoletti, J. M. Haut, R. Fernandez-Beltran, J. Plaza, A. Plaza, J. Li, and F. Pla, "Capsule networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 4, pp. 2145–2160, 2019.
- [16] S. Jia, Z. Lin, M. Xu, Q. Huang, J. Zhou, X. Jia, and Q. Li, "A lightweight convolutional neural network for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, pp. 1–14, 2020.
- [17] L. Mou, X. Lu, X. Li, and X. X. Zhu, "Nonlocal graph convolutional networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 12, pp. 8246–8257, 2020.
- [18] Y. Chen, H. Jiang, C. Li, X. Jia, and P. Ghamisi, "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 6232–6251, 2016.
- [19] B. Liu, X. Yu, P. Zhang, A. Yu, Q. Fu, and X. Wei, "Supervised deep feature extraction for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 4, pp. 1909–1921, 2018.
- [20] S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson, "Deep learning for hyperspectral image classification: An overview," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 9, pp. 6690–6709, 2019.
- [21] ESA, "Sentinel-2 satellite mission," 2000. [Online]. Available: <https://sentinel.esa.int/web/sentinel/missions/sentinel-2>
- [22] NASA, "Earth observation 1 (eo1) satellite mission," 2000. [Online]. Available: https://www.usgs.gov/centers/eros/science/earth-observing-1-eo-1?qt-science_center_objects=0#qt-science_center_objects

- [23] X. Ma, X. Mou, J. Wang, X. Liu, H. Wang, and B. Yin, "Cross-data set hyperspectral image classification based on deep domain adaptation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 12, pp. 10 164–10 174, 2019.
- [24] C. Zhong, J. Zhang, S. Wu, and Y. Zhang, "Cross-scene deep transfer learning with spectral feature adaptation for hyperspectral image classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 2861–2873, 2020.
- [25] D. Hong, L. Gao, N. Yokoya, J. Yao, J. Chanussot, Q. Du, and B. Zhang, "More diverse means better: Multimodal deep learning meets remote-sensing imagery classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 59, no. 5, pp. 4340–4354, May 2021.
- [26] C. Xiang, L. Zhang, Y. Tang, W. Zou, and C. Xu, "Ms-capsnet: A novel multi-scale capsule network," *IEEE Signal Processing Letters*, vol. 25, no. 12, pp. 1850–1854, Dec 2018.
- [27] K. Zhu, Y. Chen, P. Ghamisi, X. Jia, and J. A. Benediktsson, "Deep convolutional capsule network for hyperspectral image spectral and spectral-spatial classification," *Remote Sensing*, vol. 11, no. 3, 2019. [Online]. Available: <https://www.mdpi.com/2072-4292/11/3/223>
- [28] C. Zhong, J. Zhang, S. Wu, and Y. Zhang, "Cross-scene deep transfer learning with spectral feature adaptation for hyperspectral image classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 2861–2873, 2020.
- [29] H. C. Li, W. Y. Wang, L. Pan, W. Li, Q. Du, and R. Tao, "Robust capsule network based on maximum correntropy criterion for hyperspectral image classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 13, pp. 738–751, 2020.
- [30] B. Olshausen, C. Anderson, and D. Van Essen, "A neurobiological model of visual attention and invariant pattern recognition based on dynamic routing of information," *Journal of Neuroscience*, vol. 13, no. 11, pp. 4700–4719, 1993. [Online]. Available: <https://www.jneurosci.org/content/13/11/4700>
- [31] G. E. Hinton, Z. Ghahramani, and Y. W. Teh, "Learning to parse images," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 2000. [Online]. Available: <https://proceedings.neurips.cc/paper/1999/file/5a142a55461d5fef016acfb927fee0bd-Paper.pdf>
- [32] S. Billot and B. Lang, "The structure of shared forests in ambiguous parsing," Ph.D. dissertation, INRIA, 1989.
- [33] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," *CoRR*, vol. abs/1710.09829, 2017. [Online]. Available: <http://arxiv.org/abs/1710.09829>
- [34] J. Rajasegaran, V. Jayasundara, S. Jayasekara, H. Jayasekara, S. Seneviratne, and R. Rodrigo, "Deepcaps: Going deeper with capsule networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [35] T. Jeong, Y. Lee, and H. Kim, "Ladder capsule network," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 3071–3079. [Online]. Available: <http://proceedings.mlr.press/v97/jeong19b.html>

- [36] V. M. d. Rosario, E. Borin, and M. Breternitz, "The multi-lane capsule network," *IEEE Signal Processing Letters*, vol. 26, no. 7, pp. 1006–1010, 2019.
- [37] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," 2017.
- [38] S. Srivastava, P. Khurana, and V. Tewari, "Identifying aggression and toxicity in comments using capsule network," in *Proceedings of the First Workshop on Trolling, Aggression and Cyberbullying (TRAC-2018)*, 2018, pp. 98–105.
- [39] A. Shahroudnejad, P. Afshar, K. N. Plataniotis, and A. Mohammadi, "Improved explainability of capsule networks: Relevance path by agreement," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, 2018, pp. 549–553.
- [40] T.-H. Hsieh and J.-F. Kiang, "Comparison of cnn algorithms on hyperspectral image classification in agricultural lands," *Sensors*, vol. 20, no. 6, 2020. [Online]. Available: <https://www.mdpi.com/1424-8220/20/6/1734>
- [41] W. Wang, H. Li, L. Pan, G. Yang, and Q. Du, "Hyperspectral image classification based on capsule network," in *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, 2018, pp. 3571–3574.
- [42] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966. [Online]. Available: <https://science.sciencemag.org/content/153/3731/34>
- [43] J. Yang, Y.-Q. Zhao, J. C.-W. Chan, and L. Xiao, "A multi-scale wavelet 3d-cnn for hyperspectral image super-resolution," *Remote Sensing*, vol. 11, no. 13, 2019. [Online]. Available: <https://www.mdpi.com/2072-4292/11/13/1557>
- [44] X. Li, M. Ding, and A. Pižurica, "Deep feature fusion via two-stream convolutional neural network for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 4, pp. 2615–2629, April 2020.
- [45] L. Ran, Y. Zhang, W. Wei, and Q. Zhang, "A hyperspectral image classification framework with spatial pixel pair features," *Sensors*, vol. 17, no. 10, 2017. [Online]. Available: <https://www.mdpi.com/1424-8220/17/10/2421>
- [46] M. He, B. Li, and H. Chen, "Multi-scale 3d deep convolutional neural network for hyperspectral image classification," in *2017 IEEE International Conference on Image Processing (ICIP)*, Sep. 2017, pp. 3904–3908.
- [47] C. Xiang, L. Zhang, Y. Tang, W. Zou, and C. Xu, "Ms-capsnet: A novel multi-scale capsule network," *IEEE Signal Processing Letters*, vol. 25, no. 12, pp. 1850–1854, Dec 2018.
- [48] W. Song, S. Li, L. Fang, and T. Lu, "Hyperspectral image classification with deep feature fusion network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 6, pp. 3173–3184, June 2018.
- [49] J. Zhu, L. Fang, and P. Ghamisi, "Deformable convolutional neural networks for hyperspectral image classification," *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 8, pp. 1254–1258, Aug 2018.

- [50] X. Cao, F. Zhou, L. Xu, D. Meng, Z. Xu, and J. Paisley, "Hyperspectral image classification with markov random fields and a convolutional neural network," *IEEE Transactions on Image Processing*, vol. 27, no. 5, pp. 2354–2367, May 2018.
- [51] B. Pan, Z. Shi, and X. Xu, "Mugnet: Deep learning for hyperspectral image classification using limited samples," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 145, pp. 108–119, 2018, deep Learning RS Data. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0924271617303416>
- [52] "Source code on github," <https://github.com/zenodhaene/HyperspectralClassification>, 2021.
- [53] "Konvajs," <https://github.com/react-bootstrap/react-bootstrap>, 2015.
- [54] "React bootstrap," <https://github.com/react-bootstrap/react-bootstrap>, 2011.
- [55] "Mongodb," <https://github.com/mongodb>, 2009.
- [56] ".net core," <https://github.com/dotnet/core>, 2016.
- [57] "scipio," <https://pypi.org/project/scipio/>, 2015.
- [58] "Tensorflow," <https://github.com/tensorflow/tensorflow>, 2015.