Silesian University of Technology

Silesian University of Technology

Optimization of deep learning network architectures for hyperspectral data classification

This thesis has been submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

mgr Kamil Książek

Supervisor:
dr hab. inż. Przemysław Głomb

Co-supervisor:
dr Krisztián Búza

Gliwice, October 2022

Abstract

In recent years, deep learning has achieved undisputed success in many domains, including the classification of hyperspectral images. These images are acquired by hyperspectral sensors that capture data in hundreds or thousands of narrow channels per pixel from various ranges of the electromagnetic spectrum. They measure energy from the region of visible light, the near-infrared, the short-wave infrared or even the long-wave infrared. Hyperspectral images have various applications in non-invasive substance classification, including geology, precision agriculture, environmental monitoring, hydrology and military science.

In this work, we focus on the problem of optimization of deep neural networks applied to the classification of hyperspectral data. Due to the variety and complexity of neural architectures, the network configuration for a particular problem poses a challenge. We chose the classification of blood and blood-like substances on a dataset of hyperspectral images as an experimental setting for the neural network optimization task. We applied several state-of-the-art architectures, i.e. one-, two- and three-dimensional convolutional neural networks, a recurrent network based on GRU units and a multilayer perceptron. We designed two different scenarios of experiments, i.e. a commonly used *Hyperspectral Transductive Classification* (HTC) scenario in which samples from both a training set and a test set come from one image and a more realistic *Hyperspectral Inductive Classification* (HIC) scenario in which a test set comes from a different source than a training set. In the HIC scenario, we also evaluated the influence of diverse background substances and varying days of image capture on network performance. We observed that for the HTC scenario, the classification accuracy of all methods tested exceeded 90%, while for the HIC scenario, the performance was much lower and varied between 57.2 and 99.5%. Furthermore, in some cases, more sophisticated methods such as convolutional neural networks were less efficient than the simplest feedforward architecture. We also identified a problem with the network stability for one of the architectures used in the experiments. The problem was more investigated in the following chapters and led to the study about vanishing gradients and, finally, to network reinitialization methods.

One of the main limitations of network performance in the HIC scenario is the presence of pixels containing a mixture of spectra of several substances. Therefore, for further studies, we selected a branch of neural network architectures, i.e. linear autoencoders. They are trained without labels, and, despite the simplicity, they can achieve competitive results. We conducted a series of hyperspectral unmixing experiments in which original spectra of mixed pixels are being reconstructed. We observed that, in some cases, trained networks with a given set of hyperparameters achieve different levels of performance. We prepared an extensive statistical study that confirmed the important impact of weight initialization on the final network reconstruction error. Furthermore, we identified a problem of dead activations in networks using the ReLU activation function, related to the large number of inactive neurons. This phenomenon makes in some situations training of such networks difficult or even impossible. We proposed three network reinitialization methods that alleviate the negative consequences of dead activations. Based on the threshold for ratios of dead activations, we reinitialize a subset or all network weights, depending on the selected reinitialization method. We confirm that for experiments with hyperspectral data, the proposed methods increase network performance. Finally, we evaluated the robustness of these approaches using the MNIST dataset. The observed results lead to the conclusion that network reinitialization methods are an effective solution and reduce network error, especially for suboptimal sets of hyperparameters.

The following dissertation contributes to the problem of optimization of deep learning networks for hyperspectral data classification through an extensive study of different architectures. Furthermore, the problem of network stability was discussed and three network reinitialization methods were proposed to mitigate the identified dead activation phenomenon.

Acknowledgements

I would like to thank everyone who contributed to the fact that I was able to write the following dissertation. I acknowledge all the people who accompanied me during my scientific career and who taught me topics related to mathematics, informatics and other disciplines. I would also like to thank all those with whom I wrote scientific publications, not only those related to the topic of my dissertation.

I would like to thank my supervisor, Przemysław Głomb. He taught me a lot about the way of conducting research and machine learning. He guided me and devoted his time to scientific discussions and advised me on how to properly solve research problems. I would like to acknowledge my co-supervisor, Krisztián Búza, for scientific discussions and all suggestions related to my work. I also acknowledge Prof. Joanna Polańska for her support and advice during my doctoral studies.

I would like to thank all my colleagues from the Institute of Theoretical and Applied Informatics of the Polish Academy of Sciences (ITAI PAS), especially Michał Romaszewski and Michał Cholewa, for their comments on my dissertation and their support during the whole period of my work at the Institute. I would like to acknowledge all the people I have collaborated with throughout my scientific career so far, including Marcin Woźniak, Dawid Połap, Wojciech Kempa and Krzysztof Grochla.

Furthermore, I would like to thank all the people who accompany me during my life, especially my parents Danuta and Ryszard, as well as Fr. Mieczysław Kożuch. I thank everyone who has helped me in any way in my life.

I acknowledge all the authors of the datasets that I used during my study, including my colleagues from ITAI PAS for preparing the dataset with samples of blood and blood-like substance. I thank the Florida Environmental Research Institute for making the Samson dataset available and NASA Jet Propulsion Laboratory for the Jasper Ridge dataset as well as AT&T for the MNIST dataset.

The dissertation was prepared as part of the work in ITAI PAS.

This work has been co-financed by the European Union through the European Social Fund (grant POWR.03.02.00-00-I029).

Contents

List of Figures

List of Tables

List of publications used in the dissertation

The parts of this dissertation are based on or contain fragments of the following published papers. Chapter 2 is based on [80] while Chapters 3–4 are based on and extend the study described in [79].

1. **K. Książek**, P. Głomb, M. Romaszewski, M. Cholewa, B. Grabowski, K. Buza, *Improving Autoencoder Training Performance for Hyperspectral Unmixing with Network Reinitialisation*, ICIAP 2022, In: S. Sclaroff, C. Distante, M. Leo, G.M. Farinella, F. Tombari, (eds) Image Analysis and Processing – ICIAP 2022, Lecture Notes in Computer Science, 2022, Vol. 13231, Springer, Cham, pp. 391-403, ISBN: 978-3-031-06426-5, DOI: 10.1007/978-3-031-06427-2_33.

   My contributions in [79]: The idea of using autoencoders for hyperspectral unmixing, the idea and design of network reinitialization methods, performing experiments, writing a source code, the selection of statistical tests, the preparation and verification results of experiments, participating in discussions and writing of the article (including the preparation of Figures 1–2).

2. **K. Książek**, M. Romaszewski, P. Głomb, B. Grabowski, M. Cholewa, *Blood Stain Classification with Hyperspectral Imaging and Deep Neural Networks*, Sensors, 2020, Vol. 20, Issue 22, No. 6666, pp. 1-24, ISSN 1424-8220, DOI: 10.3390/s20226666.

   My contributions in [80]: Preparation and performing experiments for deep learning networks, preparation of Figures: 1–7, 9, 12–13, writing a source code, the description of selected neural networks in the *Methods* section, taking part in discussions and writing other parts of the article.

Other publications

The following list presents selected other papers published by the Author that were not included in the dissertation:

1. M. Żarski, B. Wójcik, **K. Książek**, J. A. Miszczak, *Finicky transfer learning–A method of pruning convolutional neural networks for cracks classification on edge devices*, Computer-Aided Civil and Infrastructure Engineering, 2022, Vol. 37, Issue 4, pp. 500-515, ISSN 1093-9687, DOI: 10.1111/mice.12755.

2. K. Grochla, A. Strzoda, R. Marjasz, P. Głomb, **K. Książek**, Z. Łaskarzewski, *Energy-Aware Algorithm for Assignment of Relays in LP WAN*, ACM Transactions on Sensor Networks, 2022, Association for Computing Machinery, New York, pp. 1-23, ISSN 1550-4859, DOI: 10.1145/3544561.

3. W. M. Kempa, **K. Książek**, R. Marjasz, *On Time-Dependent Queue-Size Distribution in a Model With Finite Buffer Capacity and Deterministic Multiple Vacations With Applications to LTE DRX Mechanism Modeling*, IEEE Access, 2021, Vol. 9, pp. 148374-148383, DOI: 10.1109/ACCESS.2021.3123897.

4. **K. Książek**, K. Grochla, *Flexibility Analysis of Adaptive Data Rate Algorithm in LoRa Networks*, 2021 International Wireless Communications and Mobile Computing (IWCMC), 2021, pp. 1393-1398, DOI: 10.1109/IWCMC51323.2021.9498664.

5. F. Pałka, W. Książek, P. Pławiak, M. Romaszewski, **K. Książek**, *Hyperspectral Classification of Blood-Like Substances Using Machine Learning Methods Combined with Genetic Algorithms in Transductive and Inductive Scenarios*, Sensors, 2021, Vol. 21, Issue 7, No. 2293, pp. 1-18, ISSN 1424-8220, DOI: 10.3390/s21072293.

6. W. M. Kempa, **K. Książek**, *On transient queue-size distribution in a finite-buffer model with threshold waking and early setup policy*, Performance Evaluation, 2020, Vol. 140-141, pp. 102107, ISSN 0166-5316, DOI: 10.1016/j.peva.2020.102107.

7. **K. Książek**, K. Grochla, *Aggregation of GPS, WLAN, and BLE Localization Measurements for Mobile Devices in Simulated Environments*, Sen-

sors, 2019, Vol. 19, Issue 7, No. 1694, pp. 1-15, ISSN 1424-8220, DOI: 10.3390/s19071694.

8. M. Woźniak, **K. Książek**, J. Marciniec, D. Połap, *Heat production optimization using bio-inspired algorithms*, Engineering Applications of Artificial Intelligence, 2018, Vol. 76, pp. 185-201, ISSN 0952-1976, DOI: 10.1016/j.engappai.2018.09.003.

9. **K. Książek**, Z. Marszałek, G. Capizzi, C. Napoli, D. Połap, M. Woźniak, *The Impact of Parallel Programming on Faster Image Filtering*, 2018 Federated Conference on Computer Science and Information Systems (FedCSIS), In: Annals of Computer Science and Information Systems, 2018, Vol. 15, pp. 545-550, DOI: 10.15439/2018F71.

10. M. Woźniak, **K. Książek**, D. Połap, *Benchmark tests on heuristic methods in the darts game*, International Journal of Electronics and Telecommunications, 2018, Vol. 64, No. 2, pp. 115-121, DOI: 10.24425/119358.

11. D. Połap, K. Kęsik, **K. Książek**, M. Woźniak, *Obstacle Detection as a Safety Alert in Augmented Reality Models by the Use of Deep Learning Techniques*, Sensors, 2017, Vol. 17, Issue 12, No. 2803, pp. 1-16, DOI: 10.3390/s17122803.

12. **K. Książek**, D. Połap, M. Woźniak, R. Damaševičius, *Radiation heat transfer optimization by the use of modified ant lion optimizer*, 2017 IEEE Symposium Series on Computational Intelligence (SSCI), 2017, pp. 1-7, DOI: 10.1109/SSCI.2017.8280853.

13. **K. Książek**, W. Masarczyk, I. Nowak, *Heuristic approach to the game of darts by using genetic algorithm and ant colony optimization*, Proceedings of the International Conference for Young Researchers in Informatics, Mathematics and Engineering (ICYRIME 2017), In: CEUR, 2017, Vol. 1852, pp. 33-38.

14. **K. Książek**, Z. Marszałek, *Combinatorial Summation Primes: a Discussion of Different Methods to Solve the Problem*, SYSTEM 2016: Symposium for Young Scientists in Technology, Engineering and Mathematics, In: CEUR, 2016, Vol. 1730, pp. 25-34.

Notations

For convenience, the notation used in the following chapters of the dissertation will be presented:

Notations used in Chapter 1

B	a number of spectral bands
w	a width of the hyperspectral image
l	a length of the hyperspectral image
$\mathcal{X}$	input data space
$\mathcal{Y}$	data label space
$\boldsymbol{\theta}^{(i)}$	the parameters of the i–th layer of the neural network
f	a mapping function: $\mathcal{X} \to \mathcal{Y}$
$f^{(i)}$	a mapping function in the i–th layer of the neural network
$\hat{y}$	the output of the network
n_i	a number of neurons in the i–th network layer
$\mathbf{W}^{(i)}$	the weights in the i–th network layer
$\mathbf{X}^{(i)}$	the output data of the i–th network layer
$\mathbf{b}^{(i)}$	the bias vector of the i–th network layer
$\phi(\cdot)$	an activation function
$*$	a convolution operation
$\mathbf{K}$	a two-dimensional kernel (filter)
$\mathbf{C}$	the output of the convolution operation (a feature map)
$x_{l,m,s}^{(i,k)}$	the value at position (l, m, s) of the k–th feature map of the i–th network layer
$w_{l',m',s'}^{(i,k,c)}$	the value at position (l', m', s') of the k–th kernel of the i–th network layer connected to the c–th feature map of the $(i-1)$–th network layer
$b^{(i,k)}$	the bias of the k–th feature map of the i–th network layer
$h^{(i)}$	a height of the feature maps of the i–th network layer
$w^{(i)}$	a width of the feature maps of the i–th network layer
$d^{(i)}$	a depth of the feature maps of the i–th network layer
$C^{(i)}$	a number of kernels in the i–th network layer
$\mathbf{x_t}$	an input data sample for RNNs at time $t \geq 0$
$\mathbf{W_x}$	a weight matrix for the input data in RNNs
$\mathbf{W_y}$	a weight matrix for the output data from the previous time step in RNNs
$\mathbf{h_t}$	a hidden state at time $t \geq 0$
$\mathbf{W_{xh}}$	an input-hidden matrix of weights in RNNs
$\mathbf{W_{hh}}$	a hidden-hidden matrix of weights in RNNs

$\mathbf{W_{hy}}$	a hidden-output matrix of weights in RNNs
p	a size of the input data sample in RNNs
q	a number of neurons in RNNs
h	a number of features in the hidden state
$\mathbf{u_t}$	an update gate
$\mathbf{r_t}$	a reset gate
$\mathbf{a_t}$	an activation gate
$\mathbf{W_{xu}}$	a matrix of weights for connections between the input and the update gate
$\mathbf{W_{hu}}$	a matrix of weights for connections between the hidden state and the update gate
$\mathbf{W_{xr}}$	a matrix of weights for connections between the input and the reset gate
$\mathbf{W_{hr}}$	a matrix of weights for connections between the hidden state and the reset gate
$\mathbf{W_{xa}}$	a matrix of weights for connections between the input and the activation gate
$\mathbf{W_{ha}}$	a matrix of weights for connections between the hidden state and the activation gate
$\mathbf{b_u}$	the bias of the update gate
$\mathbf{b_r}$	the bias of the reset gate
$\mathbf{b_a}$	the bias of the activation gate
$\otimes$	the Hadamard product
$\mathbf{h}$	a hidden representation of the autoencoder
$\mathbf{d}$	a decoder
L	a loss function
n	a size of the input data sample in autoencoders

Notations used in Chapter 2

C	a regularization parameter in SVMs
γ	a kernel coefficient in SVMs
B	a number of the spectral bands
X_{var}	a variance of the dataset
c	a size of the convolution matrix in 1D CNN [65]
p	a size of the pooling matrix in 1D CNN [65]
f_j	the activation value of the j–th filter in 2D CNN [83]
$\tilde{f}_j$	a local response normalization of the j–th filter in 2D CNN [83]
$F(i)$	a *frame* scene from the i–th day
$E(i)$	a *comparison* scene from the i–th day
$\mathbf{X_{train}}$	a training set
$\mathbf{X_{test}}$	a test set
N	a total number of classes in the training set
$n_{samples}^{min}$	a number of samples from the least numerous class in the training set
n_{train}	a number of samples in the training set
OA	overall accuracy
AA	average accuracy
κ	Cohen's κ coefficient

Notations used in Chapter 3

$\mathcal{U}$	a neural network with n hidden layers
u_0	the input size
u_{n+1}	the output layer size
u_i	the number of neurons in the i–th layer, $i \in \{1, ..., n\}$
$\mathbf{y_j}$	the output of the j–th layer
$\mathbf{b_j}$	the bias of the j–th layer
$\mathbf{x_{j-1}}$	the input of the j–th layer, the output of the $(j-1)$–th layer
$\mathbf{P_j}$	the weights of the j–th layer
$g(\cdot)$	an activation function
$Var(\cdot)$	a variance of the random variable
$\mathbf{E}(\cdot)$	an expected value of the random variable
$f(y_j)$	a probability density function of the random variable y_j
$\tilde{\mathbf{P}}_\mathbf{j}$	the rearranged matrix of the network weights
Δx_j	the values of gradients in neurons of the j–th layer
Δy_j	the values of gradients in the outputs of neurons of the j–th layer
B	a number of spectral bands
S	a number of endmembers
P	a number of hyperspectral pixels
$\mathbf{x}$	a hyperspectral pixel
$\mathbf{e_i}$	the i–th endmember
$\mathbf{a}$	the fractional abundance vector
$\mathbf{X}$	a hyperspectral image containing B–dimensional spectra of P pixels
$\mathbf{E}$	a matrix of S endmembers
$\mathbf{A}$	a matrix of fractional abundances for P pixels
$\mathbf{W}$	a noise matrix
$\mathcal{B}$	a batch of B–dimensional vectors
b_s	a batch size
$\mathcal{B}_{act}^k$	a set of b_s activation values for the k–th neuron
$\mu_{\mathcal{B}}^k$	a mean value of activations for the k–th neuron
$\sigma_{\mathcal{B}}^k$	a variance of activations for the k–th neuron

M	a number of neurons
${}_{(BN)}\tilde{g_i}^k$	a batch normalization transform for the i–th activation value of the k–th neuron
${}_{(BN)}\mathbf{\tilde{g}_i}$	a vector of transformed neurons values using batch normalization technique for the i–th batch sample
${}_{(ST)}\mathbf{\tilde{g}_i}$	a vector of transformed neurons values using soft thresholding technique for the i–th batch sample
$\boldsymbol{\alpha}$	a vector of trainable parameters in soft thresholding
${}_{(norm)}\tilde{g_i}^j$	the j–th activation value after the application of sum-to-one normalization, calculated for the i–th batch sample
${}_{(GD)}\mathbf{\tilde{g}_i}$	a vector of values after the application of Gaussian Dropout for the i–th batch sample
K_j^i	the j–th autoencoder model of the i–th set of initial weights
N	a number of different initial sets of weight values
R	a number of training runs
$\mathbf{Y}$	a matrix of the reconstructed image
$\mathbf{E_{GT}}$	a ground truth matrix of endmembers
$\mathbf{E_{rec}}$	a matrix of reconstructed endmembers
$\mathbf{A_{GT}}$	a ground truth matrix of fractional abundances
$\mathbf{A_{rec}}$	a matrix of reconstructed fractional abundances

Notations used in Chapter 4

$\mathcal{U}$	a neural network with n hidden layers
u_0	the input size
u_{n+1}	the output layer size
u_i	the number of neurons in the i–th layer, $i \in \{1, ..., n\}$
$\mathbf{P_j}$	the weights of the j–th layer
$\mathbf{b_j}$	the bias of the j–th layer
$\mathbf{g_j}(\mathbf{x})$	a vector of values of the ReLU activation function for the j–th layer, $j \geq 1$, where x is an input vector
$\mathbf{G}(\mathbf{x})$	a matrix of neurons activations from the whole network for an input vector $\mathbf{x}$
$\mathbf{P_j}[\cdot, \mathbf{i}]$	the weights (parameters) between all neurons in the $(j-1)$–th layer and the i–th neuron of the j–th layer
$\mathfrak{D}$	a dataset
P	a number of input points
B	a dimension of input points
$\mathfrak{G}$	an array of network activations for P input vectors
$\mathfrak{G}^j$	a matrix of activations of the j–th layer for all input instances
d^j_{dead}	a dead activations' coefficient for the j–th network layer
$d^{j,i}_{dead}$	a dead activations' coefficient for the i–th neuron of the j–th network layer
$\mathcal{N}^j_0$	a number of zero activations for the j–th layer
$\mathcal{N}^{j,i}_0$	a number of zero activations for the i–th neuron of the j–th network layer
$\overline{RMSE_i}$	mean RMSE for the i–th network model
$\mathbf{X}$	a matrix of input vectors
$\tilde{\mathbf{X}}$	a matrix of reconstruction of the input vectors
$RMSE_{i,j}(\mathbf{X}, \tilde{\mathbf{X}})$	RMSE for the j–th training run of the i–th model, calculated at the end of the training session of the j–th run between $\mathbf{X}$ and $\tilde{\mathbf{X}}$
t	a reinitialization threshold
b_s	a batch size
M	a number of network models
$arch$	a network architecture
l_r	a learning rate

Notations used in Appendix

K	a number of data samples
X_i	the i–th random sample
k_i	a number of observations in the i–th random sample
M	a total number of observations
$\mathbf{E}(\cdot)$	an expected value of the random variable
H	a test statistic of the Kruskal-Wallis H-test
R_i	the sum of the ranks for the observations from the i–th sample
S^2	the variance of ranks for populations with ties
r_s	a Spearman's rank correlation coefficient
$\mathcal{X}$	a random variable
$\mathcal{Y}$	a random variable
F	a cumulative distribution function
G	a cumulative distribution function

Chapter 1

Introduction

Artificial intelligence (AI) is an rapidly developing research field [51]. Intelligence systems are aimed at industry process automatization, object detection or classification in images, etc. Due to the fact that it is not easy to precisely describe processes taking place in the world, one can create machine learning systems that gain own knowledge based on patterns from raw data. The efficiency of such systems is, however, related to the data representation and its features. In the approach called representation learning, algorithms not only map the representation to output but also learn the intrinsic data representation [51].

The important aspect of this dissertation is deep learning (DL), which is a class of methods that enables machines to discover the representation of data [81] and to express it using a series of simpler representations [51]. It is a subfield of machine learning inspired by neuroscience [51, 118]. The basic DL models are feedforward neural networks, also called multilayer perceptrons (MLPs). Through the composition of multiple non-linear modules, they create a more abstract data representation in consecutive hidden layers. The weights (parameters) of hidden layers are not directly given in the input data, but they are learned during the training procedure. The observed data values are provided to the DL model through the input layer.

DL algorithms are efficient in discovering complex features in high-dimensional datasets which make them useful in many research areas. They achieved successes in many domains, such as image classification, which aims at assigning one of the several defined classes to the given image [77] or object detection [10, 129] which relies on the detection of all instances of objects among one or more classes in images and finding their exact locations [3]. Another task for DL methods is image segmentation, i.e. pixel classification or individual object partitioning [105]. In recent years, deep learning architectures have also proven to be efficient in natural language processing (NLP) tasks [165] including sentiment analysis [170], speech recognition [52] or language modeling [146].

Various DL applications are connected with medicine, i.e. tumor identification [126], prevention of neurological diseases, e.g. Alzheimer's disease [164], cardiovascular issues such as heart attacks [21], lung diseases [12], celiac disease detection [155], etc. They are also used for the analysis of biosignals like electrocardiograms (ECG, [30]) or encephalograms (EEG, [64]) and in biomedicine, i.e. for the prediction of missing values of DNA structure [45] or drug discovery [162].

Another important part of this dissertation is hyperspectral imaging (HSI) [118], also called imaging spectroscopy. In HSI sensors collect hundreds or thousands of narrow, neighboring bands per pixel [19, 139]. Hyperspectral data samples are high-dimensional vectors, that is a single spectral vector $\mathbf{x} \in \mathbb{R}^B$ is an element in the B-dimensional space, where B is a number of spectral bands, $B \gg 1$. Hyperspectral sensors differ from multispectral sensors in that the number of spectral bands is higher and they are contiguous [140]. The result of the hyperspectral data capture (the acquisition) is a hyperspectral image. They are three-dimensional arrays (*cubes*) of $w \times l \times B$ size in which the width w and the length l of the image correspond to its spatial dimensions while B is the spectral dimension. The illustration of a sample hyperspectral dataset is presented in Figure 1.1. This picture shows the scheme of a three-dimensional cube, a false-color RGB image and a spectrum of an exemplary pixel of this dataset.

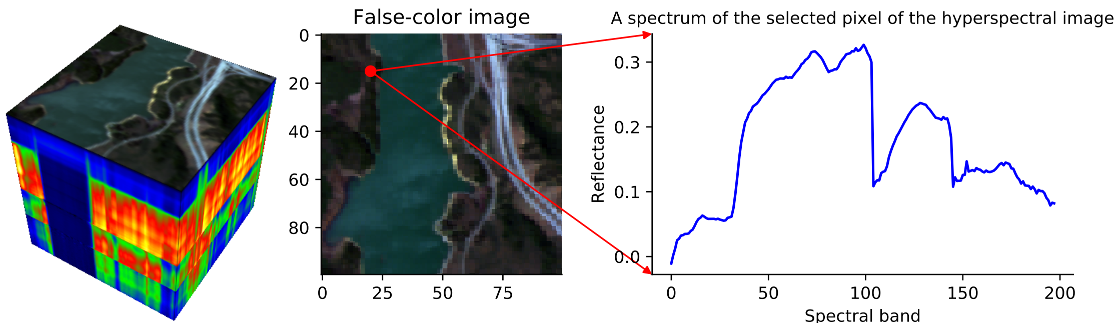


Figure 1.1: The visualization of an exemplary hyperspectral dataset. On the left side, a three-dimensional cube of the Jasper Ridge dataset with the reflectance of edge pixels is presented. The lowest reflectance values are marked in blue while the highest are in red. In the middle, a false-color RGB composite is depicted while on the right side a spectrum of the selected pixel is shown.

In the case of remote sensing applications, hyperspectral data usually come from satellites or airborne platforms [140]. They can be used for environmental [141] or agricultural monitoring [95], terrain classification [118], food safety [123], forensic [133] or biomedical applications [56] and others. Recently, DL architectures have been successfully applied for HSI processing, e.g. for hyperspectral data classification [13], semantic segmentation [149] or hyperspectral unmixing [16].

However, hyperspectral data are associated with various difficulties like spectral mixing of pixels, noise and atmospheric effects related to the acquisition process as well as high-dimensionality of the images. In the case of mixed pixels, spectral unmixing algorithms are applied which aim to decompose such pixels into a set of pure spectra, i.e. endmembers, and a set of coefficients indicating the proportions of consecutive endmembers in a given pixel, i.e. fractional abundances [140].

1.0.1 Thesis

In this dissertation, we[1] worked on the optimization of deep learning network architectures for hyperspectral data classification and unmixing. We analyzed several architectures, including multilayer perceptrons and autoencoders as well as more sophisticated ones like convolutional neural networks. We performed the optimization using various datasets, including a dataset for blood stain classification and datasets for unmixing. Furthermore, after the observation of some undesirable states of selected networks, we proposed three network reinitialization methods. They prepare the reinitialization of some or all network weights to improve the selected network training run. We also studied their impact on the performance of neural networks, focusing mainly on autoencoders. We stated the following thesis:

Thesis statement: *Optimization of deep learning network architectures and weight reinitialization methods improve the performance of neural networks for hyperspectral data.*

We will discuss the dissertation thesis in the next chapters.

1.0.2 Overview of the work

We started the analysis with the issue of the selection of neural network architecture in hyperspectral data classification. We chose a dataset with blood and blood-like substances as an environment for experiments with the use of several deep learning architectures designed for HSI classification. We studied different scenarios of data representation in training sets: transductive and inductive. In the first case, the training set came from the same dataset as the test set while in the second case, the training and the test samples were from different images; thus this scenario was more challenging. The conclusions from this part of the research prompted us to study the spectral mixing phenomenon, which made the classification task more difficult. We decided to tackle this problem through the use of a special type of neural network, i.e. autoencoders. Their design enables

[1]From that moment on, the first-person plural will be used.

the extraction of endmembers and fractional abundances in an unsupervised way, i.e. without pixel labels.

During the experiments, we noticed problems with the training stability of the selected autoencoder models. We decided to investigate them in more detail. We performed a series of experiments for different autoencoder setups, including network hyperparameters, weight initialization approaches, datasets, loss functions, etc. Through statistical tests, we verified that the initial state of the network, i.e. set of initial weights, has an influence on the further network performance. Moreover, we observed some models that performed poorly, despite using the same set of hyperparameters as in the remaining cases. After an in-depth analysis, which was confirmed by the results of the Spearman's rank correlation coefficient, we identified the dead activations phenomenon for a subset of DL architectures. It relies on the loss of importance of certain neurons expressed in zero contribution of selected network weights to further calculations. We pointed out why it is not desirable during the network training and we tried to find methods of mitigating the negative impact of this phenomenon. Therefore, we proposed three network reinitialization methods that aim to detect its occurrence and reinitialize some or all weights of the network. We evaluated the methods through comparisons of the network performance without and with the application of the proposed algorithms. We conducted the study focusing on hyperspectral datasets but we also extended the experiments to the well-known MNIST dataset [82].

The next sections will introduce the types of neural network used in further chapters, i.e. multilayer perceptrons, convolutional and recurrent neural networks, as well as autoencoders. Furthermore, the basic concepts of hyperspectral imaging will be discussed and the outline of all subsequent chapters of the dissertation.

1.1 Neural networks

1.1.1 Multilayer perceptrons

Multilayer perceptrons are directed acyclic graphs, which means that they have connections between units (called also nodes or neurons) only in one direction [134]. The sign and strength of the connection are defined by the value of the weight assigned to it. In the case of MLPs, there are no loops between connections.

The main goal of these architectures is the approximation of the function $\tilde{f}$ to solve a classification or regression problem. Let us suppose that f is a mapping function, $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathbf{x} \in \mathcal{X}$ is an input data sample and $y \in \mathcal{Y}$ is a category or a continuous value, depending on the defined problem. The network produces an output $\hat{y} = f(\mathbf{x}, \boldsymbol{\theta})$ where its parameters are defined by $\boldsymbol{\theta}$ to minimize a difference between the obtained and the desired value. The output can also be expressed in terms of sub-mapping functions, i.e.:

$$\hat{y} = f(\mathbf{x}, \boldsymbol{\theta}) = \bar{f}\Bigg(f^{(n)}\Bigg(f^{(n-1)}\Big(\cdots\Big(f^{(1)}(\mathbf{x}, \boldsymbol{\theta}^{(1)})\cdots\Big), \boldsymbol{\theta}^{(n-1)}\Big), \boldsymbol{\theta}^{(n)}\Bigg)\Bigg). \tag{1.1}$$

For $i \in \{1, ..., n\}$, $f^{(i)}$ is the i–th network layer while $\theta^{(i)}$ represents the parameters of this layer. Each layer consists of neurons and the number of neurons in the i–th layer, n_i, is its width. The output layer $\bar{f}$ prepares the final classification or regression step. The total number of layers is denoted as the network depth. A single neuron of a selected layer receives values from the units of the previous layer units and applies an activation function. A general form of the i–th layer response is as follows:

$$\mathbf{X}^{(i)} = f^{(i)}(\mathbf{X}^{(i-1)}, \mathbf{W}^{(i)}, \mathbf{b}^{(i)}) = \phi\Big((\mathbf{W}^{(i)})^{\top} \cdot \mathbf{X}^{(i-1)} + \mathbf{b}^{(i)}\Big), \tag{1.2}$$

where $\mathbf{W}^{(i)} \in \mathbb{R}^{n_{i-1} \times n_i}$ denotes the weights and $\mathbf{b}^{(i)} \in \mathbb{R}^{n_i}$ represents a bias vector of the i–th layer. $\mathbf{X}^{(i-1)} \in \mathbb{R}^{n_{i-1}}$ is the output data of the $(i-1)$–th layer which is also an input into the i–th network layer while $\phi(\cdot)$ is the activation function applied element-wise. The weights $\mathbf{W}^{(i)}$ and the biases $\mathbf{b}^{(i)}$ together are the parameters of the i–th layer, $\boldsymbol{\theta}^{(i)}$. In recent years, the activation function most frequently used in DL has become the Rectifier Linear Unit (ReLU) [125], which is expressed by the formula $\phi(x) = \max(0, x)$, and its generalizations, for example, leaky or parametric ReLU [51]. The layer presented in MLPs is called fully-connected because all neurons in the $(i-1)$–th layer are connected to all neurons in the i–th layer which means that there are $n_{i-1} \cdot n_i + n_i$ parameters that have to be optimized.

Generally in images, including HSI, two adjacent pixels are strongly correlated [38, 101]. Therefore, the use of spatial dependencies can improve the efficiency of feature extraction. In MLPs these relationships are lost because the input of the network is a one-dimensional vector and networks cannot gain benefits from relative pixel positions.

General remarks

During the training phase, the activations propagate from the input through hidden layers to the network output $\hat{y}$. This process is called forward propagation [51]. In the next step, the back-propagation algorithm is responsible for computing the error with respect to each network parameter. Then, the learning algorithm, e.g. stochastic gradient descent or Adam optimizer [72] modifies network weights, according to the calculated errors. DL models act as feature extractors in which various sub-mapping functions learn different input characteristics [118]. Neural networks are able to extract non-linear data features.

1.1.2 Convolutional neural networks

Convolutional neural networks (CNNs) are a special kind of neural network architectures designed to process multidimensional data like 2D images or 3D hyperspectral images, as well as one-dimensional time series or other data types. In the case of CNNs, a matrix multiplication in at least one layer is replaced by a special linear operation called convolution [51]. The pattern for convolution ($*$) between a two-dimensional kernel (also called filter) $\mathbf{K}$ and a two-dimensional image $\mathbf{X}$ is expressed as follows [27]:

$$\mathbf{C}(p,q) = (\mathbf{K} * \mathbf{X})(p,q) = \sum_{(i,j)\in\mathbf{K}_I} \mathbf{X}(p-i, q-j)\mathbf{K}(i,j), \tag{1.3}$$

where $\mathbf{K}_I$ is a set of coordinates of the filter $\mathbf{K}$. The set of indices $\mathbf{K}_I$ is dependent on the width and length of the kernel. For example, when $\mathbf{K} \in \mathbb{R}^{5\times5}$, Equation 1.3 can be rewritten in the following way:

$$\mathbf{C}(p,q) = (\mathbf{K} * \mathbf{X})(p,q) = \sum_{i=-2}^{2}\sum_{j=-2}^{2} \mathbf{X}(p-i, q-j)\mathbf{K}(i,j). \tag{1.4}$$

According to the CNNs terminology, the output of a convolution is called a feature map.

Unlike MLPs, CNNs do not have connections between all neurons of the current layer and all neurons of the preceding layer, but they have sparse connectivity [51]. More specifically, it means that a block of neurons, i.e. a kernel, that is smaller than the input image, is applied over small regions of the input data. The same kernel can be used for different regions of the image. It is not necessary to create separate filters having the same function (e.g. detecting the horizontal edges) for various places of the image. Due to this property, we can reduce both the memory complexity and perform fewer operations.

Not only is it possible to detect small but important data features, like edges, but the same kernel can be used for other regions of the image.

In the most general considered case, i.e. 3D CNNs, consecutive values of feature maps can be calculated in the following way [68, 91, 118, 137]:

$$x_{l,m,s}^{(i,k)} = \phi\Big(\sum\nolimits_{c=1}^{C^{(i-1)}} \sum\nolimits_{l'=1}^{h^{(i)}} \sum\nolimits_{m'=1}^{w^{(i)}} \sum\nolimits_{s'=1}^{d^{(i)}} w_{l',m',s'}^{(i,k,c)} \cdot x_{l+l',m+m',s+s'}^{(i-1,c)} + b^{(i,k)}\Big), \tag{1.5}$$

where $C^{(i-1)}$ is the number of feature maps (cubes) in the preceding layer, $x_{l,m,s}^{(i,k)}$ is the value at position (l,m,s) of the k–th feature map of the i–th network layer, $w_{l',m',s'}^{(i,k,c)}$ is the value at position (l',m',s') of the k–th kernel of the i–th layer connected to the c–th feature map of the previous layer, $b^{(i,k)}$ is the bias for the corresponding feature map, $(h^{(i)}, w^{(i)}, d^{(i)})$ are height, width and depth of the feature maps of the i–th layer, respectively, while $\phi(\cdot)$ is the activation function, for example ReLU. Generally, we can express the output of the i–th convolutional layer as follows:

$$\mathbf{X}^{(i)} = \phi\Big(\mathbf{W}^{(i)} * \mathbf{X}^{(i-1)} + \mathbf{b}^{(i)}\Big)_{C^{(i)} \times h^{(i)} \times w^{(i)} \times d^{(i)}}, \tag{1.6}$$

where $\mathbf{X}^{(i)}$ is the output of the i–th layer, $\mathbf{W}^{(i)}$ is the weight matrix and $\mathbf{b}^{(i)}$ is the bias of this layer, respectively. $C^{(i)}$ denotes the number of kernels in the i–th layer while $*$ is a convolution operation. This approach allows extracting both spectral and spatial relationships from the input data (in the case of 2D or 3D CNNs). The convolution operation schemes in 2D and 3D CNNs are presented in Figures 1.2–1.3, respectively. In the first case, the result of this operation is two-dimensional while in the second case it is three-dimensional. A single network layer usually consists of multiple feature maps.

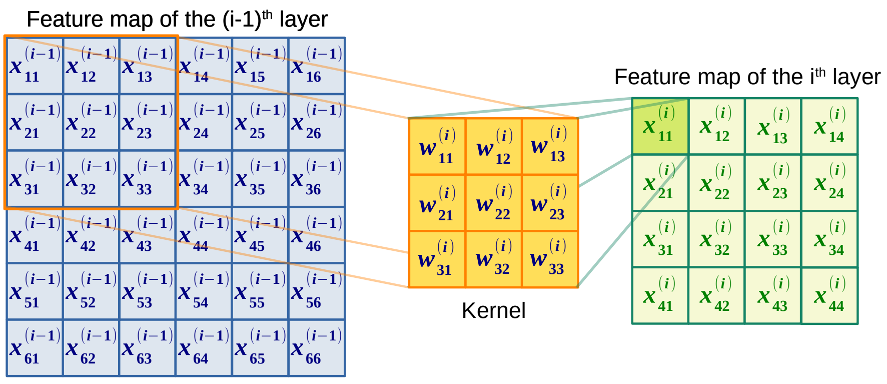

Feature map of the (i-1)th layer
Feature map of the ith layer
Kernel
$x^{(i-1)}_{11}$ … $x^{(i-1)}_{66}$
$w^{(i)}_{11}$ … $w^{(i)}_{33}$
$x^{(i)}_{11}$ … $x^{(i)}_{44}$

Figure 1.2: The scheme of the two-dimensional convolution (based on Figure 2 in [118]).

Another important layer in CNNs is connected to a pooling function [51]. It works in such a way that for a given location a selected statistic is calculated based on values from its neighborhood. Two popular pooling functions calculate either a maximum within a rectangular neighborhood (max pooling [173]) or an average value of adjacent outputs (average pooling), but usually a maximum operation is selected. It helps the network remain invariant to small input data translations, rotations and scaling.

Deep neural networks learn more general data features in the first layers while the further ones try to find more sophisticated and abstract features, more dependent on the current application. In the case of images and CNNs, the first convolutional layers learn basic features like edges or simple patterns [107]. Along

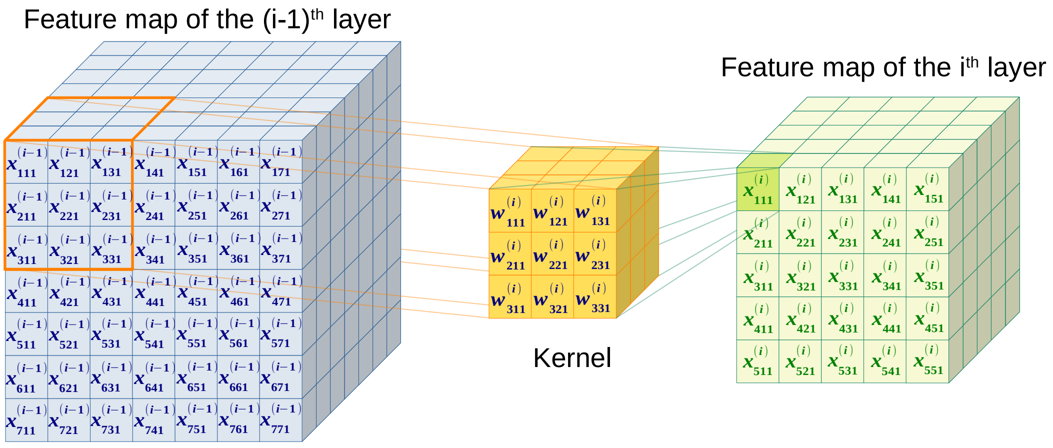

Feature map of the (i-1)th layer
Feature map of the ith layer
Kernel

Figure 1.3: The scheme of the three-dimensional convolution (based on Figure 2 in [169]).

with successive convolutional layers, networks develop more complex patterns up to the last layers where whole or parts of objects are formed. It means that the level of abstraction increases with the depth of a given network [75]. Finally, fully-connected layers, which are typically the last step of the CNNs pipeline, perform a prediction based on the extracted high-level features.

1.1.3 Recurrent neural networks

Recurrent neural networks (RNNs) create output at a given time step using input data from this time step and the output from the preceding one, which means that the network directly uses the context information. This construction advantages the application of RNNs for time series, sequences and the prediction of future events based on the previous ones [118].

Let us suppose that $\mathbf{x_t} \in \mathbb{R}^p$ is an input data sample at time $t \geq 0$, $\mathbf{W_x} \in \mathbb{R}^{p\times q}$ is a weight matrix for the input data, q is the number of neurons, while $\mathbf{W_y} \in \mathbb{R}^{q\times q}$ is a weight matrix for the output data from the previous time step. Furthermore, $\mathbf{b} \in \mathbb{R}^q$ is a bias vector. Then, the output of the RNN cell at time step t can be expressed by the following equation:

$$\mathbf{y_t} = \begin{cases} 0 & t = 0, \\ \phi(\mathbf{W_x}^\top \cdot \mathbf{x_t} + \mathbf{W_y}^\top \cdot \mathbf{y_{t-1}} + \mathbf{b}) & t > 0. \end{cases} \tag{1.7}$$

$\phi(\cdot)$ is the activation function, $\mathbf{y_t} \in \mathbb{R}^p$. One of the most commonly used activation functions in RNNs is the hyperbolic tangent [48]. In such simple situations, we say that the output of the network is also the hidden state, $\mathbf{h_t} = \mathbf{y_t}$, but in more complicated cases the output of a cell is a function of the hidden state of the previous time step and a current input vector, $\mathbf{h_t} = g(\mathbf{h_{t-1}}, \mathbf{x_t})$ [6, 48]. It means that the hidden state of a given cell can be different from the output of this cell, i.e.:

$$\begin{aligned} \mathbf{h_t} &= \phi(\mathbf{W_{xh}^\top} \cdot \mathbf{x_t} + \mathbf{W_{hh}} \cdot \mathbf{h_{t-1}}), \\ \mathbf{y_t} &= \mathbf{W_{hy}^\top} \cdot \mathbf{h_t}, \end{aligned} \tag{1.8}$$

where $\mathbf{W_{xh}} \in \mathbb{R}^{p \times q}$ is an input-hidden matrix, $\mathbf{W_{hh}} \in \mathbb{R}^{q \times q}$ is a hidden-hidden matrix and $\mathbf{W_{hy}} \in \mathbb{R}^{q \times p}$ is a hidden-output matrix.

The basic RNN architectures suffer from the problem of vanishing or exploding gradients. Furthermore, in the case of high-dimensional data, they have difficulties learning long-term dependencies. Solutions to these problems are gated RNNs with internal recurrence connections like long short-term memory (LSTM, [61]) and gated recurrent units (GRUs, [31]). They have units controlling the flow of information like hidden and cell states as well as input, output and forget gates. They allow the network to forget old information and remember new ones as necessary. GRUs networks are considered simplified LSTMs because the number of units is reduced, which translates into fewer parameters. Regardless of reducing the gates, the performance of GRUs is comparable to basic LSTMs [53]. In GRUs, a single unit controls the update and the reset gate and there is no output gate. The calculations for GRU cells are performed as follows [1, 32, 48]:

$$\mathbf{u_t} = \sigma(\mathbf{W_{xu}^\top} \cdot \mathbf{x_t} + \mathbf{W_{hu}^\top} \cdot \mathbf{h_{t-1}} + \mathbf{b_u}), \tag{1.9}$$

$$\mathbf{r_t} = \sigma(\mathbf{W_{xr}^\top} \cdot \mathbf{x_t} + \mathbf{W_{hr}^\top} \cdot \mathbf{h_{t-1}} + \mathbf{b_r}), \tag{1.10}$$

$$\mathbf{a_t} = \tanh\Big(\mathbf{W_{xa}^\top} \cdot \mathbf{x_t} + \mathbf{W_{ha}^\top} \cdot (\mathbf{r_t} \otimes \mathbf{h_{t-1}}) + \mathbf{b_a}\Big), \tag{1.11}$$

$$\mathbf{h_t} = \begin{cases} 0 & t = 0, \\ \mathbf{u_t} \otimes \mathbf{h_{t-1}} + (1 - \mathbf{u_t}) \otimes \mathbf{a_t} & t > 0, \end{cases} \tag{1.12}$$

where $\mathbf{x_t} \in \mathbb{R}^p$ is an input data sample of p features, $\mathbf{u_t}$ is the update gate, $\mathbf{r_t}$ is the reset gate, $\mathbf{a_t}$ is the candidate activation gate, $\mathbf{W_{xu}}, \mathbf{W_{xr}}, \mathbf{W_{xa}} \in \mathbb{R}^{p \times h}$ are weight matrices for connections between input and update, reset and activation gates, respectively; h is the number of features in the hidden state, $\mathbf{W_{hu}}, \mathbf{W_{hr}}, \mathbf{W_{ha}} \in \mathbb{R}^{h \times h}$ are weight matrices for connections between the previous hidden state and the update gate, the reset gate and the candidate activation, respectively. $\mathbf{b_u}, \mathbf{b_r}, \mathbf{b_a} \in \mathbb{R}^h$ are biases, $\sigma(\cdot)$ is a logistic sigmoid function while $\otimes$ is the Hadamard product. Finally, $\mathbf{y_t} = \mathbf{h_t}$. The symbolic representation of a GRU cell is presented in Figure 1.4.

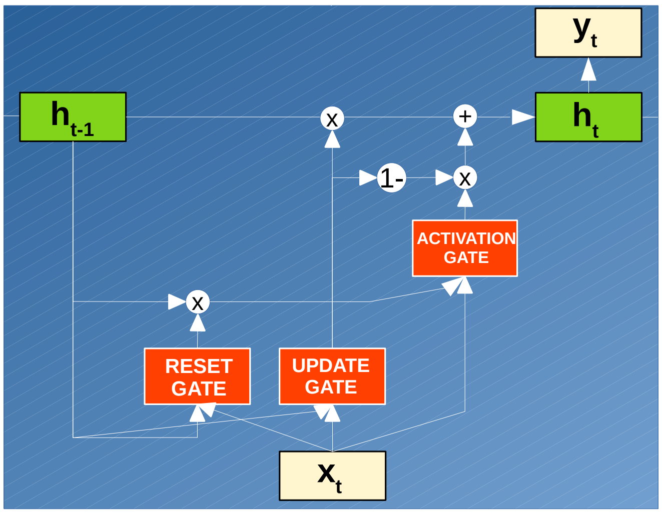


Figure 1.4: The scheme of the Gated Recurrent Unit cell (based on [48]). $\mathbf{x_t}$ and $\mathbf{y_t}$ denote input and output data at time $t \geq 0$, respectively, while $\mathbf{h_t}$ is hidden state at time t. $\otimes$ is the Hadamard product, $\oplus$ is addition and **1-** in a circle means the subtraction the preceding vector from one, according to Equation 1.12.

1.1.4 Autoencoders

An autoencoder is a neural network which aims at the reconstruction of selected input points [51], first reduced to a hidden representation, called also a latent space or code. The output layer of the autoencoder is the same dimension as the input layer. A model learns a hidden representation of input data and reconstructs the original points based on model output from hidden layers. It is an unsupervised learning algorithm because no labels are needed in the input data [48]. Basically, an autoencoder consists of two main parts: an encoder $\mathbf{h} = f_{enc}(\mathbf{x}) \in \mathbb{R}^h$ which transforms the input vector $\mathbf{x} \in \mathbb{R}^n$ into a hidden representation $\mathbf{h}$ and a decoder $\mathbf{d} = f_{dec}(\mathbf{h}) \in \mathbb{R}^n$ which creates an output based on the transformed data from hidden layers. Finally, the reconstruction vector $\mathbf{y} = f_{dec}(f_{enc}(\mathbf{x})) \in \mathbb{R}^n$ is compared to the original vector $\mathbf{x}$ using a loss function L:

$$L(\mathbf{x}, f_{dec}(f_{enc}(\mathbf{x}))).$$

L checks a similarity between the input vector $\mathbf{x}$ and their reconstruction $\mathbf{y}$. One of the loss functions used for the training of autoencoders is mean squared error (MSE) which is defined as follows:

$$\mathrm{MSE}(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} (x_i - y_i)^2, \tag{1.13}$$

where $\mathbf{x} = [x_1, ..., x_n]$ is an n–dimensional input vector and $\mathbf{y} = [y_1, ..., y_n]$ is an n–dimensional output vector.

We define an autoencoder as undercomplete [51] if dimension of a hidden data representation is smaller than the dimension of the input. Such a construction makes it impossible to use the identity function, which would return the input data without additional constraints. In this case, the autoencoder has to find dependencies between the data and code them in a latent space with fewer dimensions than the input, which will prevent the autoencoder from simply copying the input to the output. It should learn data distribution and leave noise, through learning the structure of a lower-dimensional data manifold, i.e. the internal data representation. As in the case of multilayer perceptrons, autoencoders can be trained using a back-propagation algorithm. The general scheme of the autoencoder architecture with hidden layers is presented in Figure 1.5.

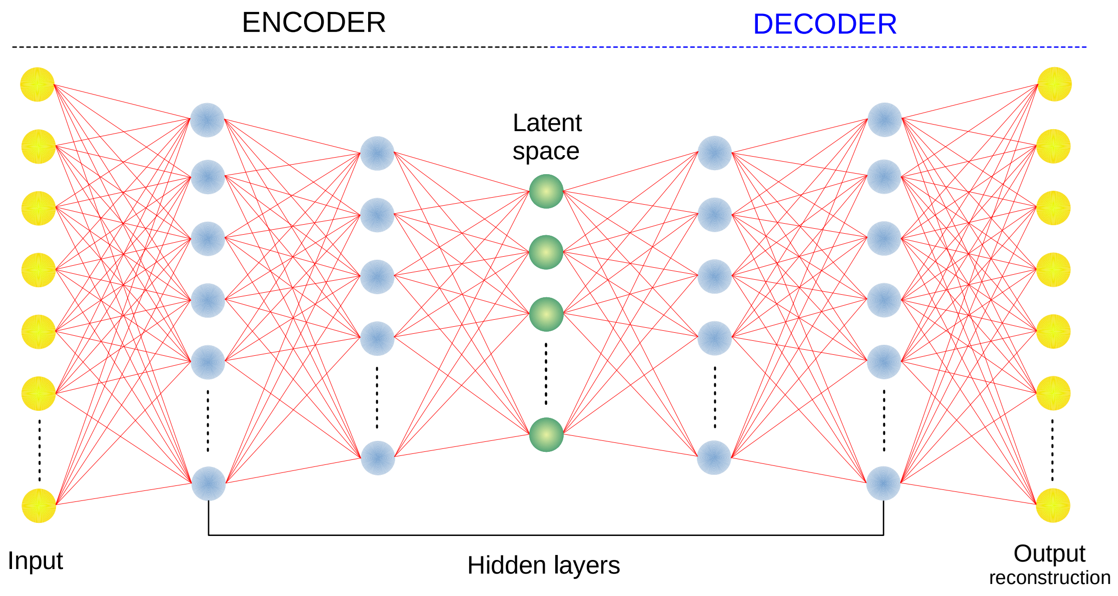


Figure 1.5: The scheme of the autoencoder architecture.

The universal approximation theorem guarantees that a multilayer perceptron with at least one hidden layer with enough neurons can approximate any function arbitrarily well [63]. Thus, the autoencoder with at least one hidden layer and with a sufficiently large number of units can reconstruct original data with any non-zero error value [51].

Autoencoders have various applications, e.g. dimensionality reduction or visualization of complex data in the Cartesian coordinate system [6]. They can reduce the dimension of nonlinear data through the mapping of the data manifold into a simpler representation. If one of the innermost hidden layers has two or three neurons then the autoencoder can be used for visualization of the dataset, often with data separated into clusters according to the different classes. This can be achieved by extracting neuron activation values from this layer during inference. Autoencoders can be used for outlier detection because some important details of outliers are lost during their encoding and reconstruction [6]. Points with the biggest differences between input and reconstruction can be considered outliers. Furthermore, these architectures are used for data denoising. This task is accomplished by adding artificial noise to the training data and calculating the loss L between the reconstruction and the original points, not the distorted ones. Autoencoders are also one of the pretraining methods for initializing the weights of the target neural network [112].

1.2 Hyperspectral images

Hyperspectral cameras take advantage of the fact that different materials emit, absorb and reflect electromagnetic energy, for given wavelengths, according to the chemical composition of a substance [99]. Many hyperspectral sensors like HYDICE, AVIRIS, HYPERION, EnMAP or PRISMA capture spectral information from the visible light and near-infrared (VNIR) range [19], i.e. specified in the wavelength range between 0.4 and 1 μm, as well as in the short-wave infrared (SWIR) region which ranges from 1 to 2.5 μm [19]. Some sensors also make measurements of bands from the long-wave infrared (LWIR), i.e. within the range 2.5 and 7 μm. For example, the AVIRIS sensor, which produced one of the datasets presented in Chapter 3.5.1, has a wavelength range of 0.4 to 2.5 μm. Furthermore, it has a spatial resolution of 20 meters, a spectral resolution of 10 nanometers and creates data cubes of size $512 \times 614 \times 224$.

The sensors measure the radiance intensity that characterizes the photon flux incident at the specified location of a reflecting surface in the direction defined relative to this surface [136]. Reflected sunlight is partially absorbed or scattered by the atmosphere and this process has an important impact on spectral values [19]. When considering the illumination effects, as well as the surface properties and the viewing angle, we can obtain the reflectance values. The spectral reflectance defines the ratio of reflected energy to the incident one and it is an intrinsic property of materials. For many substances, the reflectance values differ at various wavelengths because the energy at different wavelengths can be absorbed or scattered in a different way for specific materials [139]. On the basis of these quantities it is possible to create a reflectance curve as a function of wavelength and use it to identify substances visible in the image. A plot of reflectance curves for several substances from the dataset described in Section 2.5.1 is depicted in Figure 1.6.

The classification of hyperspectral pixels can be burdened by atmospheric ef-

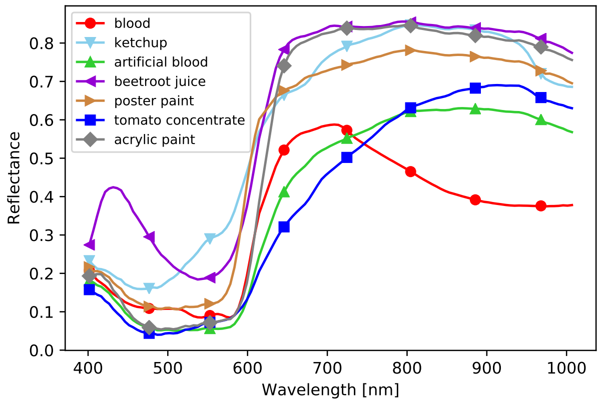


Figure 1.6: The reflectance of several substances based on a sample image from the dataset presented in Section 2.5.1.

fects [139]. The amount of solar energy which goes to the surface is reduced by absorption or scattering the flux by some gas molecules. As an example of the effect of these disturbances, water vapor and carbon dioxide almost completely reduce energy near 1.4 and 1.9 μm. In many cases, adjacent spectral bands are removed. Furthermore, other spectral values should also be corrected to receive proper radiance or reflectance values [19]. We call this process atmospheric calibration. It is also necessary to take into account cloud cover, illumination geometry and shadows. Additional perturbations may be caused by sensor effects related to its detector variations [139, 140]. Most of the work on HSI is focused on the remote sensing approach in which images are typically collected with the use of satellites or aircraft. In the *local* sensing, the impact of a transmission medium, like the atmosphere, can be neglected. An exemplary picture of the stand for *local* sensing is presented in Figure 1.7. It is related to the preparation of the dataset for blood and blood-like substances classification experiments, which will be discussed in Chapter 2.

Another important obstacle is that the spectra of pixels can represent a mixture of different substances. This phenomenon is called spectral mixing [70]. It can occur when the spatial resolution of a sensor is low and different materials are present in a single pixel. Furthermore, substances can create a homogeneous mixture that is independent of the sensor's spatial resolution. In the first case, called the linear one, the reflected energy is combined additively, which means that the contribution of consecutive substances is proportional to the area they occupy in a given pixel [139]. In the latter case, the components are in an intimate mixture, which means that light interacts with more components and is multiply scattered [70]. This mixing systematic is called non-linear. The resulting spectrum will be a composite of *pure* substances called endmembers, whereas the fractions

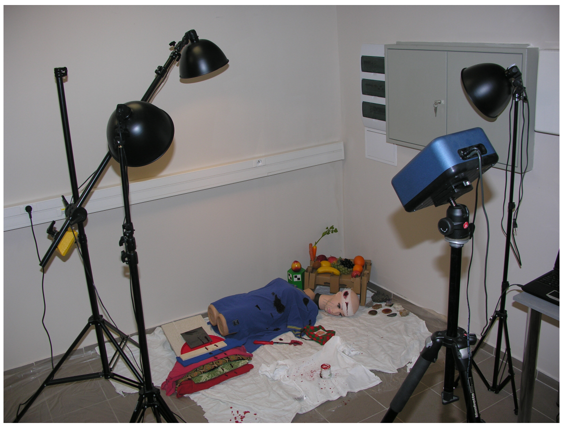


Figure 1.7: The picture of the *local* sensing stand with a hyperspectral camera and several halogen lamps.

which indicate their proportion in a given pixel are called abundances. Exemplary sets of endmembers can be water, trees and soil, vegetables and soil, etc. The inverse problem, which relies on the determination of the original spectra from the mixture pixels (endmembers) and fractional abundances, is called spectral unmixing. It is often solved in an unsupervised way [19]. In Section 3.4.1 we present in detail the Linear Mixing Model which is very common in spectral unmixing [19], despite its simplicity.

In recent years, many deep learning models have achieved state-of-the-art performance in HSI applications. In a hyperspectral classification problem, DL architectures, especially convolutional neural networks, were the most efficient in classification of well-known HSI datasets like Indian Pines, University of Pavia or Salinas [91, 118]. They were also successfully applied in unsupervised problems connected with HSI, e.g. spectral unmixing [113, 114] or anomaly detection [163, 172]. Recent successes justify the selection of deep neural networks as a subject of further research related to hyperspectral imaging.

1.3 Outline of the dissertation

The following four chapters present the results of our research related to neural networks and hyperspectral images. The structure of Chapters 2–4 is as follows: they start with an introduction and the presentation of related works. Then, we describe the necessary definitions and experiment scenarios. Each chapter is summarized by the results and their discussion.

Chapter 2: Blood stain classification with hyperspectral imaging and deep learning architectures
In this chapter, we perform a classification of hyperspectral pixels. As an environment for experiments, we select a dataset designed for the classification of stains of blood and other visually similar substances [133] such as artificial blood, paints, tomato concentrate, etc. We treat this classification scheme as a starting point for the optimization of deep learning architectures in terms of the selection of the neural network type and its individual components. We choose several deep learning architectures, i.e. multilayer perceptron, 1D, 2D and 3D convolutional neural networks as well as a recurrent architecture based on GRU cells. We consider two classification scenarios, i.e. Hyperspectral Transductive Classification (HTC) and Hyperspectral Inductive Classification (HIC). In the first case, the test set is made up of pixels of the same image as the training set, while in the second, a more demanding scenario, the test set come from a different image than the training pixels. We perform experiments using various scenes in both scenarios and we evaluate diverse deep learning models.

Chapter 3: Investigation of weight initialization methods for autoencoders
In the previous chapter, we identified that a part of the problems with HSI classification, and thus architecture optimization, is related to mixed pixels. Therefore, our goal was to prepare spectral unmixing which can improve the results of hyperspectral classification experiments, especially for the HIC scenario. We focused on neural networks designed for hyperspectral unmixing, i.e. specially prepared autoencoders. During the study, we discovered a problem with networks' stability connected with weight initialization. It means that some neural network runs are low-performing and lead to weak reconstruction errors as well as other unmixing scores like abundance or endmember errors. This is consistent with observations from the previous chapter, where we spotted an architecture that in some runs achieved worse results than in the others, even with the same set of hyperparameters. In this chapter, as a result of the above premises, we conduct several unmixing experiments on two different datasets and architectures, four weight initialization methods and various sets of network hyperparameters. We perform statistical tests to thoroughly investigate the impact of weight initialization on the final network result. We conclude that the initialization of network parameters has indeed a significant impact on network performance, i.e. the pixel reconstruction error. Furthermore, we identify a vanishing gradient problem related to the architecture with the ReLU activation function.

Chapter 4: Network reinitialization methods for improving the autoencoders' performance

In the previous chapter, we identified problems connected with the stability of neural networks thus we decided to study some problems in detail. In this chapter, we present an in-depth analysis of selected models trained in experiments from Chapter 3. We identified the dying ReLU neurons phenomenon for a subset of low-performing models. We propose three network reinitialization methods that aim to mitigate the negative effects that occur during the training process. We test their efficiency for selected hyperspectral unmixing experiments and compare the results with the baseline, i.e. scores for models from the previous chapter, without reinitialization techniques. We conclude that in many cases reinitialization methods statistically significantly improve baseline results. Furthermore, we try to generalize the results through the application of proposed methods for larger autoencoder architectures, using the MNIST dataset. We also observe improvement with the use of reinitialization methods, especially for selected network hyperparameters. In many real-world applications, we cannot perform a full hyperparameter optimization due to the high calculation costs, so weight reinitialization methods can minimize the negative impact of dying neurons on network performance and avoid wasting computational time on failed training runs.

Chapter 5: Conclusions
In the last chapter, we briefly summarize the results of the dissertation. We describe the studies presented in consecutive chapters with the corresponding experiments and the conclusions drawn from them. We emphasize which parts of the research confirm the thesis statement. Finally, we present some propositions and ideas for improving the work. They can be used for further research and more detailed investigations in selected areas.

Chapter 2

Blood stain classification with hyperspectral imaging and deep learning architectures

2.1 Introduction

One of the main challenges in applying deep learning is the choice of the appropriate network architecture. We can distinguish several different types of layers which are the main components of deep learning architectures. The linear or fully-connected layer is the most common in neural networks and is the main part of multilayer perceptrons. The convolutional and max pooling layers are used in convolutional neural networks, while LSTM and GRU cells are parts of recurrent networks. Their details were discussed in Section 1.1. Furthermore, not only the architecture selection, but even the choice of the number of neurons in consecutive layers and their order, is a non-trivial task and it has a significant impact on network performance. The problem of hyperparameter optimization or methods like transfer learning, while important for overall network effectiveness, we do not treat it as a part of architecture optimization. We consider architecture optimization as a process of the choice of the neural network type (e.g. linear, convolutional, recurrent ones) and its individual elements, i.e. the number, type and order of layers, etc. Thus, we have to select a reference experiment that allows us to evaluate deep learning architectures, as well as a set of reference architectures that will be reliable representatives of different neural network types. Our considerations are focused on hyperspectral data whose most basic definitions were discussed in Section 1.2.

As an experimental setting for the neural network optimization task, we selected the blood stain classification problem using hyperspectral data. Hyperspectral imaging allows for the non-invasive classification of materials present in a scene. This property is especially important when dealing with a crime scene where interference during evidence collection may have a negative impact on the investigation.

We conducted the study using a dataset from ITAI PAS [133] which was prepared as a part of the project founded by The Polish National Centre for Research and Development[1]. It aimed at the development of methods to support the investigation of the crime scene through the use of laser scanners and satellite measurement techniques. One of the results of the work was a publication dealing with the detection of gunshot residue patterns using machine learning algorithms and hyperspectral analysis [57].

The dataset was created to evaluate blood detection algorithms. It was prepared to emphasize various difficulties to face in real crime scenes, i.e. different sizes of blood stains, diverse composition of backgrounds, the presence of other substances similar to the blood, various acquisition days, etc. The dataset consists of several hyperspectral images of scenes acquisited on different hours and days after substance pouring. The first subgroup with so-called *frame* scenes is composed of large stains of substances spilled on a piece of white fabric while the second one contains more challenging *comparison* images with smaller stains of substances poured on darker materials. In addition, single images of another type can be found, for instance with blood ”splashed” on a fabric. Most of the scenes include stains of blood and other red substances: artificial blood, tomato concentrate, ketchup, acrylic and poster paint, as well as beetroot juice. The reality of the evaluation scenario as well as the diversity of the dataset makes it a good environment for architecture optimization.

We performed experiments according to the two scenarios: *Hyperspectral Transductive Classification* (HTC) and *Hyperspectral Inductive Classification* (HIC). The HTC approach is typical in pixel classification in HSI [118], i.e. the training and the test set come from the same image. It is assumed that the training samples are a good representation of the data distribution. In the field of classification of RGB images, labels are usually assigned to whole images [77] while in the case of hyperspectral images individual pixels are classified. In the HIC scenario, the test set is selected from another image (or images) but contains objects of the same classes. Due to the potential differences in lighting conditions and various chemical and physical properties of background materials, the spectra of

[1]The project no. DOB-BIO6/18/102/2014 entitled *Development of system solutions supporting the analysis of crime and accident scenes on the basis of the data acquired by laser scanning and satellite measurement techniques* was founded by The Polish National Centre for Research and Development. It was realized between 2014 and 2017 in consortium of Silesian University of Technology, Institute of Theoretical and Applied Informatics of Polish Academy of Sciences and others.

pixels containing the corresponding classes in the test set may vary compared to those in the training set. It leads to difficulties in material recognition because machine learning algorithms learn patterns from training samples and new patterns in test samples may lead to misclassification. Furthermore, it violates the i.i.d. assumption [51], which is basic in machine learning. It assumes that the data samples in the training and test set are independent of each other and that they are identically distributed. In the HIC scenario, the second condition may not be fulfilled. This scenario is related to the crime scene investigation where a classifier can be trained in laboratory conditions in a mock-up scene and tested at the crime scene. The HIC scenario is rarely considered by researchers in HSI but has practical applications, e.g. in forensic sciences. These two experimental scenarios are an additional argument for using this schema for architecture optimization because networks will be tested in various cases.

As the network architecture optimization we are faced with an exceedingly large set of possible combinations of candidate architectures, as a first step we select a number of representative architectures from different domains, by which we understand 6 state-of-the-art deep learning methods: a multilayer perceptron, a recurrent neural network consisting of GRU units [109] as well as architectures based on one-[65], two-[83] and three-dimensional [15, 91] convolutions. This choice provides a variety of models used for HSI classification, starting from straightforward architectures like multilayer perceptron up to more complicated 3D convolutional neural networks. Furthermore, some networks take into account only spectral patterns (MLPs, 1D CNN and RNN), while others (2D and 3D CNNs) capture both spectral and spatial dependencies. We additionally selected Support Vector Machines as a baseline algorithm. In our research, we modified and adapted the DeepHyperX library to our requirements. The previous work [13] has tested those architectures in a different hyperspectral setting, so we build on that, as this gives us the additional advantage of comparing the performance with original DeepHyperX experiments.

The foregoing chapter is based on the work *Blood Stain Classification with Hyperspectral Imaging and Deep Neural Networks* [80] published in the Sensors journal (MDPI). According to the Authors' knowledge, it was the first paper that considered the classification of blood stains with HSI and a deep learning approach. In this chapter, the following topics will be discussed:

1. We describe the process of hemoglobin degradation from a chemical point of view which is crucial in hyperspectral imaging. We also present a dataset, including the spectra characteristics, RGB images and ground truth classes. Furthermore, we formally define two scenarios of experiments: the transductive and the inductive one.

2. We introduce a method of the training/test set selection without knowledge

leaks, i.e. unaware use of test samples in the training set as neighbors of center pixels in patches. We also describe all of the network architectures used in the further experiments as well as their evaluation metrics and details connected with the code implementation.

3. We present the results of experiments in two main categories: 9 HTC and 8 HIC setups. We evaluate methods using basic classification metrics (such as overall accuracy) as well as per-class percentage errors. We compare the performance of algorithms in different scenarios. We also depict exemplary classification maps and we analyze calculation times of the considered methods. Finally, we discuss the results obtained, with particular attention to each individual neural network and its advantages and disadvantages. We conclude that the choice of the proper network architecture has a significant impact on the classification results.

As explained in Section 1.2 most of the works on HSI are focused on the remote sensing approach [118] while we concentrate on *local* sensing.

2.2 Related works

The literature review can be divided into several parts. Section 2.2.1 describes machine learning methods applied for hyperspectral data classification, including both the deep learning approach and the classical approach. Section 2.2.2 is focused on previous approaches to the blood stain identification problem, which has become our platform for the optimization of deep learning architectures. Section 2.2.3 describes works related to the HIC scenario in which the distribution of test samples differs from that of the training set. Finally, Section 2.2.4 discusses methods of optimization of neural networks.

2.2.1 Hyperspectral data classification

The key aspects of hyperspectral data classification are widely discussed in [49]. The Authors describe important factors which make difficult a correct classification, like the curse of dimensionality, uncertainties due to e.g. atmospheric conditions; the dependence of spatial resolution on the mixed pixels, etc. They discuss different methods designed for classification: Support Vector Machines, decision trees, ensemble methods and a deep learning approach. A method combining SVMs with a spectral gradient and a spatial random forest is presented in [33]. Another ensemble mechanism, called AdaBoost composite kernel extreme learning machines, is described in [87]. The Authors also examine their algorithm on a very limited number of training samples.

In recent years, due to the rapid development of deep neural networks, this approach in many cases outperformed classic machine learning algorithms. Researchers presented several efficient neural methods designed for hyperspectral data classification, including a dual-channel convolutional neural network [168]. It uses one- and two-dimensional convolutional layers to extract both spectral and spatial information. Another model based on two- and three-dimensional convolutions is presented in [106]. Furthermore, the initial step of the pipeline is dimensionality reduction using a Gaussian random projection and kernel principal component analysis. In [117] the Authors proposed a method called MugNet which comprises of two branches of networks extracting spectral and spatial information from data, respectively. An interesting approach that combines deep learning with SVMs is described in [111]. The deep support vector machine consists of a neural network where weights are dependent on some SVM functions.

Some researchers prepared broad reviews of state-of-the-art classification methods for HSI. In the article [118] some learning strategies and architectures are widely described. The Authors discussed not only individual models but also deep learning limitations, i.e. susceptibility to overfitting, vanishing gradients, etc. Furthermore, a series of experiments using various datasets and complex architectures is performed and then, the results of applied methods are compared. Other overviews of hyperspectral classification approaches are presented in [7, 88] and [13]. We followed the study from [13] and extended the DeepHyperX library that has implemented different deep learning architectures, including multilayer perceptrons, a recurrent and convolutional neural networks. Due to the fact that we would like to optimize architectures, we selected a set of diverse neural networks which were previously studied in remote sensing applications and applied them for *local* sensing in the case of the blood and blood-like substances dataset.

2.2.2 Blood stain identification

Hyperspectral imaging has various applications in medicine [96], including disease detection: segmentation of pathological white blood cells [54] or cancer detection through measurement of blood oxygenation and blood volume of tissues, etc. The high capabilities of HSI open the door to application to the problem of detection and identification of crime evidence by investigators in a non-destructive way. In the review paper [43], the Authors are concerned with using HSI in forensic sciences. They list various subfields of this domain and point to the useful wavelength range. They consider treated and untreated fingermarks, blood stains and other traces. In [85] the Authors present a work connected with the identification of blood stains using HSI. They describe trials with different materials similar in color to blood, such as a red T-shirt, dark blue card, etc. The paper shows the results of experiments discriminating blood from other substances, as well as the detection of latent blood stains, even invisible to the naked eye. Another work [28] is devoted to the identification of blood-stained fingerprints on various porous and non-porous substances as well as on tiles of different colors, both light and dark ones. Furthermore, experiments are performed for dry and wet stains.

Hyperspectral images can also be used for the estimation of blood age. The Authors of [44] estimate the actual age of blood stains from samples 0.1 to 200 days old using the clustering approach. Their method is also based on splitting the pixel spectra into chemical components. In the publication [5] a discussion is made about different methods for estimating blood age; invasive and non-invasive techniques are compared, including Raman spectroscopy [40] and hyperspectral imaging. Authors notice that one of the main HSI limitations, i.e. hardware complexity, was reduced in recent years since the rapid development of devices. They conclude that, together with advanced signal processing methods, HSI is the most efficient technology allowing for non-invasive blood stain age estimation.

2.2.3 Changes in distribution between the training set and the test set

In the presented HIC scenario, test samples come from an image different from the training set. In most cases, this phenomenon is connected with the change in data distribution between the training and test data. Therefore, the works associated with this topic can be found in various terms.

The HIC scenario is related to the covariate shift [143] in which even though the distribution of training points is different from the test samples, the conditional distribution of the output for the given input remains the same. The Authors proposed an improvement of the cross-validation approach in which the validation error of samples is weighted by the ratio of test and training distributions. A modification of this method is proposed in [150] where the aforementioned ratio is estimated through the Kullback-Leibler divergence, while a linear importance model is replaced by the log-linear model. In the work [69] the Authors described methods based on transfer learning in which a target problem is solved through a model trained on a slightly different source problem and its retraining (this process is called fine-tuning). They consider both unsupervised and supervised learning paradigms. In the second scenario, they transfer one or more network layers from the source problem and retrain the model on a target problem. They test the presented idea with different configurations on stacked denoising autoencoders and convolutional neural networks.

2.2.4 Architecture optimization

In this subsection, different approaches for network architecture optimization will be presented, including the choice of network layers, their hyperparameters and more general techniques like Neural Architecture Search (NAS). The Authors in [83] optimized their architecture in such a way that they evaluated different settings of the network, including the number of filters in the multi-scale filter bank, the use of residual learning connections and the determination of the training set size. In the work [15] the Authors considered several convolutional neural networks and they selected one of their propositions making a compromise between high classification accuracy and relatively low calculation time. Another aspect is hyperparameter optimization, which, as we explicitly stated in the introduction to this chapter, is not a part of our approach to network architecture optimization. The choice of hyperparameter values can be performed through libraries such as RayTune [92] or Optuna [8]. This process is related to the adjustment of batch size, learning rate, number of training epochs and other hyperparameters specified for the selected architecture. In the publication [98] Neural Architecture Optimization framework is presented. In this approach, the design of optimal network is determined through the encoder, the performance predictor and the decoder. In the first part, the sequence that describes the architecture is encoded in a continuous representation. The predictor assesses the performance of the encoded architecture while the decoder recovers the input architecture. However, in this work, we consider the architecture optimization similarly to the Authors in [118] and [13] where various neural networks, e.g. multilayer perceptrons or different variants of CNNs, are compared in hyperspectral classification tasks.

Another approach related to architecture optimization is Neural Architecture Search (NAS) which is focused on automatizing the neural network selection process with a limited need for human intervention [130]. There can be distinguished three categories of NAS algorithms [94]: based on reinforcement learning (RL) [177], gradient optimization [135] and evolutionary computation. However, in the case of RL-based algorithms, a huge calculation time is needed, requiring even thousands of GPU cards [94]. In the exemplary gradient-based approach, like DARTS [93], a directed acyclic graph is created. Then, a gradient optimization technique is possible due to the so-called continuous relaxation of the representation of architectures. For evolutionary computation methods, different heuristic algorithms simulating nature behavior are applied, like genetic algorithms [145] or based on swarm intelligence, like particle swarm optimization [144].

2.3 Blood stains

Hemoglobin is a protein that constitutes the vast majority (nearly 90%) of the dry weight of red blood cells [26]. Two species of hemoglobin can be detected in a healthy person's bloodstream: deoxyhemoglobin (deoxyHb) and oxyhemoglobin (oxyHb) [166]. Although oxyhemoglobin is a stable protein, due to auto-oxidation it is transformed into methemoglobin (metHb) and superoxide radical [151]. Special enzymes in erythrocytes, i.e. glutathione peroxidase and methemoglobin reductase, cause a reverse conversion of metHb to oxyHb [14], therefore, the ratio of metHb in blood exiting the human body should not exceed 1% [71]. At this point, various physicochemical processes begin [166]. As a result of saturation, deoxyHb is transformed into oxyHb, and, as the next step, oxyHb degrades to metHb in auto-oxidation. Due to the fact that the antioxidant defense mechanism of red blood cells is no longer available, metHb does not convert back into oxyHb [26, 39]. Then, through the denaturation process, hemi- and hemochromes (HC) are formed.

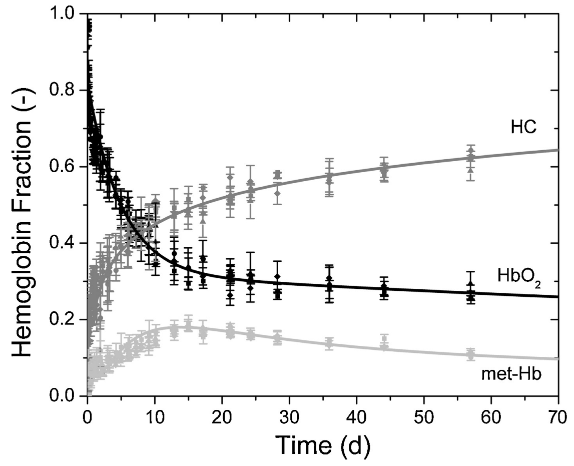


Figure 2.1: The hemoglobin degradation depending on the age of the blood stains (plot from [25] used with permission). In this picture met-Hb means methemoglobin, HbO_2 denotes oxyhemoglobin while HC is hemichrome.

Some factors may lead to hemoglobin distortions and difficulties in blood age estimation. One can mention differences between individuals such as various ages, genders, races, diets, taking medicines (especially aspirin) and the carboxyhemoglobin level, which can be abnormally changed due to smoking or exposure of a given person to fire. Other important factors are related to environmental conditions in which blood was stored, i.e. temperature, humidity or light exposure, both natural and man-made.

The hemoglobin degradation process in an exemplary dataset described in [25] is depicted in Figure 2.1. The presented values are averaged over 20 blood stains. One can notice a rapid decrease in the oxyhemoglobin ratio in the considered blood stains from almost 100% at the beginning to about 40% on the tenth day. Then, the decline slows down, and on the seventieth day, oxyhemoglobin achieves a ratio slightly less than 30%. In the meantime, we observe growth of hemichrome and methemoglobin fractions as a result of auto-oxidation and denaturation processes. In the first case, we notice an increase from 0% for fresh blood to about 45% on the tenth day. Then, the hemichrome fraction still grows but not as rapidly as before and is equal to about 65% on the seventieth day. The methemoglobin ratio in the blood reaches a peak just after the tenth day and is slightly less than 20%. We notice a slight decline later. Finally, the metHb ratio is approximately 10% on the last day of measurements.

The spectral characteristics of blood, especially in the visible light spectrum [166], make hyperspectral imaging a suitable tool for the analysis of blood. This range of the electromagnetic spectrum is predominated by erythrocytes [103]. We can specify several absorption bands of oxyhemoglobin, namely Soret or γ band about 414 nm, α band ~542 nm and β band ~576 nm [176]. An illustrative plot with the blood spectrum and three absorption bands emphasized is presented in Figure 2.2.

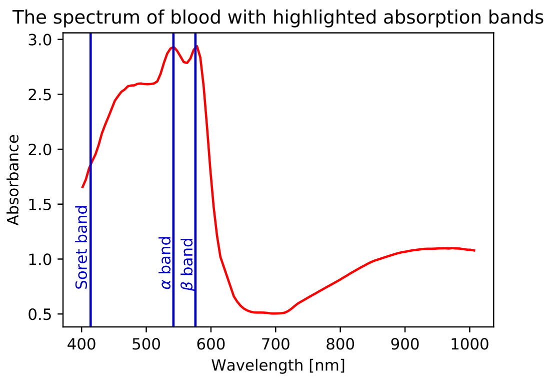


Figure 2.2: The mean spectrum of blood with highlighted three absorption bands: Soret band (414 nm), α band (542 nm) and β band (576 nm). The values of absorbance were calculated as $\log(\frac{1}{R})$, where R is reflectance. The depicted spectrum is based on a subset of blood pixels from the dataset described in [133].

2.4 Candidate architectures description

In this study, we used six deep learning architectures designed for hyperspectral classification: a multilayer perceptron, a recurrent neural network [109], as well as 1D [65], 2D [83] and 3D [15, 91] convolutional neural networks. We followed the implementation from the DeepHyperX library [13] and created its modified version designed for blood stain classification, named DeepHyperBlood [80]. The modifications include but are not limited to implementing a new way of a training and test set selection, different kinds of data preprocessing, modifications of functions calling network training and inference, a different set of functions for SVM classification, plots designed for our datasets and removal of some redundant function from the blood stain classification point of view.

The network hyperparameters, except for a few cases, were taken directly from DeepHyperX. We did not perform additional hyperparameter optimization to ensure the possibility of comparing the results with remote sensing datasets described in [13]. In this way, we also tested the robustness of the default hyperparameters' choice prepared by the authors of architectures and assessed their generalization possibility.

As a reference algorithm, we selected Support Vector Machines (SVMs) [36]. We performed hyperparameter optimization using the grid search technique with the radial basis function kernel. The regularization parameter C was selected from the following set: $\{10^{-3}, 10^{2}, ..., 10^{2}, 10^{3}\}$ while the kernel coefficient γ was chosen among the values: $\{\sigma \cdot 10^{-3}, \sigma \cdot 10^{-2}, ..., \sigma \cdot 10^{2}, \sigma \cdot 10^{3}\}$, where $\sigma = (X_{var} \cdot B)^{-1}$, X_{var} is the variance of the dataset while B is the number of spectral bands.

For all network architectures, we applied a reduction in the learning rate by an order of magnitude when the training process gets stuck. In all considered cases, classification layers are followed by softmax activation and a cross-entropy loss function is used. Moreover, during network learning, 5% of the samples from each class are selected for the validation set.

2.4.1 Multilayer perceptron

We selected multilayer perceptron (MLP), which is the simplest neural network architecture in terms of its construction and it is often used as a reference algorithm for more sophisticated ones [13]. It uses only spectral information of a given pixel and it does not take into account its spatial neighbors. It is composed of three hidden layers with 2048, 4096 and 2048 neurons, respectively. The weights are initialized according to the He approach with normal distribution [60] and all biases are initially set to zero. The activation function is ReLU and the batch size is set to 100. The network is trained through 100 epochs with the Adam optimizer [72] and a learning rate equal to 10^{-4}. The illustration of this architecture is shown in Figure 2.3.

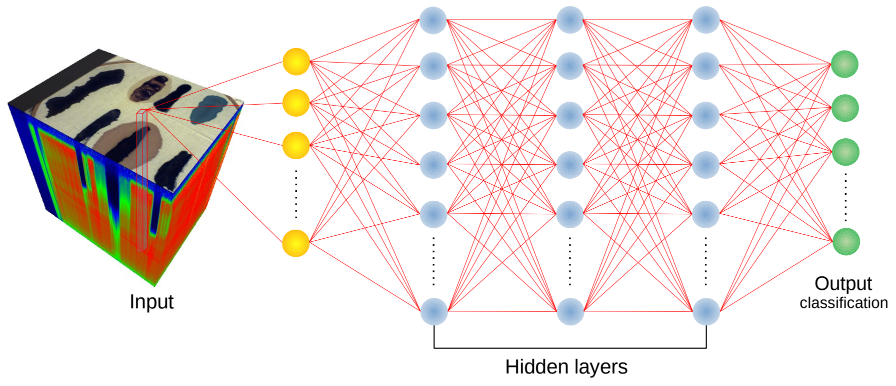


Figure 2.3: The scheme of the MLP architecture for HSI classification.

2.4.2 1D convolutional neural network

One of the convolutional neural networks that we used in our experiments is the architecture of [65] based on one-dimensional layers, presented schematically in Figure 2.4. Similarly, as in the case of MLP, spatial information is not used during training or inference. The first layer after the input prepares one-dimensional convolution and consists of 20 kernels of size $c = \left\lceil \frac{B}{9} \right\rceil$, where B is the number of spectral bands. In this case, stride is set to 1 while padding equals 0. Then, a 1D max pooling layer is applied with the kernel size $p = \left\lceil \frac{c}{5} \right\rceil$ is applied. In this layer, no padding is added but stride is set to p. In the subsequent step, all features are flattened and the pipeline ends in two fully-connected layers. The first has 100 neurons, while the size of the second one corresponds to the number of classes in the dataset. The Authors in [65] suggest that despite the proposed numbers of channels/neurons being effective for different hyperspectral datasets, they can be suboptimal for individual cases.

As an activation function, the hyperbolic tangent was selected. Weights were initialized from the uniform distribution within the range -0.05 and 0.05 while all biases were initialized by zeros. The batch size was set to 50 and the number of learning epochs was 400. Furthermore, the network was trained using SGD optimizer with learning rate 0.01.

2.4.3 2D convolutional neural network

The following architecture is based on the article [83] and takes into account both spectral and spatial relationships in the analyzed data. Its depth is larger than in

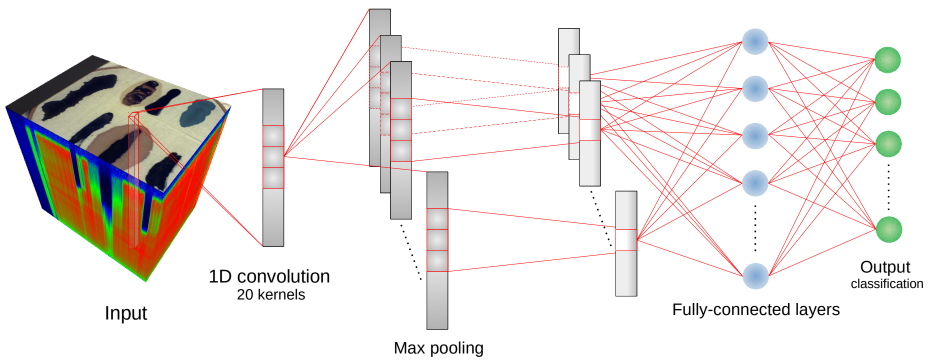


Figure 2.4: The scheme of the 1D CNN architecture [65] for HSI classification.

the case of previously presented networks and it consists of 9 layers.

The network starts with a multi-scale filter bank which consists in simultaneous application of different filters on the input and combining their outputs before passing to the subsequent network layer. In this case, these are $1 \times 1 \times B$ and $3 \times 3 \times B$ convolution filters, where B is the number of spectral bands. The first one is responsible for capturing spectral dependencies, while the second filter exploits primarily spatial relationships. In the original article [83], the authors also included a $5 \times 5 \times B$ filter in the bank, but we followed the implementation from [13] which uses only two filters. The concept of the multi-scale filter bank is similar to the Inception module from [147] which aims to facilitate the extraction of local data structures. The multi-scale filter bank is applied at the first phase of the pipeline to directly analyze the input image.

After processing the input data through the filter bank, local response normalization (LRN) is applied twice and it is intertwined by a convolutional layer. The goal of LRN is to normalize the filter response for selected pixel coordinates (x, y) based on values of m neighboring filters for the same coordinates (x, y) [2, 83], that is:

$$\tilde{f}_j = \frac{f_j}{\left(k + \frac{\alpha}{m} \sum_{i=\max\{1,j-\frac{m}{2}\}}^{\min\{C,j+\frac{m}{2}\}} f_i^2\right)^{\beta}}, \tag{2.1}$$

where f_j is the activation value of the j–th filter for the coordinates (x, y), while k, α, β and m are hyperparameters and C is the total number of channels.

Then, as the next step in forward propagation, the data is passed to two consecutive blocks of layers, each of which consists of two convolutional layers and a residual connection. In the penultimate step, two additional convolutional layers with dropout with a threshold of 0.05 are applied. The pipeline is finished by the eighth convolutional layer without activation function. For each convolutional layer except the last one and at the end of the filter bank, the ReLU activation function is used.

In the code implementation that we followed, the initial learning rate was equal to 0.001 and gradually decreased. The network was trained through 200 epochs, the batch size was set to 100 and the patch size was equal to 5. All weights were initialized according to the He approach with a uniform distribution [60] while biases of the bank layers were initialized to zero.

In this architecture, no dimensionality reduction is performed. Furthermore, for each layer, the number of output channels is equal to 128. We did not use any data augmentation method as the Authors suggest in [83] because we wanted to ensure the same conditions for all algorithms. An overview of the architecture described above is presented in Figure 2.5.

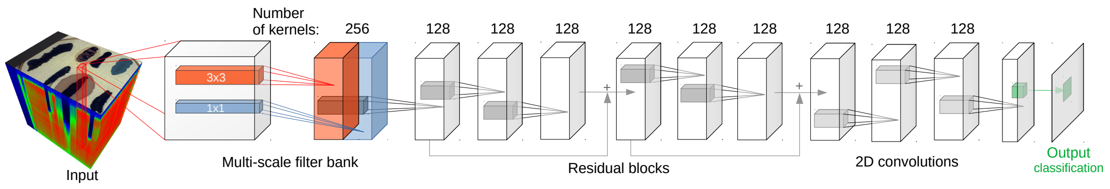


Figure 2.5: The scheme of the 2D CNN architecture [83] for HSI classification.

2.4.4 3D convolutional neural network

We used two 3D convolutional neural networks [15, 91] which can capture both spectral and spatial dependencies between pixels, similarly as architectures based on 2D convolutions.

The first architecture [91], shown illustratively in Figure 2.6, consists of two 3D convolutional layers, having 16 and 32 channels, respectively, and ending with the ReLU activation function. In both layers, small receptive fields are used, i.e. kernel sizes (width $\times$ height $\times$ depth) of $(3\times3\times7)$ and $(3\times3\times3)$. The padding is set to $(0, 0, 1)$ in two cases. In this network, there are no pooling or dropout layers. The pipeline is finished by a single fully-connected layer which is responsible for classification. All weights are initialized according to the Glorot approach [50] with uniform distribution, while biases are set to zero. The network is trained through 200 epochs with batch size 100 and stochastic gradient descent optimizer (SGD) with momentum set to 0.9 while the initial value of learning rate is 0.01. Furthermore, the size of a single patch is equal to 5.

The second architecture [15], depicted in Figure 2.7, is more sophisticated and is composed of two 3D convolutional layers with 20 and 35 feature maps. After each convolutional layer, a pooling layer is applied. The kernel sizes of the convolutional layers are $(3 \times 3 \times 3)$ while the kernel sizes of the pooling layers are $(1 \times 1 \times 3)$.

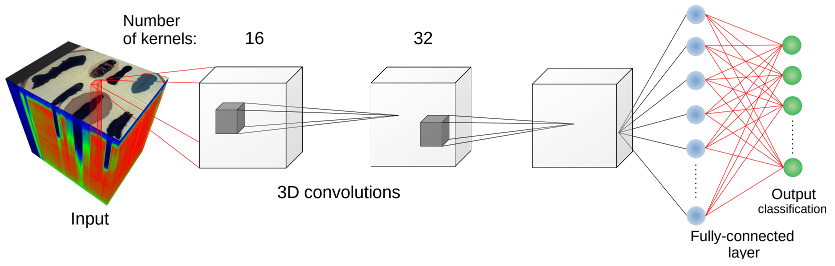


Figure 2.6: The scheme of the 3D CNN architecture [91] for HSI classification.

Furthermore, for convolutions, stride is set to $(1\times1\times1)$ while in the case of pooling layers it is $(1\times1\times2)$ to reduce the spectral dimension of the sample. Then, two 1D convolutions with kernel sizes 3 and 2, as well as with strides 1 and 2, respectively, are applied. Finally, a fully-connected layer performs a classification. Similarly to the previously described networks, the ReLU activation function is applied. Moreover, in our implementation, for each layer except the first one, one-pixel padding is added for the spectral dimension. The number of network training epochs and batch size are set to 100. The remaining hyperparameters are similar to those from the first 3D convolutional architecture, except the momentum in SGD which is set to 0.

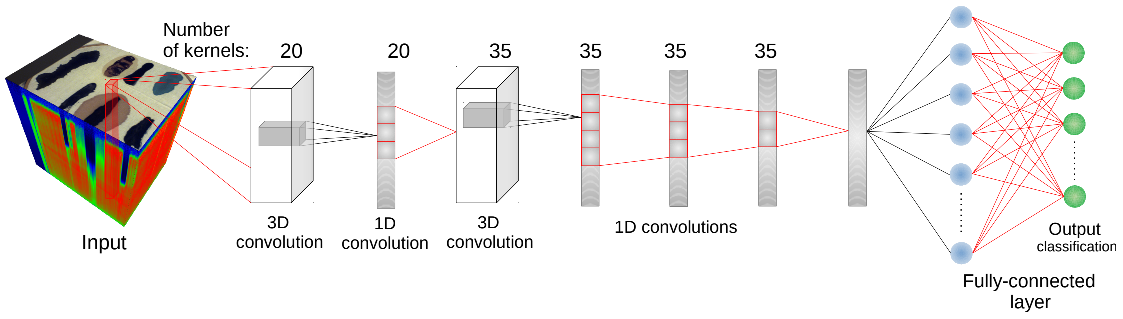


Figure 2.7: The scheme of the 3D CNN architecture [15] for HSI classification.

2.4.5 Recurrent neural network

Except for multilayer perceptron and various convolutional neural networks, we also selected a recurrent network architecture based on gated recurrent units (GRUs). As discussed in Section 1.1.3, GRU cells have fewer parameters than LSTM units and can efficiently extract long-term dependencies from spectral sequences [109]. In this case, a hyperspectral pixel is treated as a sequence and consecutive spectral bands are its successive values. This network, similarly to MLP and 1D CNN, uses only spectral information and does not benefit from the spatial relationships of a selected pixel. The pipeline of this network starts from a single layer of the gated recurrent unit with 64 features in its hidden state. Then, the batch normalization layer and hyperbolic tangent as an activation function are applied. The authors of this architecture designed their own activation function called a parametric rectified hyperbolic tangent (PRetanh) that allows the use of high values of learning rates while minimizing the risk of convergence [109]. However, we followed the implementation of [13] in which PRetanh is replaced by a hyperbolic tangent. In the last step, a linear layer predicts the label for a selected data sample. The overall scheme of this architecture is shown in Figure 2.8. Network training is performed through 100 epochs with batch size 100 and Adadelta optimizer [167] with a learning rate set to 1.0. All network parameters, including biases, are initialized according to the uniform distribution between -0.1 and 0.1.

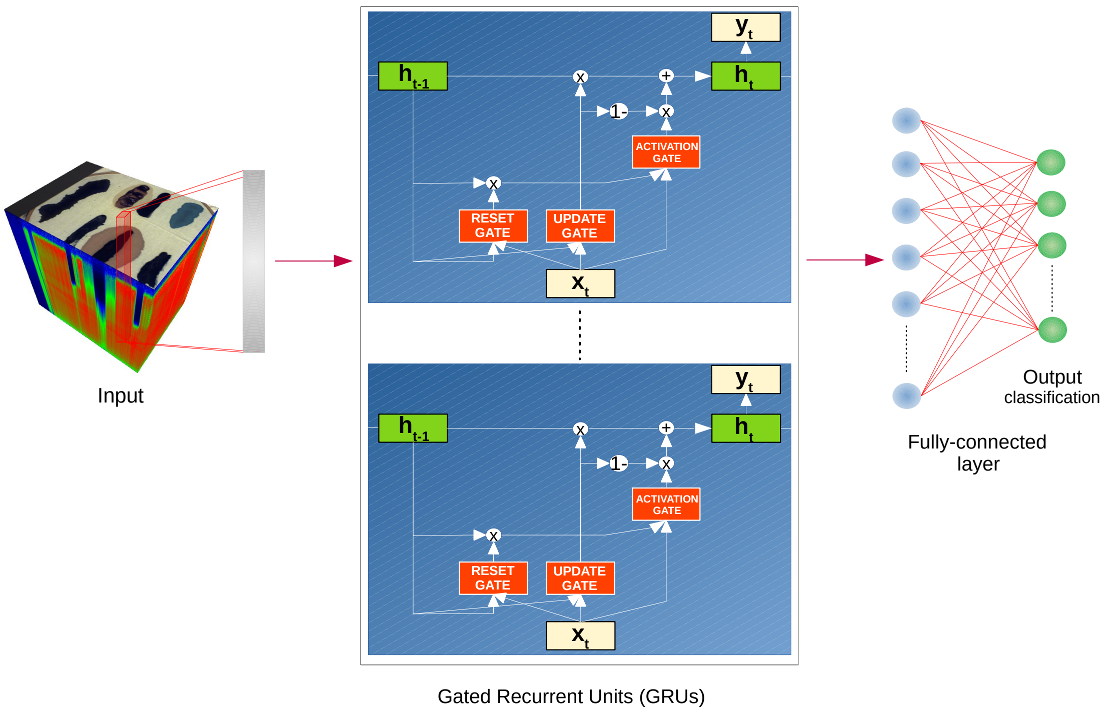


Figure 2.8: The scheme of the RNN architecture [109] for HSI classification.

2.5 Experiments

2.5.1 Dataset

The dataset which we used in the research contains several scenes with *blood* and substances visually similar to *blood*, i.e. *artificial blood*, *ketchup*, *tomato concen-*

trate, *beetroot juice* as well as *poster* and *acrylic paints*. It was prepared in ITAI PAS and was described in detail in the paper [133]. It is also publicly available in the Zenodo repository [132].

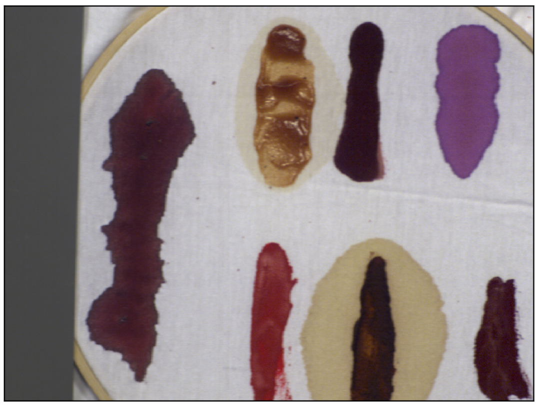
We selected several images to perform classification experiments, i.e. *frame* and *comparison* scenes. *Frame* scenes simulate image acquisition in laboratory conditions. In this case, the substances were spilled onto a white fabric that covered a wooden frame. *Comparison* scenes correspond to real crime scenes because the background is dark and diverse. It is composed of metal, plastic, wood and several fabrics of different dark colors, including red. Furthermore, the substances' stains are smaller than in the case of *frame* images and they are vertically arranged. An additional difficulty can be the spectral mixing of spilled substances with the background materials. Because *beetroot juice* was present only in *frames*, we excluded it from the experiments.

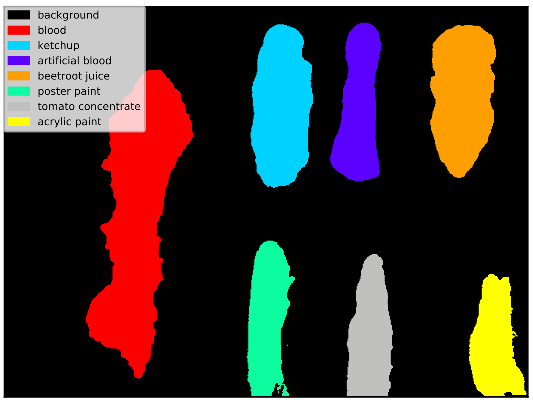
In two presented scenarios, for the purpose of reflectance corrections, the Munsell Color calibration panel was placed on the left side. 8 out of 9 images were performed using the Surface Optics SOC710 hyperspectral camera covering the wavelength range between 377 and 1046 nm. The spatial resolution of the output images is 696×520 while the number of spectral bands is equal to 128. According to suggestions from [133], some spectral bands, i.e. 0-4, 48-50 and 122-128, were removed due to noise, and finally, 113 bands were left. One of the images was made with different equipment, i.e. Specim hyperspectral camera. In this case, the number of spectral bands exceeded 1000, so to ensure the consistency between images, we downsampled it using linear interpolation. We also corrected the reflectance values because we detected artefacts connected with the SOC710 equipment [133].

Both *frame* and *comparison* scenes were captured on different days after the spilling of the substances: from the same day until the twenty-first day after the scene preparation. The detailed description of the images with abbreviations used in further experiments is presented in Table 2.1 while RGB images of considered scenes with corresponding ground truth of class annotations are presented in Figure 2.9. The spectra of substances from six different images are depicted in Figure 2.10. For each substance and image, the spectra were averaged over all pixels in the given class. It can be seen that the drying out of the substances affects the spectra shapes. An example of such a phenomenon is *blood* class in which the reflectance peak at about 700 nm in the *F(1)* image disappears during further days in *F(7)* and *F(21)*. Furthermore, one can observe differences in spectra between *frame* and *comparison* scenes coming from the same day. This observation will be important for the interpretation of one of the experimental scenarios presented here.

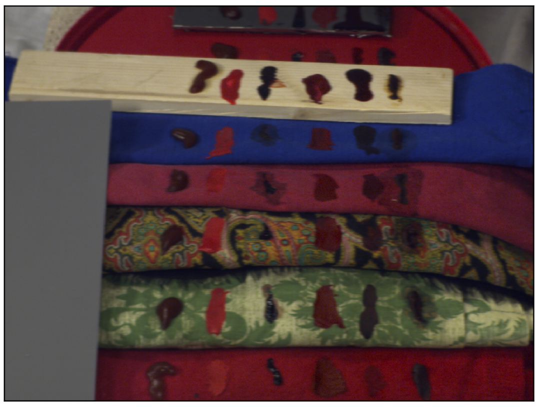
Table 2.1: Summary of images from the dataset [132] used in the study.

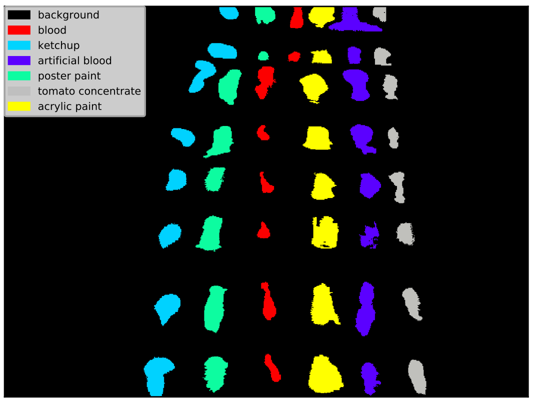
abbreviation	scene type	day	remarks
F(1)	*frame*	1	–
F(1a)	*frame*	1	acquisition about 7 hours after *F(1)*
F(2)	*frame*	2	–
F(2k)	*frame*	2	different hyperspectral cameras
F(7)	*frame*	7	–
F(21)	*frame*	21	–
E(1)	*comparison*	1	–
E(7)	*comparison*	7	–
E(21)	*comparison*	21	–


(a) RGB image of *F(1)* scene.



(b) Ground truth classes of *F(1)* scene.


(c) RGB image of *E(1)* scene.



(d) Ground truth classes of *E(1)* scene.

Figure 2.9: RGB images with corresponding ground truth classes of two selected scenes from the dataset.

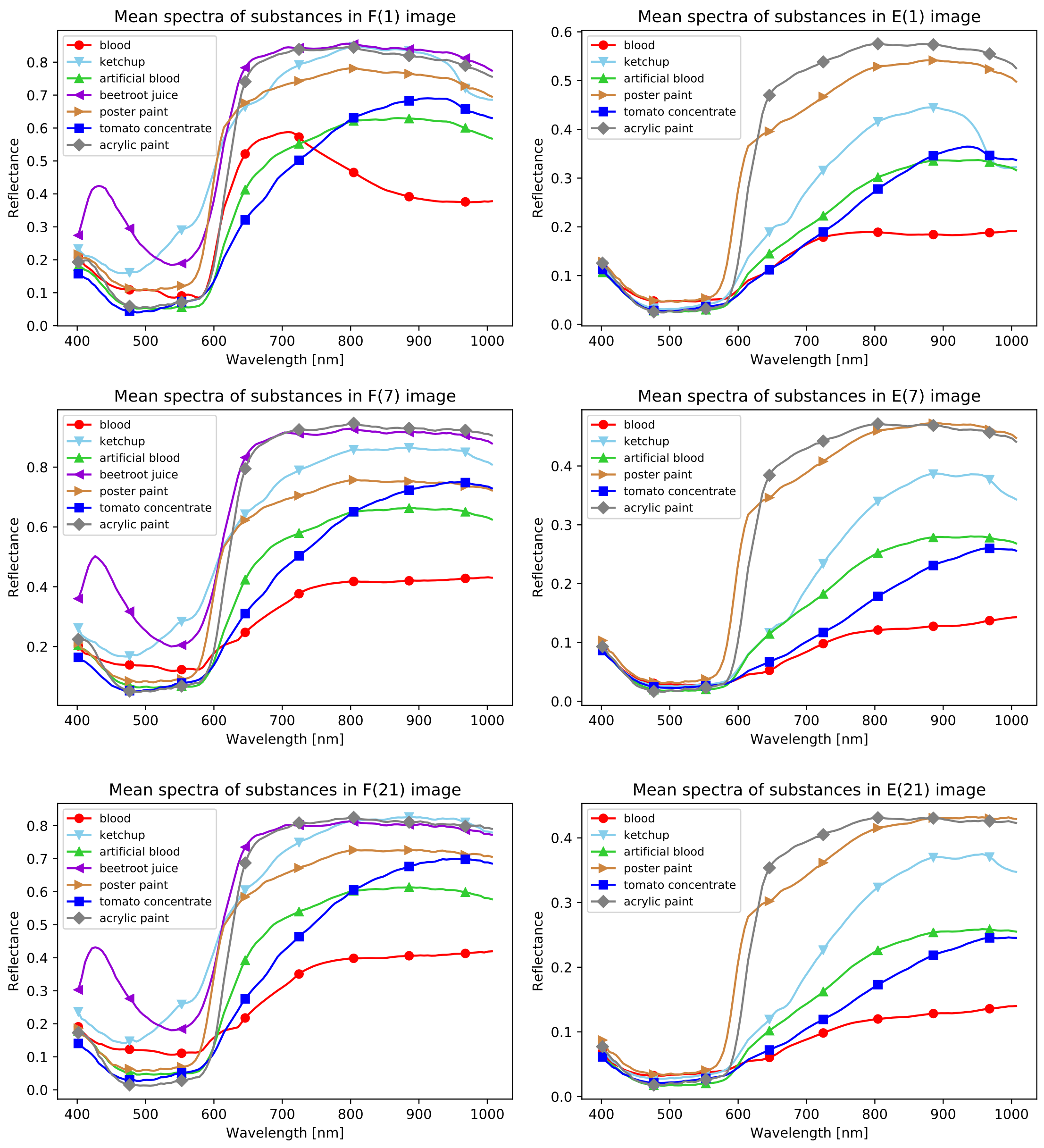


Figure 2.10: Mean spectra of substances in *frame* images and *comparison* scenes from different days after substances' spilling.

2.5.2 Scenarios

We designed two different experimental scenarios:

- The *Hyperspectral Transductive Classification* (HTC) scenario is a typically considered situation for classifiers. HSI pixels from the training set $\mathbf{X_{train}}$ come from the same image $\mathbf{X}$ as the test set $\mathbf{X_{test}}$, i.e. $\mathbf{X_{train}} \subset \mathbf{X}$, $\mathbf{X_{test}} \subset \mathbf{X}$ and $\mathbf{X_{train}} \cap \mathbf{X_{test}} = \emptyset$ [47]. The intersection of the training and the test set is empty, which means that there are no common samples in these sets.

- The *Hyperspectral Inductive Classification* (HIC) scenario is less common but more challenging for classifiers. In this case, two different images, $\mathbf{X_1}, \mathbf{X_2}$, $\mathbf{X_1} \cap \mathbf{X_2} = \emptyset$, are selected. A subset of pixels from one of these images, $\mathbf{X_1}$, is selected for the training set, that is $\mathbf{X_{train}} \subset \mathbf{X_1}$, while the test set $\mathbf{X_{test}}$ is an improper subset of pixels from the second image, $\mathbf{X_2}$, that is $\mathbf{X_{test}} \subseteq \mathbf{X_2}$. In this scenario, the typical machine learning assumption of independent and identically distributed samples in both sets is most likely violated.

The HIC scenario can be especially useful from the point of view of forensic science. Samples of blood or other substances may be prepared under laboratory conditions and can be selected for training a classifier, while samples from the real crime scene may be used as a test set. In the most favorable case, the traces of the crime are judged in place. There is a need for non-invasive methods for substance identification to avoid the violation of samples. Currently, chemical methods are applied, but they result in e.g. substance color changes [24].

We performed experiments according to the HTC scenario for all presented images, that is, *frame* scenes *F(1)*, *F(1a)*, *F(2)*, *F(2k)*, *F(7)*, *F(21)* and *comparison* scenes *E(1)*, *E(7)*, *E(21)*. We also prepared a series of experiments in the HIC scenario, for example *F(1)* $\rightarrow$ *E(1)*, where *F(1)* is a source image, while *E(1)* is a target image. The rest of inductive classification experiments were as follows: *F(1a)* $\rightarrow$ *E(1)*, *F(2)* $\rightarrow$ *E(7)*, *F(2k)* $\rightarrow$ *E(7)*, *F(7)* $\rightarrow$ *E(7)* and *F(21)* $\rightarrow$ *E(21)*. Among these pairs, we have images with the corresponding acquisition times, e.g. *F(1)* $\rightarrow$ *E(1)* as well as with different acquisition hours, like *F(1a)* $\rightarrow$ *E(1)*, or days, as in *F(2)* $\rightarrow$ *E(7)*. Furthermore, we performed two additional HIC experiments between images from two hyperspectral cameras to study the impact of the difference in equipment: *F(2)* $\rightarrow$ *F(2k)* and *F(2k)* $\rightarrow$ *F(2)*.

2.5.3 The selection of training and test set

In the HTC scenario, when we randomly choose indices for training and test sets, there is a risk that patches from the training set will have a non-empty intersection with patches from the test set. All architectures which take into account neighbors of a given pixel are susceptible to information leakage. Two- and three-dimensional convolution networks in which input consists of patches of pixels may covertly use the knowledge from the test set during the training process. It happens when a pixel from the test set is included in the patch of the adjacent pixel that was selected for the training set. To prevent this phenomenon, we designed an algorithm for the division of images into fully separate training and test sets. In the HIC scenario, its application is not necessary because the test set comes from another image, thus there is no threat of information leakage. However, to ensure that the results from HTC and HIC scenarios can be accurately compared, the same training sets were used in both cases for the corresponding images.

We set the number of samples in the training set as $n_{train} = N \cdot 5\% \cdot n^{min}_{samples}$, where $n^{min}_{samples}$ is the number of samples from the least numerous class and N is the total number of classes in the training set. At the beginning, we determine which class is the least frequent in the training set and we set the number of corresponding samples to $n^{min}_{samples}$. Finally, other pixels that are not within k-neighborhood of pixels chosen for the training set or near the image borders, are assigned to the test set. We assumed that $k = 2$ because the maximum patch size for the architectures used is equal to 5×5. In our experiments, all background pixels were ignored. Exemplary applications of this method are depicted in Figure 2.11. There are presented randomly chosen training and test sets for scenes *F(1)* and *E(1)*. In the case of Figures 2.11a-2.11d yellow represents selected pixels, while black corresponds to the background which was excluded from the experiments. The combined training and test set pixels with unused parts of the images are shown in Figures 2.11e-2.11f.

To ensure high reliability of the results, we prepared 10 training sets for each image. The results presented further are averaged over 10 runs per individual scenario.

2.5.4 Evaluation metrics

Overall accuracy

Overall accuracy (OA) is one of the most basic metrics in classification problems. It calculates the proportion of properly classified samples (T) to the total number of data samples (S), which can be delivered in the following way:

$$OA = \frac{T}{S} \cdot 100\%. \tag{2.2}$$

Average accuracy

Average accuracy (AA) indicates the mean classification accuracy over all classes in the dataset. It can be expressed as the following formula:

$$AA = \frac{1}{N} \sum_{i=1}^{N} \frac{T_i}{S_i} \cdot 100\%, \tag{2.3}$$

where T_i is the number of correctly classified samples belonging to the i–th class, S_i is the total number of samples of the i–th class and N is the total number of classes in the considered dataset. Average accuracy can indicate disproportions among ratios of correct predictions for consecutive classes when a significant difference can be observed between its value and overall accuracy.

Cohen's κ coefficient

Cohen's κ coefficient [34] was designed to test the agreement between a pair of variables, most often judges' ratings on a nominal scale. This measure compares the probability of judgments' agreement to the expected one when judges' ratings are independent. Let us assume that p_0 is the ratio of cases in which the judges' ratings are consistent, while p_c is the ratio of cases for which the agreement between judges was only random. κ coefficient can be calculated as follows:

$$\kappa = \frac{p_0 - p_c}{1 - p_c}. \tag{2.4}$$

Therefore, it measures the judges' agreement when the impact of random agreement between judges is removed. The difference $p_0 - p_c$ indicates the scale of the non-random agreement due to the fact that the impact of the random agreement was subtracted from the coefficient of the agreement between judges. The denominator $1 - p_c$ refers to the cases for which the statement about no dependence in data will be indicated differently by the judges. It means that random agreements are subtracted from all possible cases.

In the presented case, κ coefficient was used for the comparison of ground truth classes with label predictions. κ achieves values in the range $[-1, 1]$ while 1 means full agreement between judges and 0 refers to the amount of agreement that can occur randomly [102, 160]. However, some disadvantages of this coefficient for multi-class classification are considered in the work [37] in which the Authors compared Cohen's κ with Matthews Correlation Coefficient. In some cases of unbalanced datasets, higher κ scores are obtained by worse performing models which yields that κ coefficient should be used carefully.

2.5.5 Implementation details

The source code for the described experiments was written in Python 3.7.6 (64-bit) and is publicly available at our GitHub repository: `https://github.com/iitis/DeepHyperBlood`. We modified the DeepHyperX [13] library with several deep learning architectures. The training of neural networks was carried out using PyTorch 1.5.0 [119] and CUDA Toolkit 10.2.89 while Support Vector Machines were learned by using scikit-learn 0.22.1 [120]. Other libraries applied in the study were, e.g.: matplotlib 3.1.2 [29], numpy 1.18.1 [59], scipy 1.4.1 [154], spectral 0.21, torchsummary 1.5.1 and a package management system, conda 4.8.2.

2.6 Results

Upon preliminary analysis of the results, we have discovered individual degenerate cases of 11 training runs. As they represent an utter failure of the learning process, we excluded them from further consideration. These cases are related to the assignment of all (or almost all) image pixels to one class, which significantly disturbs the evaluation metric scores. Such individual failures obscure the real potential of a given network, especially when most of the training session is successful. Therefore, these cases should be identified and excluded from the set of results. Moreover, this phenomenon makes the hyperparameter optimization difficult when sometimes only one network run per each set of hyperparameters is performed. This phenomenon was subsequently investigated in detail, leading to the observation of the degenerate behavior of individual networks. Further analysis leads to conclusions about dead activations and input points and the proposition of reinitialization methods, described in Chapters 3 and 4.

All results that were deemed worthy are presented in Tables 2.2–2.3. They include classification results for all considered architectures and datasets, for two scenarios: HTC and HIC, respectively. The results presented for the transductive classification experiments confirm that *comparison* scenes were more challenging than *frame* images. It is understandable because *frame* images have a white, uniform background while *comparison* scenes are definitely more complex, with different, darker materials than in the aforementioned case and pixel spectra are possibly mixed [70]. For *frame* images, all architectures achieved high efficiency, i.e. between 96.8% and 99.9%. The bigger differences can be noticed in the case of *comparison* scenes. The mean classification accuracy for most effective methods varies between 90.6% and 94.5%, while differences between low- and high-performing algorithms amount to 20%. For three datasets, i.e. *F(2k)*, *E(1)* and *E(7)* scenes, the SVM achieved the highest results. In five other images (*F(1)*, *F(1a)*, *F(7)*, *F(21)* and *E(21)*) 3D convolutional neural networks were the most efficient while in the remaining case, the recurrent neural network obtained slightly better results than the other architectures. We can conclude that all classes were predicted comparably well because overall accuracy scores were similar to the corresponding average accuracy. We also observe that in the case of more challenging *comparison* scenes, the MLP and SVMs were one of the best architectures. This phenomenon occurred despite the fact that they do not take into account the spatial neighborhood of given pixels. It may be related to the complexity of the images and the small size of substance stains. For *comparison* scenes with dried substances, i.e. *E(7)* and

E(21), the 1D CNN achieved the lowest performance when compared to the other architectures. Moreover, it should be mentioned that the HTC scenario is typical in hyperspectral imaging and a general look at the results leads to the conclusion that the task was done correctly.

In the case of the inductive scenario, the classification metrics are significantly lower than those for transductive experiments. The highest performance was achieved for pairs of images in which acquisition was made with different cameras, that is *F(2)* and *F(2k)*. For these datasets, both the overall and average accuracy exceeded 97% for the most efficient architecture. The scenarios, in which the *frames* were in the training set and the *comparison* scenes in the test set, were the most challenging for the classifiers. The overall accuracy ranged between 57% in the case of *F(21)*$\rightarrow$*E(21)* and 72% for the pair *F(1a)*$\rightarrow$*E(1)*. It looks like the change in equipment while preserving other scene conditions, i.e. days of acquisition, background materials, etc., is not as harmful to classifiers as differences in the above-mentioned conditions. It is worth emphasizing that inductive scenarios are not typical from the classifiers' point of view, but they are important for forensic sciences. They correspond to the situation where the real crime scene is investigated, while the classification of substances is performed on the basis of scenes taken under laboratory conditions. In 4 out of 8 cases the recurrent neural network achieved the highest accuracy. The high performance of this method was best demonstrated for the pair *F(1)*$\rightarrow$*E(1)* where the advantage over other algorithms exceeded 8%, in terms of OA. For two other cases, one of the 3D convolutional neural networks was the most efficient while for pairs of images prepared with different equipment, Support Vector Machines achieved the highest scores. The most difficult pair was connected with dried images from the twenty-first day after the substances were spilled, while the least problematic was using pixels from the *F(1a)* as training and pixels from *E(1)* image as a test set. The least efficient architecture for scenarios in which *frames* were training images while *comparison* scenes were test datasets was the 1D convolutional neural network [65] while in the case of *F(2)*$\leftrightarrow$*F(2k)* pairs its two-dimensional counterpart achieved the lowest scores.

Tables 2.4-2.7 indicate the percentages of classification errors for consecutive classes. They are broken down into different scenarios. In the case of HTC experiments on *frame* images, mean errors are low and do not exceed 2%. The lowest error values are obtained by Support Vector Machines while the highest ratio of misclassification can be noticed for the RNN [109]. The toughest class was *tomato concentrate*, but even in this situation, the mean error was only 1.7%. Importantly from the point of view of forensic science, errors for the *blood* were very small and did not exceed 0.5%. For the analogous study with *comparison* scenes, the per-class errors are significantly higher than in the previous case. It is interesting that both *poster* and *acrylic paints* were definitely easier to predict than the remaining classes. It suggests that they differ significantly from the others, which is confirmed by the mean substance spectra depicted in Figure 2.10. The most challenging class was *artificial blood*, especially for the 1D CNN [65]. In the case of inductive scenarios, the scores are slightly different. For results aggregated over *frame* images, *poster paint* and *blood* were classified with the highest efficiency, while *tomato concentrate* caused the most problems. One of the considered architectures, i.e. the 3D CNN [91], obtained definitely worse results than the others and the mean error exceeds 20%. The lowest performance was achieved in the HIC scenario where *comparison* scenes were used as test sets. In such a situation results form two groups: the first of them is created by classes with errors between 18 and 38%, i.e. *tomato concentrate*, *poster* and *acrylic paints* as well as blood. The second, more challenging group, whose errors range from 65 to even 81%, is made up of *ketchup* and *artificial blood.* Similarly, as in the case of full results, the most successful architecture was the RNN [109].

Exemplary classification maps with ground truth and corresponding confusion matrices for selected HTC and HIC scenarios are presented in Figures 2.12 and 2.13, respectively. Figures 2.12a–2.12c present a sample run of the MLP on the *F(1)* image. In this case, classification was performed almost perfectly, although at the bottom edge of the blood stain some pixels were recognized as *artificial blood.* Figures 2.12d–2.12f depict an exemplary transductive training run of the RNN [109] architecture on the *E(1)* image. We can conclude that most misclassified pixels are located near the edges of substances. Possibly in such locations, substance spectra are mixed with background pixels' spectra. About 20% of the *ketchup* and *artificial blood* pixels were labeled as *tomato concentrate.* Furthermore, for almost 10% of cases, *blood* and *poster paint* pixels were assigned to the *artificial blood* class. An interesting phenomenon is presented in Figures 2.12g–2.12i where the 3D CNN [91] network predicted a single class for all pixels of the image. It proves that individual training sessions with this architecture and the selected set of hyperparameters can diverge and lead to degenerate results. Although it was not the only situation in which such a phenomenon occurred, most runs lead to high-performing results. A more detailed analysis of the stability of neural networks with solutions alleviating the described state will be presented in the following chapters. For the HIC scenario, in test sets were all non-background pixels because in training sets, samples were used from other datasets. Figures 2.13a–2.13c compare the prediction of the exemplary RNN [109] run for the pair $F(1a)\rightarrow E(1)$ with its ground truth counterpart. It can be noticed that only 16.1% of the *artificial blood* pixels were correctly classified. Furthermore, the fourth row of substance stains was probably the most challenging for the network. Most of the pixels were assigned as *tomato concentrate.* Problems can be caused by the dark color of the background material which is presented in Figure 2.9c. For an exemplary prediction of the 1D CNN [65] for the pair *F(2k)*$\rightarrow$*F(2)*, the classification metrics were significantly higher than in the previous considered case. The largest level of misclassification was between *ketchup* and *tomato concentrate*. Other classes were labeled with high accuracy.

2.7 Discussion

2.7.1 General remarks

An observation common to all tested methods is that the HTC scenario, which is more often considered in hyperspectral imaging, is definitely easier for classifiers than the HIC scenario. On the basis of the performed inductive experiments, we noticed that different background colors, materials and various acquisition times are more challenging than changes in equipment. Moreover, the substances do not necessarily fully cover the background materials, which can lead to creating a homogeneous mixture of several substances and, consequently, spectral mixing [70]. Thus, the shape of substance spectra will be distorted and the retrieval of original spectra shapes may facilitate the classification process. In the next chapter, we consider spectral unmixing through the application of autoencoders. Furthermore, in the case of changes in acquisition time, the spectra of substances can be reshaped due to their drying, which is harmful to machine learning models trained on samples with slightly different properties. However, we claim that from the point of view of forensic science, the HIC scenario is very important. We would like to prepare models that would be useful for investigators to detect and classify blood stains. In such a case, it is necessary to prepare a picture in laboratory conditions, train the model and use it in a real crime scene. This scenario is definitely more challenging and requires further studies.

We also experienced that neural networks can be unstable and one set of hyperparameters can lead to both high-performing and degenerate results, such as assigning all pixels to one class. While this was not the goal of this study, extensive hyperparameter optimization could improve the results. We will discuss neural network stability and methods of improving wrong training sessions in the next chapters.

2.7.2 Computational times

Table 2.8 describes averaged computational times with standard deviations (given in seconds) for different architectures, datasets and two experimental scenarios if both were used. For all neural networks, experiments were performed with GPU acceleration, which means that SVM is the only architecture trained fully on CPU. The lowest training times were obtained for the MLP and the 3D CNN [15] and they did not exceed half a minute for all cases. Slightly higher values were achieved for the RNN [109]. The training times were by far the longest for the 2D CNN [83] architecture and they were taking more than 4 minutes. It is worth noticing that SVMs need considerably less time for inference than for training. Their calculation time would be reduced if they used the GPU. Another observation is that both training and test times are significantly lower for *comparison* scenes than for *frame* images, because in those cases the number of training pixels is lower due to small stain sizes compared to the background shapes, which is clearly visible in Figure 2.9.

2.7.3 Individual methods evaluation

We also watched all the methods used in the experiments and observed some features related to various training approaches.

- **Support Vector Machines**: This method was used as a baseline for deep learning architectures, but it achieved high classification accuracy for most of the performed experiments. In 3 out of 8 transductive scenarios, including two *comparison* scenes, and in 2 out of 8 inductive scenarios, the Support Vector Machines were the most accurate among the compared algorithms. However, the results obtained for the hardest experiments, i.e. between *frames* and *comparison* scenes, were definitely lower than for more sophisticated neural network architectures. The SVMs also reached the lowest per-class errors for the HTC scenario with *frame* images and for the HIC scenario with datasets made with different equipment.

- **Multilayer perceptron**: Multilayer perceptron, i.e. the simplest feedforward architecture, achieved very competitive results, in some cases better than convolutional neural networks. This corresponds to the conclusions of [13]. In the most demanding transductive experiments on *comparison* scenes, its per-class errors were the lowest. The advantage of MLP is the relatively low calculation time compared to some more sophisticated methods.

- **1D convolutional neural network [65]**: The simplest convolutional neural network among the presented ones was efficient for most transductive experiments, excluding *E(7)* and *E(21)*. However, it achieved high errors per-class in all cases where pixels from *comparison* scenes were in the test set. Although the performance was high for inductive scenarios using *frame* datasets, for other, more demanding HIC scenarios, its classification scores were very low. We think that this architecture may be too simple for more complicated cases. One can notice that it was efficient in experiments carried out by the authors of the DeepHyperX library [13] but they correspond to transductive experiments on other hyperspectral datasets like Indian Pines [100].

- **2D convolutional neural network [83]**: The results of the only two-dimensional convolutional neural network are rather mediocre in both classification scenarios. In none of the cases considered, this method was the most efficient. Furthermore, both the training of the models and the inference lasted the longest among the tested approaches.

- **3D convolutional neural network [91]**: This method obtained the highest accuracy in 3 of 8 transductive classification experiments, including *E(21)*. On the other hand, the scores for *E(1)* and *E(7)*, as well as inductive experiments for datasets from different cameras, are the weakest among other architectures, largely due to the high standard deviation. However, this network has the potential to make high-quality predictions because in [13] for two hyperspectral datasets it was the most efficient. We also observed some under-performing models, which predicted one class for all image's pixels.

- **3D convolutional neural network [15]**: Another architecture that uses three-dimensional layers was very competitive. For this method, we noticed the highest overall accuracy for two transductive and two inductive scenarios, including pairs *F(2)→E(7)* and *F(21)→E(21)*. Its relatively low training costs, the smallest *blood* class error in inductive experiments and quite auspicious results in [13] confirm that it is an efficient architecture. Results for this approach and the previously described 3D CNN [91] indicate that such a network design can efficiently use both spectral and spatial dependencies between HSI pixels.

- **Recurrent neural network [109]**: This method, although it used only spectral information and neglected spatial relationships between pixels, proved to be one of the most efficient in the research. Surprisingly, in 4 out of 8 inductive cases its overall accuracy was the best compared to the other methods. Also, in one transductive experiment, this architecture was top-ranked. Furthermore, the RNN [109] obtained the lowest mean per-class errors in the most demanding inductive scenarios. Results presented in [13] were not as promising as in our case, but this may be due to the fact that they rather correspond to transductive scenarios, while we observed some advantages mainly in inductive experiments.

2.8 Conclusions

We prepared an optimization of deep neural networks for hyperspectral data classification based on a representative experiment with different state-of-the-art architectures and a dataset with blood and blood-like substances. They are visually similar but have different chemical properties and can be distinguished using machine learning algorithms that work on data from HSI cameras. We tested several deep learning methods, including one-, two- and three-dimensional convolutional neural networks, a recurrent neural network and a multilayer perceptron. For comparison purposes, we used Support Vector Machines.

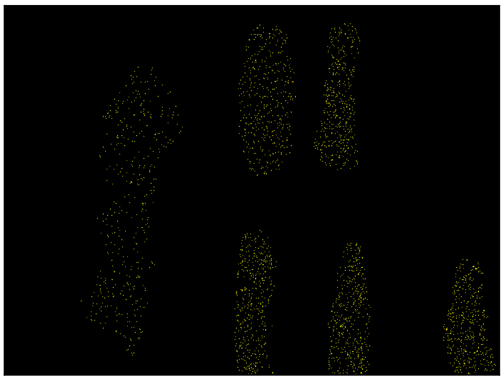
We observed that complex architectures are not necessarily the best solution. For example, a recurrent neural network [109] achieved the best scores in 4 out of 8 HIC experiments, while for *comparison* scenes, in the HTC scenario, the classification accuracy was lower than for some simpler architectures. One of the 3D convolutional neural networks outperformed other architectures for the demanding *E(21)* scene while it was not as effective for other images. Furthermore, one concept of the network may lead to various scores, which can be noticed for two evaluated 3D CNNs ([91] and [15]). The difference was particularly noticeable in the case of inductive scenarios for images acquisited using various equipment. Generally, it is not possible to identify an obvious winner because the overall scores among the networks are diverse. However, multilayer perceptron, i.e. the simplest feedforward architecture, achieved very competitive results.

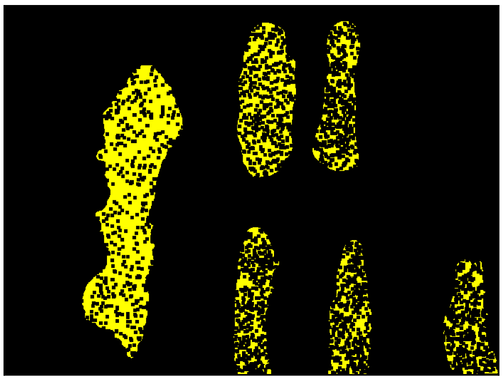
We can also compare our observations with results presented by the Authors of the DeepHyperX library [13]. Similarly, as in our case, the multilayer perceptron was one of the most efficient architectures and achieved the highest performance in one of the presented scenarios. Opposite to our results, 1D CNN [65] was very competitive in the article [13] but in our experiments was one of the weakest architectures. In the blood stain classification RNN [109] distinguished itself from others in the HIC scenarios, while its results were not spectacular in [13]. Two three-dimensional convolutional neural networks were highly effective in [13]. In our experiments, one of these architectures [91] achieved the highest classification accuracy in two HIC and three HTC scenarios. However, this is the network for which we identified a stability problem. The second 3D CNN [15] was comparable to the other evaluated architecture in [13] while in our case the differences between the two 3D CNNs were higher.

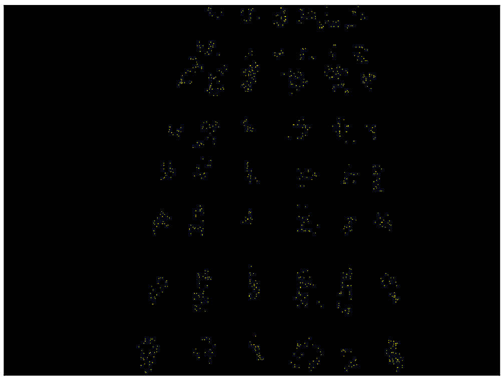
Based on the results of experiments for the HIC scenario, we can conclude that the spectral mixing of pixels affects network performance. Therefore, networks should be able to perform unmixing. An example of such architecture is autoencoders, which can be trained without labels. This is an important advantage because pixel labeling is very time-consuming. Furthermore, the learned weights can be applied in further classification experiments using transfer learning techniques.

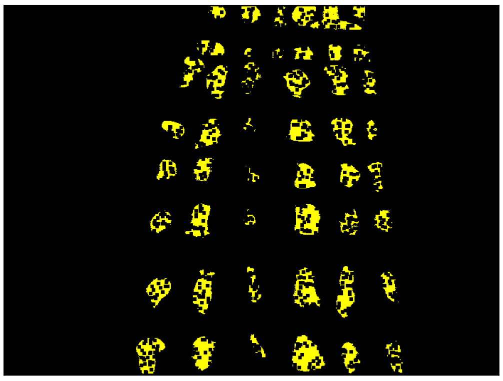
We also identified a network stability problem, i.e. some model training runs with a particular set of hyperparameters led to underperformed models compared to others. Due to the fact that the costs of training complex architectures may be very high, it is very undesirable to waste calculation time on failed training sessions. Thus, network stability analysis is an integrated part of architecture optimization and will be the subject of further consideration.

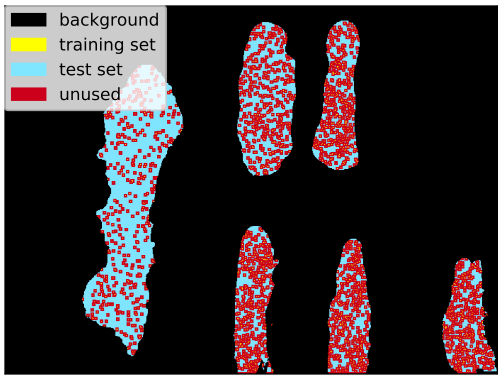
We argue that an initial point of further studies is autoencoders for hyperspectral unmixing. Furthermore, we decided to focus on simple linear architectures because, based on the results from this chapter, we can conclude that they may achieve high performance. Their analysis should be simpler than in the case of architectures with many linear or convolutional layers. We also want to verify the stability of autoencoders to avoid underperformed models. Based on these conclusions, we prepared a series of experiments with linear autoencoders which will be presented in Chapter 3.


(a) An exemplary training set for the *F(1)* image.

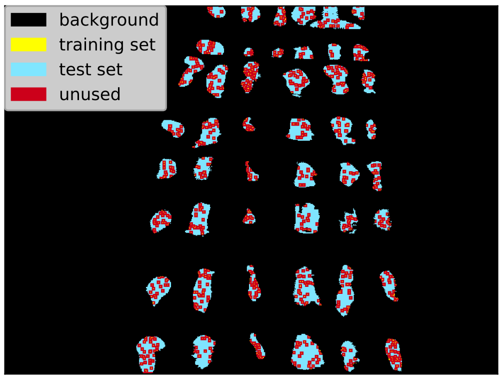

(b) An exemplary test set for the *F(1)* image.


(c) An exemplary training set for the *E(1)* image.


(d) An exemplary test set for the *E(1)* image.

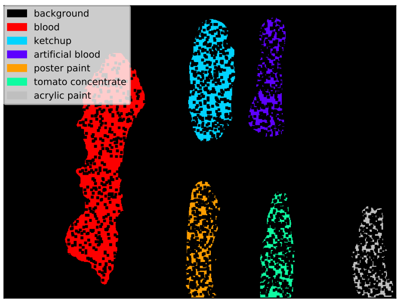


(e) An exemplary division into the training and the test set for the *F(1)* image, presented separately in 2.11a and 2.11b.

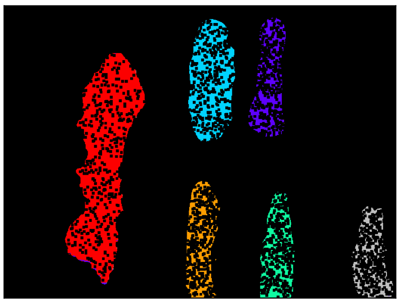


(f) An exemplary division into the training and the test set for the *E(1)* image, presented separately in 2.11c and 2.11d.

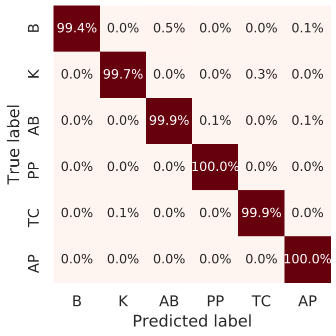
Figure 2.11: A sample of the method used in HTC scenarios that divides images into the training and test set.



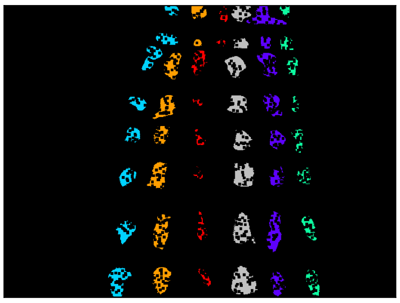
(a) Ground truth of the *F(1)*.



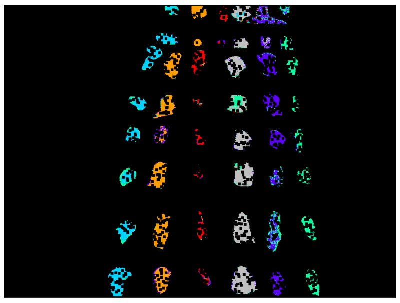
(b) Prediction of the MLP.



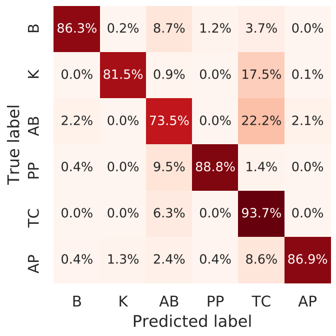
(c) Confusion matrix.



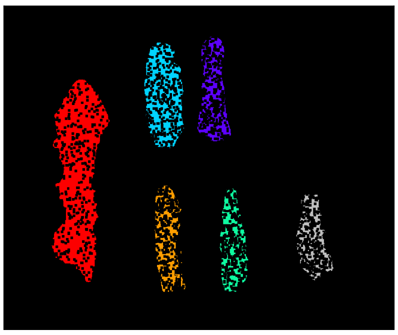
(d) Ground truth of the *E(1)*.



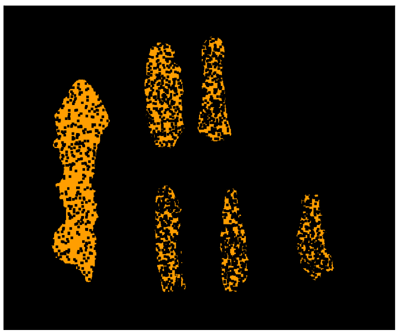
(e) Prediction of the RNN [109].



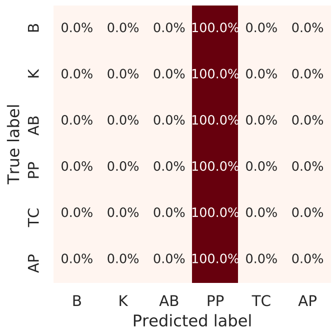
(f) Confusion matrix.



(g) Ground truth of the *F(2k)*.

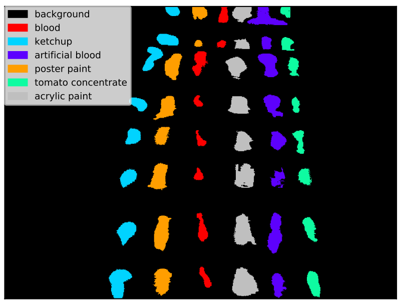


(h) Prediction of the 3D CNN [91].

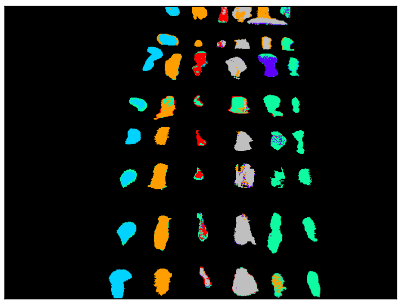


(i) Confusion matrix.

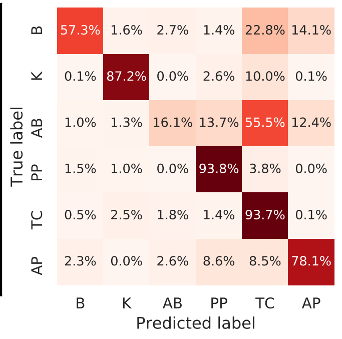
Figure 2.12: Exemplary classification maps of HTC experiments for selected runs of neural networks. The training and test set selection was performed according to the procedure described in Section 2.5.3, which explains the ”holes” in the stains.



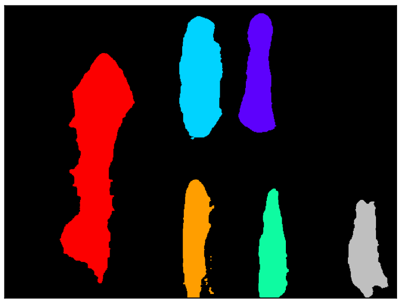
(a) Ground truth of the *E(1)*.



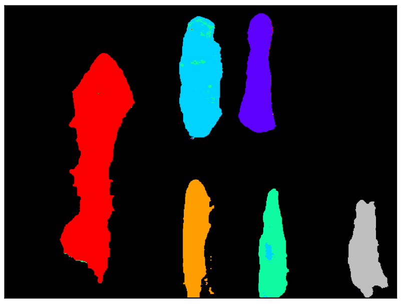
(b) Prediction of the RNN [109].



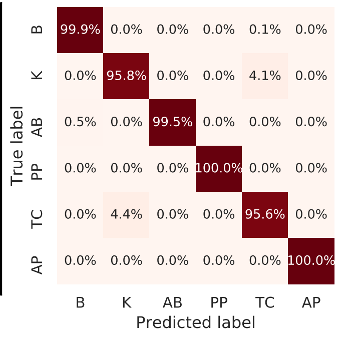
(c) Confusion matrix.



(d) Ground truth of the *F(2)*.



(e) Prediction of the 1D CNN [65].



(f) Confusion matrix.

Figure 2.13: Exemplary classification maps of HIC experiments for selected runs of neural networks.

Table 2.2: The results of the *Hyperspectral Transductive Classification* (HTC) scenario in terms of overall accuracy (OA), average accuracy (AA) and Cohen's κ coefficient. Consecutive rows represent tested images while columns correspond to mean metrics scores (with standard deviations) for different architectures.

				1D CNN	2D CNN	3D CNN	3D CNN	RNN
		SVM	MLP	[65]	[83]	[91]	[15]	[109]
	OA:	99.7 ± 0.1	99.7 ± 0.1	99.2 ± 0.3	98.9 ± 0.8	99.4 ± 0.5	**99.8 ± 0.2**	98.6 ± 3.0
F(1)	AA:	99.8 ± 0.1	99.7 ± 0.2	99.5 ± 0.1	99.2 ± 0.7	99.3 ± 0.8	**99.9 ± 0.1**	97.7 ± 5.5
	κ:	1.00 ± 0.0	1.00 ± 0.0	0.99 ± 0.0	0.98 ± 0.0	0.99 ± 0.0	**1.00 ± 0.0**	0.98 ± 0.0
	OA:	99.7 ± 0.1	99.6 ± 0.3	99.1 ± 0.2	99.3 ± 0.6	99.8 ± 0.2	**99.9 ± 0.1**	99.7 ± 0.1
F(1a)	AA:	99.7 ± 0.1	99.4 ± 0.5	99.0 ± 0.2	99.4 ± 0.4	99.8 ± 0.4	**99.8 ± 0.2**	99.7 ± 0.1
	κ:	1.00 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	1.00 ± 0.0	**1.00 ± 0.0**	1.00 ± 0.0
	OA:	99.8 ± 0.1	99.7 ± 0.2	99.3 ± 0.2	99.7 ± 0.1	99.7 ± 0.3	99.8 ± 0.1	**99.9 ± 0.1**
F(2)	AA:	99.7 ± 0.1	99.4 ± 0.7	98.8 ± 0.3	99.4 ± 0.4	99.4 ± 0.5	99.6 ± 0.2	**99.8 ± 0.1**
	κ:	1.00 ± 0.0	1.00 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	1.00 ± 0.0	1.00 ± 0.0	**1.00 ± 0.0**
	OA:	**99.9 ± 0.1**	99.7 ± 0.1	99.2 ± 0.2	98.9 ± 0.5	99.2 ± 1.7	99.8 ± 0.1	96.8 ± 6.2
F(2k)	AA:	**99.8 ± 0.1**	99.5 ± 0.1	98.8 ± 0.3	98.6 ± 0.8	98.7 ± 2.8	99.7 ± 0.2	95.9 ± 7.5
	κ:	**1.00 ± 0.0**	1.00 ± 0.0	0.99 ± 0.0	0.98 ± 0.0	0.99 ± 0.0	1.00 ± 0.0	0.95 ± 0.1
	OA:	99.8 ± 0.1	99.7 ± 0.1	99.1 ± 0.2	99.4 ± 0.2	**99.9 ± 0.1**	99.4 ± 1.4	98.9 ± 1.0
F(7)	AA:	99.7 ± 0.1	99.5 ± 0.1	98.6 ± 0.2	99.3 ± 0.3	**99.9 ± 0.2**	98.9 ± 2.7	98.1 ± 2.1
	κ:	1.00 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	**1.00 ± 0.0**	0.99 ± 0.0	0.98 ± 0.0
	OA:	99.8 ± 0.0	99.8 ± 0.0	99.4 ± 0.1	99.6 ± 0.3	**99.9 ± 0.1**	99.7 ± 0.2	98.7 ± 2.2
F(21)	AA:	99.7 ± 0.1	99.6 ± 0.2	99.0 ± 0.2	99.4 ± 0.5	**99.8 ± 0.3**	99.5 ± 0.3	96.9 ± 5.8
	κ:	1.00 ± 0.0	1.00 ± 0.0	0.99 ± 0.0	0.99 ± 0.0	**1.00 ± 0.0**	1.00 ± 0.0	0.98 ± 0.0
	OA:	**94.5 ± 0.7**	93.5 ± 1.7	89.8 ± 0.9	93.4 ± 0.8	86.4 ± 8.2	90.8 ± 4.5	85.1 ± 8.3
E(1)	AA:	**93.7 ± 0.8**	93.1 ± 1.3	89.4 ± 0.6	92.7 ± 1.0	84.8 ± 9.3	89.9 ± 3.7	85.8 ± 6.0
	κ:	**0.93 ± 0.0**	0.92 ± 0.0	0.87 ± 0.0	0.92 ± 0.0	0.83 ± 0.1	0.89 ± 0.1	0.82 ± 0.1
	OA:	**90.6 ± 1.7**	90.1 ± 1.9	75.2 ± 1.0	85.5 ± 1.9	80.7 ± 11.3	81.2 ± 9.5	85.3 ± 4.4
E(7)	AA:	**88.9 ± 1.6**	89.1 ± 1.2	74.2 ± 1.0	84.2 ± 2.3	78.2 ± 13.5	79.5 ± 7.7	83.0 ± 6.1
	κ:	**0.88 ± 0.0**	0.87 ± 0.0	0.69 ± 0.0	0.81 ± 0.0	0.76 ± 0.1	0.76 ± 0.1	0.81 ± 0.1
	OA:	87.8 ± 1.7	89.4 ± 1.6	74.0 ± 2.4	83.3 ± 3.5	**94.3 ± 1.8**	87.4 ± 3.5	85.0 ± 4.1
E(21)	AA:	86.1 ± 1.6	88.1 ± 1.5	72.3 ± 2.3	82.9 ± 2.8	**93.3 ± 2.4**	86.9 ± 3.4	83.0 ± 3.7
	κ:	0.85 ± 0.0	0.87 ± 0.0	0.67 ± 0.0	0.79 ± 0.0	**0.93 ± 0.0**	0.84 ± 0.0	0.81 ± 0.1

Table 2.3: The results of the *Hyperspectral Inductive Classification* (HIC) scenario in terms of overall accuracy (OA), average accuracy (AA) and Cohen's κ coefficient. Consecutive rows represent pairs of training / test images (for example *F(1)→E(1)* means that pixels from the *F(1)* scene are in a training set and pixels from the *E(1)* scene form a test set), while columns correspond to mean metrics scores (with standard deviations) for different architectures.

		SVM	MLP	1D CNN [65]	2D CNN [83]	3D CNN [91]	3D CNN [15]	RNN [109]
F(1)→E(1)	OA:	55.9 ± 2.9	57.8 ± 3.6	46.4 ± 2.3	48.5 ± 5.4	58.1 ± 2.8	53.8 ± 3.2	**66.8 ± 3.4**
	AA:	58.6 ± 3.1	60.3 ± 3.3	48.4 ± 2.4	50.6 ± 5.4	55.2 ± 3.6	54.6 ± 2.9	**68.2 ± 3.7**
	κ:	0.48 ± 0.0	0.50 ± 0.0	0.36 ± 0.0	0.39 ± 0.1	0.49 ± 0.0	0.45 ± 0.0	**0.60 ± 0.0**
F(1a)→E(1)	OA:	60.7 ± 2.6	64.8 ± 2.0	52.1 ± 1.5	59.0 ± 4.9	59.3 ± 4.4	66.6 ± 2.1	**71.8 ± 0.9**
	AA:	62.0 ± 3.1	66.0 ± 1.9	53.1 ± 1.5	60.1 ± 4.7	58.2 ± 4.4	67.6 ± 1.7	**71.6 ± 1.2**
	κ:	0.53 ± 0.0	0.58 ± 0.0	0.43 ± 0.0	0.51 ± 0.1	0.51 ± 0.1	0.60 ± 0.0	**0.66 ± 0.0**
F(2)→E(7)	OA:	56.5 ± 5.0	58.7 ± 2.0	49.7 ± 1.0	57.0 ± 2.3	55.3 ± 5.0	**63.0 ± 2.9**	62.2 ± 1.2
	AA:	58.7 ± 6.0	61.3 ± 2.2	52.1 ± 0.9	59.1 ± 1.9	53.6 ± 6.9	**65.1 ± 3.2**	61.7 ± 1.7
	κ:	0.48 ± 0.1	0.50 ± 0.0	0.40 ± 0.0	0.48 ± 0.0	0.45 ± 0.1	**0.55 ± 0.0**	0.54 ± 0.0
F(2k)→E(7)	OA:	52.9 ± 2.6	58.8 ± 1.2	45.7 ± 0.9	51.2 ± 7.1	52.4 ± 7.7	57.3 ± 2.7	**59.6 ± 4.7**
	AA:	54.9 ± 3.1	60.9 ± 1.3	47.1 ± 1.0	52.4 ± 5.8	52.7 ± 6.9	59.0 ± 3.2	**57.4 ± 4.5**
	κ:	0.43 ± 0.0	0.50 ± 0.0	0.35 ± 0.0	0.41 ± 0.1	0.43 ± 0.1	0.48 ± 0.0	**0.51 ± 0.1**
F(7)→E(7)	OA:	54.7 ± 2.1	57.2 ± 2.0	47.2 ± 0.8	54.3 ± 2.0	60.3 ± 3.5	59.8 ± 2.6	**63.6 ± 3.8**
	AA:	59.4 ± 2.0	60.2 ± 2.2	50.8 ± 0.7	56.1 ± 2.3	57.8 ± 5.0	62.9 ± 2.5	**65.3 ± 3.7**
	κ:	0.46 ± 0.0	0.48 ± 0.0	0.37 ± 0.0	0.45 ± 0.0	0.52 ± 0.0	0.52 ± 0.0	**0.56 ± 0.0**
F(21)→E(21)	OA:	45.4 ± 1.7	49.7 ± 2.4	44.0 ± 1.4	49.2 ± 1.6	54.4 ± 3.4	**57.2 ± 1.3**	56.3 ± 3.1
	AA:	48.3 ± 2.4	51.9 ± 1.9	45.7 ± 1.7	51.0 ± 1.3	51.4 ± 3.6	**59.1 ± 1.4**	55.0 ± 2.0
	κ:	0.35 ± 0.0	0.40 ± 0.0	0.33 ± 0.0	0.39 ± 0.0	0.45 ± 0.0	**0.49 ± 0.0**	0.47 ± 0.0
F(2)→F(2k)	OA:	**98.2 ± 0.5**	97.5 ± 0.6	97.0 ± 0.3	96.9 ± 0.5	80.0 ± 12.1	98.1 ± 0.5	90.4 ± 2.0
	AA:	**97.7 ± 0.6**	96.7 ± 0.8	95.8 ± 0.5	95.6 ± 0.8	77.4 ± 12.1	97.2 ± 0.6	87.2 ± 2.5
	κ:	**0.98 ± 0.0**	0.97 ± 0.0	0.96 ± 0.0	0.96 ± 0.0	0.75 ± 0.1	0.97 ± 0.0	0.88 ± 0.0
F(2k)→F(2)	OA:	**99.5 ± 0.2**	99.2 ± 0.3	98.8 ± 0.3	96.5 ± 2.1	85.9 ± 10.6	99.2 ± 1.0	93.0 ± 6.0
	AA:	**99.4 ± 0.3**	98.8 ± 0.5	98.5 ± 0.2	96.6 ± 1.9	82.5 ± 13.9	99.2 ± 0.9	93.2 ± 6.2
	κ:	**0.99 ± 0.0**	0.99 ± 0.0	0.98 ± 0.0	0.96 ± 0.0	0.82 ± 0.1	0.99 ± 0.0	0.91 ± 0.1

Table 2.4: Percentages of errors for different classes and architectures in the HTC scenario aggregated over experiments on *frame* images, i.e. $F(i)$, $i \in \{1, 1a, 2, 2k, 7, 21\}$.

class	SVM	MLP	1D CNN [65]	2D CNN [83]	3D CNN [91]	3D CNN [15]	RNN [109]	mean
blood	0.1	0.1	0.3	0.5	0.1	0.1	0.5	0.3
ketchup	0.5	0.7	1.7	1.1	0.4	0.6	1.2	0.9
artif. blood	0.2	0.3	1.2	0.9	0.5	0.2	3.7	1.0
poster paint	0.1	0.4	0.3	0.4	0.4	0.3	0.4	0.3
tomato conc.	0.5	1.1	2.2	1.4	0.6	0.9	5.4	1.7
acrylic paint	0.1	0.2	0.6	0.5	0.9	0.5	0.2	0.4
mean	0.2	0.5	1.0	0.8	0.5	0.4	1.9	0.8

Table 2.5: Percentages of errors for different classes and architectures in the HTC scenario aggregated over experiments on *comparison* images, i.e. $E(i)$, $i \in \{1, 7, 21\}$.

class	SVM	MLP	1D CNN [65]	2D CNN [83]	3D CNN [91]	3D CNN [15]	RNN [109]	mean
blood	9.7	7.9	12.6	10.5	15.6	9.9	12.0	11.2
ketchup	10.2	8.9	26.0	16.2	17.6	15.8	12.9	15.4
artif. blood	18.3	17.7	43.1	21.4	24.6	22.2	37.5	26.4
poster paint	1.5	1.2	2.3	2.5	1.8	1.9	2.0	1.9
tomato conc.	15.3	15.1	26.4	16.4	18.7	22.0	20.8	19.2
acrylic paint	4.8	6.4	10.9	9.4	7.1	11.1	8.1	8.3
mean	10.0	9.5	20.2	12.7	14.2	13.8	15.6	13.7

Table 2.6: Percentages of errors for different classes and architectures in the HIC scenario aggregated over experiments on *frame* images, i.e. *F(2)↔F(2k).*

class	SVM	MLP	1D CNN [65]	2D CNN [83]	3D CNN [91]	3D CNN [15]	RNN [109]	mean
blood	0.1	0.1	0.1	1.7	10.6	0.0	2.4	2.1
ketchup	1.8	2.4	2.9	4.0	17.0	2.6	8.8	5.6
artif. blood	0.5	0.8	1.7	3.2	16.1	1.3	38.1	8.8
poster paint	0.9	1.0	1.6	1.9	3.9	0.8	0.3	1.5
tomato conc.	4.8	8.1	9.2	11.2	43.4	4.6	8.6	12.8
acrylic paint	0.8	1.1	1.7	1.7	32.4	1.5	0.4	5.6
mean	1.5	2.2	2.9	3.9	20.6	1.8	9.8	6.1

Table 2.7: Percentages of errors for different classes and architectures in the HIC scenario aggregated over experiments on pixels from *frame* images in training sets and pixels from *comparison* scenes in test sets.

class	SVM	MLP	1D CNN [65]	2D CNN [83]	3D CNN [91]	3D CNN [15]	RNN [109]	mean
blood	35.3	30.4	43.8	39.8	44.8	29.4	42.1	38.0
ketchup	48.4	55.1	84.6	73.8	80.8	72.2	42.7	65.4
artif. blood	83.6	82.2	83.4	81.6	77.3	76.7	85.3	81.4
poster paint	32.0	29.6	38.8	34.0	5.7	16.9	9.0	23.7
tomato conc.	20.2	13.9	16.3	13.3	39.3	13.2	14.7	18.7
acrylic paint	36.7	26.0	34.4	27.5	21.1	23.3	20.1	27.0
mean	42.7	39.5	50.2	45.0	44.9	38.6	35.6	42.4

Table 2.8: The mean calculation times with standard deviations (in seconds) of the complete runs for different architectures, datasets and classification scenarios.

		SVM	MLP	1D CNN [65]	2D CNN [83]	3D CNN [91]	3D CNN [15]	RNN [109]
F(1)	training:	72.2 ± 0.8	27.6 ± 0.5	69.4 ± 1.1	251.7 ± 1.9	44.1 ± 1.6	24.4 ± 0.7	32.7 ± 0.9
	test:	4.2 ± 0.8	10.9 ± 0.3	9.3 ± 0.5	76.4 ± 1.6	11.0 ± 0.0	11.7 ± 0.5	14.1 ± 0.3
F(1a)	training:	65.8 ± 0.7	25.3 ± 0.5	65.3 ± 2.7	241.3 ± 0.5	41.0 ± 0.5	24.9 ± 1.1	32.0 ± 1.1
	test:	4.1 ± 0.8	12.2 ± 0.8	10.8 ± 0.4	76.3 ± 2.0	11.0 ± 0.0	11.3 ± 0.5	13.7 ± 0.5
F(2)	training:	60.4 ± 0.8	25.4 ± 0.5	63.0 ± 1.3	234.6 ± 0.5	39.7 ± 0.8	23.4 ± 0.5	29.7 ± 0.5
	test (HTC):	4.1 ± 0.8	11.4 ± 1.0	9.6 ± 0.5	76.1 ± 1.6	10.9 ± 0.3	11.3 ± 0.5	14.2 ± 0.6
	test (HIC):	9.4 ± 1.5	12.0 ± 0.0	10.6 ± 0.5	76.9 ± 0.7	10.9 ± 0.3	11.4 ± 0.5	14.1 ± 0.3
F(2k)	training:	53.1 ± 0.8	26.0 ± 0.5	62.1 ± 1.3	251.0 ± 0.7	44.0 ± 1.8	24.3 ± 0.5	32.1 ± 0.3
	test (HTC):	4.1 ± 0.9	13.1 ± 0.3	12.2 ± 0.4	98.8 ± 0.6	14.2 ± 0.4	13.8 ± 0.4	16.7 ± 0.5
	test (HIC):	9.6 ± 1.9	13.7 ± 0.7	12.7 ± 0.5	98.9 ± 0.7	13.4 ± 0.5	14.2 ± 0.4	17.3 ± 0.5
F(7)	training:	73.5 ± 1.2	27.7 ± 0.5	70.6 ± 1.1	251.3 ± 0.5	43.0 ± 0.5	25.7 ± 0.9	33.1 ± 1.3
	test:	4.6 ± 0.9	11.1 ± 0.6	10.6 ± 1.3	77.3 ± 0.5	11.0 ± 0.5	11.0 ± 0.5	13.7 ± 0.7
F(21)	training:	55.6 ± 0.7	24.4 ± 0.5	60.8 ± 0.8	226.0 ± 0.0	40.0 ± 1.8	24.7 ± 0.5	31.8 ± 1.8
	test:	4.2 ± 0.7	11.3 ± 0.5	9.3 ± 0.5	76.7 ± 0.5	10.9 ± 0.3	11.3 ± 0.5	13.8 ± 0.4
E(1)	training:	15.3 ± 0.7	12.9 ± 0.3	27.3 ± 0.8	132.6 ± 0.5	18.9 ± 0.3	13.1 ± 0.6	18.9 ± 0.7
	test (HTC):	4.1 ± 0.7	12.0 ± 1.2	10.7 ± 0.5	76.8 ± 0.6	10.8 ± 0.4	11.3 ± 0.5	13.9 ± 0.3
	test (HIC):	8.5 ± 1.6	10.3 ± 0.5	8.9 ± 0.3	74.5 ± 2.8	10.8 ± 0.4	11.1 ± 0.3	13.4 ± 0.5
E(7)	training:	7.1 ± 0.6	9.0 ± 0.0	15.6 ± 0.5	101.7 ± 0.5	12.8 ± 0.4	9.0 ± 0.0	14.1 ± 0.9
	test (HTC):	3.9 ± 0.6	11.9 ± 0.6	10.6 ± 0.5	77.0 ± 1.2	10.8 ± 0.4	11.2 ± 0.4	14.1 ± 0.6
	test (HTC):	8.4 ± 1.6	10.3 ± 0.5	9.3 ± 0.7	75.4 ± 1.9	10.8 ± 0.5	11.1 ± 0.6	13.5 ± 0.8
E(21)	training:	9.9 ± 0.7	10.0 ± 0.0	20.0 ± 0.0	114.1 ± 0.7	15.1 ± 0.3	10.1 ± 0.3	13.8 ± 0.4
	test (HTC):	4.0 ± 0.6	11.4 ± 0.7	10.9 ± 0.3	73.9 ± 2.2	10.7 ± 0.5	11.1 ± 0.3	13.4 ± 0.5
	test (HIC):	8.7 ± 1.4	10.2 ± 0.4	8.9 ± 0.3	77.1 ± 1.0	10.6 ± 0.5	11.1 ± 0.3	13.7 ± 0.5

Chapter 3

Investigation of weight initialization methods for autoencoders

3.1 Introduction

In the previous chapter, we have prepared an optimization of deep learning architectures for hyperspectral data classification. Among the two main experiments, in the HIC scenario, the classification accuracy was significantly lower than in the HTC scenario. We concluded that one of the main reasons for such a difference is the spectral mixing of pixels related to e.g. various background materials in the considered images. Therefore, we decided to study the problem of hyperspectral unmixing. Due to the fact that autoencoders are state-of-the-art architectures in the field of unmixing, we focused on this type of neural network in our research. Furthermore, during the experiments with autoencoders, we identified a problem of network instability in terms of high variations of its performance, resulting in different data reconstruction errors. This means that a subset of trained network models has a lower performance than other models, despite using the same set of hyperparameters and data samples. This discovery led us to study the impact of weight initialization on the data reconstruction error. We consider well-known weight initialization methods such as He [60] and Glorot [50] approaches. We conduct an analysis of the network stability for various sets of hyperparameters and randomly generated weights according to given scenarios in a hyperspectral unmixing task. We perform statistical tests to check the impact of initial parameters on the final network reconstruction error.

3.1.1 The impact of mixed pixels

The classification of hyperspectral pixels can be more difficult because of the presence of mixed pixels in an image. The spectrum observed in a given pixel may be a mixture of several substance spectra due, for instance, to the low sensor resolution [70]. This phenomenon was discussed in more detail in Section 1.2 and it can decrease the classification accuracy, especially for the HIC scenario where a training and a test set come from different images [80]. Due to the mixed pixels, we often cannot observe spectra of original substances but only their mixture. In the case of the HIC scenario, the same basic substances can be present in the images, but because of potentially distinct background materials, we observe diverse pixel mixtures. Therefore, the spectra of the foreground substances can be different in the training and the test set. To minimize the negative consequences of spectral mixing, in some publications classification methods are combined with spectral unmixing approaches [55, 153], including autoencoders. Therefore, autoencoders for unmixing have become the main subject of our further research.

3.1.2 Network initialization bias

In Chapter 2 we presented the classification of blood stains using several deep learning architectures, including the convolutional and recurrent ones. We noticed that one of the 3D CNN [91] architectures was unstable, that is, for some runs, the classification accuracy was significantly worse than for others with the same network hyperparameters and dataset. Such a network instability decreased its mean performance and increased the standard deviation. Leaving such results without additional remarks may lead to a distortion of the real capabilities of this architecture and its premature rejection. The detection of this phenomenon and the elimination of underperforming models can improve mean classification accuracy. Moreover, the optimization of deep learning architectures or their hyperparameters assumes that obtained scores are representative, i.e. higher results for a selected network setting (i.e. the selected architecture, batch size, learning rate, etc.) are due to the advantage of this adjustment and not due to the bias of selected training sessions. An example of underperformed models is presented in Figure 3.1 where 2 out of 10 runs achieved significantly lower classification accuracy than the other 8 runs. The lowest score is equal to $\approx 7\%$ while the highest accuracy is almost 100%. The mean classification accuracy with all the presented cases is about 80.85% while after removing the two low-performing training runs it is about 99.19%.

We noticed a similar phenomenon of undertrained models in the case of autoencoders for hyperspectral unmixing. Therefore, in this chapter, we present a more detailed analysis of the network stability on the example of autoencoders.

We consider different weight initialization methods, experiment scenarios, datasets and architectures. We verify whether the initial set of network weights has a statistically significant influence on network performance.

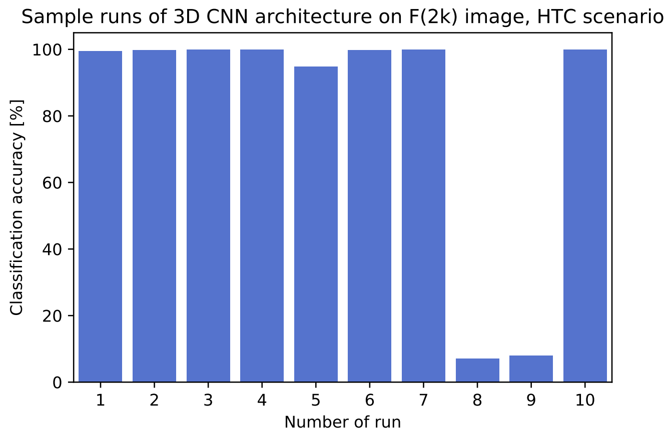


Figure 3.1: Results of sample training runs of the 3D CNN architecture [91] on the *F(2k)* dataset in the HTC scenario.

3.1.3 The further content

This chapter relates to the first part of the paper entitled *Improving Autoencoder Training Performance for Hyperspectral Unmixing with Network Reinitialisation* [79] published in the proceedings of the 21st International Conference on Image Analysis and Processing – ICIAP 2022 in Lecce. In this chapter, the following topics are presented:

1. We discuss the principles behind two widely used weight initialization methods in neural networks: He [60] and Glorot [50] approaches.

2. We describe the Linear Mixing Model (LMM) of the pixel spectra, definitions of endmembers and fractional abundances and necessary conditions which they must fulfill.

3. We present in detail the architectures selected for further experiments, i.e. autoencoder neural networks and the basic definitions related to them.

4. We introduce datasets and error metrics used for the model evaluation during the study. Furthermore, we present 10 experiment scenarios with the description of selected hyperparameters.

5. We perform experiments for 10 settings and 4 weight initialization methods, creating a total of $100,000$ trained model instances. We analyze the stability of the neural networks and unmixing scores, i.e. the reconstruction of fractional abundances and endmembers for consecutive image pixels. We present the summary of results in tables and plots. We depict errors for input reconstruction and endmembers, as well as results of statistical tests. Additionally, abundance and endmember reconstructions for selected models are compared with their ground truth counterparts.

6. Finally, we discuss the results, emphasize the differences between various experiments and present conclusions after research.

3.2 Related works

Hyperspectral unmixing algorithms can be assigned to different categories depending on how they tackle the problem [20]. Among them, methods based on the pure pixel assumption can be mentioned, i.e. the presence of at least one pure pixel per each endmember in the considered dataset. One of these algorithms is the Pixel Purity Index (PPI) [22] which initially transforms the data using the Minimum Noise Fraction (MNF) approach and then tries to find extreme vectors after data projection in different directions. Another algorithm, called N-FINDR [159], is based on the geometric property that the volume of the simplex constructed by the purest pixels is greater than the volumes of simplexes created by any other pixel combination. The Authors in [110] presented the Vertex Component Analysis (VCA) for the linear unmixing problem. The main idea of this approach relies on the iterative projection of data in a direction that is orthogonal to the subspace determined by the former endmembers.

Another category of unmixing algorithms is based on minimization of the volume of simplex containing pixels of the hyperspectral image. Two main approaches which realize this concept relax the abundance nonnegativity constraint, presented in detail in Equation 3.16. This soft constraint improves resistance to noise and outliers, as well as to weak initializations. In the case of SISAL [17], the unmixing task is done by solving a sequence of convex optimization problems via the augmented Lagrangian method. A similar approach is fulfilled by the Minimum Volume Simplex Analysis (MVSA) [86] in which the quadratic optimization problem is solved by the predictor-corrector interior point algorithm. Furthermore, for both SISAL and MVSA, the initialization step is performed by the VCA. In [104] the Authors proposed a method called the minimum transform volume-nonnegative matrix factorization (MVC-NMF), which relies on the two-component optimization problem. The first element of the sum is the square of the Frobenius norm of the difference between the input and the reconstructed pixel. The second component is a penalty function connected with the volume of the simplex determined by reconstructed endmembers. Similarly, in [121] the objective function is composed of two terms. The first of them measures the reconstruction error through the calculation of the Euclidean distance, while the second one, called $L_{1/2}$ regularizer, promotes the sparsity of fractional abundances.

In recent years, researchers proposed some deep learning-based approaches for hyperspectral unmixing. In [114] the Authors designed an autoencoder architecture for blind hyperspectral unmixing, i.e. without using ground truth data. The reconstructed pixels are linear combinations of the encoder output, corresponding to fractional abundances, and the weights of the decoder, which are considered as endmembers. The Authors compared different architectures and loss functions. They also presented a convolutional autoencoder architecture for the unmixing task [116]. Another architecture based on autoencoders, called EndNet, is proposed in [113]. The Authors designed a special objective function that measures not only the Euclidean distance between original pixels and their reconstructions but also, e.g. a Kullback-Leibler divergence in terms of the Spectral Angle Distance score between vectors. Authors in [41] described a dual branch autoencoder architecture for both linear and non-linear mixing models. Their objective function has some regularization components, such as an orthogonal sparse prior term. It is based on the assumption that abundance maps are nearly orthogonal. A similar approach presented by the same Authors in [42] is relied on a single autoencoder and the hyper-Laplacian loss function. Another autoencoder architecture, called DAEN, is presented in [142]. It is composed of several stacked autoencoders and a variational autoencoder which finally prepares the unmixing. Similar to the previous cases, the objective function minimizes not only the reconstruction error (in terms of the Frobenius norm for differences between input and output pixels), but also the volume of the simplex containing vectors of the mixing matrix. An additional regularization term forces abundance constraints. The Authors in [23] prepared a model based on a variational autoencoder. However, in this approach, they need to extract pure pixels from a given hyperspectral image. The deep learning approach was also used, e.g. in the iterative shrinkage-thresholding algorithm (ISTA) [122] with two neural networks and in the untied denoising autoencoder with sparsity (uDAS) [124]. A broad comparison of autoencoders dedicated to unmixing with discussion and results for different datasets is presented in [115]. Another review of deep learning-based methods for the unmixing task was done in [16].

Some researchers combine spectral unmixing with pixel classification methods. In [55] the Authors proposed an algorithm in which first a deep autoencoder for spectral unmixing is trained and then a convolutional neural network performs a classification task. In this approach, many datasets are merged into one bigger dataset. The Authors in [153] present an algorithm that consists of several classic methods like Support Vector Machine for classification or Fully Constrained Least Square for unmixing. The above sample publications show that spectral unmixing can be a useful step before the final pixel classification and is an important research problem in the field of hyperspectral imaging.

3.3 Weight initialization methods

Let us assume that $\mathcal{U}$ is a neural network with n hidden layers. A vector $\mathbf{u} = [u_0, u_1, ..., u_n, u_{n+1}]$ represents the numbers of neurons in consecutive layers, u_0 is the input size while u_{n+1} is the output layer size. Let $\mathbf{y_j}$ be a response of the j–th layer, $j \in \{1, ..., n, n+1\}$:

$$\mathbf{y_j} = \mathbf{x_{j-1}} \cdot \mathbf{P_j} + \mathbf{b_j}, \tag{3.1}$$

where $\mathbf{P_j} \in \mathbb{R}^{u_{j-1} \times u_j}$ is a matrix of the weights of the network, $\mathbf{b_j} \in \mathbb{R}^{u_j}$ is a bias vector and $\mathbf{x_j} \in \mathbb{R}^{u_j}$. For $i \geq 1$, $\mathbf{x_i} = g(\mathbf{y_i})$, where $g(\cdot)$ is an activation function, while for $i = 0$, $\mathbf{x_0} \in \mathbb{R}^{u_0}$ is input. We will present two well-known weight initialization methods: He [60] and Glorot [50] techniques.

He initialization [60]

He approach for weights' initialization is based on the variance control for consecutive network layers [60] and it is an extension of the work [50]. Its main goal is to avert the exponential growth or reduction of the input signals through the forward or backward propagation. Let us suppose that consecutive elements of $\mathbf{P_j}$ and $\mathbf{x_j}$ are independent and identically distributed. Furthermore, they have to be independent of each other. In such a situation, y_j, p_j and x_j are random variables corresponding to elements of $\mathbf{y_j}$, $\mathbf{P_j}$ and $\mathbf{x_j}$, respectively, and the mean of p_j is equal to 0. Thus, we can state that:

$$Var(y_j) = u_{j-1} \cdot Var(p_j \cdot x_{j-1}). \tag{3.2}$$

If X_1 and X_2 are independent random variables, then:

$$\begin{aligned} Var(X_1 \cdot X_2) = &\left(Var(X_1) + (\mathbf{E}(X_1))^2\right)\left(Var(X_2) + (\mathbf{E}(X_2))^2\right) \\ &- (\mathbf{E}(X_1))^2(\mathbf{E}(X_2))^2. \end{aligned}$$

Based on the above property, we can write that:

$$\begin{aligned} Var(p_j \cdot x_{j-1}) &= \Big(Var(p_j) + (\overbrace{\mathbf{E}(p_j)}^{=0})^2\Big)\Big(Var(x_{j-1}) + (\mathbf{E}(x_{j-1}))^2\Big) \\ &- \overbrace{(\mathbf{E}(p_j))^2(\mathbf{E}(x_{j-1}))^2}^{=0} = Var(p_j)\Big(Var(x_{j-1}) + \mathbf{E}(x_{j-1})^2\Big) \\ &= Var(p_j)\Big(\mathbf{E}(x_{j-1}^2) - (\mathbf{E}(x_{j-1}))^2 + (\mathbf{E}(x_{j-1}))^2\Big) \\ &= Var(p_j) \cdot \mathbf{E}(x_{j-1}^2). \end{aligned} \tag{3.3}$$

Taking into account Equation 3.3 it is possible to rewrite Equation 3.2 in the following way:

$$Var(y_j) = u_{j-1} \cdot Var(p_j) \cdot \mathbf{E}(x_{j-1}^2). \tag{3.4}$$

If we assume that the distribution of p_{j-1} is symmetric around zero and all biases are zero, then also the distribution of y_{j-1} is symmetric around zero and its mean is equal to zero. Based on these conditions, we can conclude that if only $g(\cdot)$ is ReLU, i.e.: $x_j = \max(0, y_j)$, then $\mathbf{E}(x_{j-1}^2) = \frac{1}{2} \cdot Var(y_{j-1})$.
By definition of the expected value of a random variable [62]:

$$\mathbf{E}(x_{j-1}^2) = \int_{-\infty}^{+\infty} \max(0, y_{j-1})^2 \cdot f(y_{j-1})dy,$$

where $f(y_{j-1})$ is the probability density function of the random variable y_{j-1}. Since the negative values of y_{j-1} do not change the result of the above integral, it is possible to state that:

$$\mathbf{E}(x_{j-1}^2) = \int_{0}^{+\infty} y_{j-1}^2 \cdot f(y_{j-1})dy.$$

Based on the fact that y_{j-1}^2 and $f(y_{j-1})$ are symmetric around 0, we can write:

$$\mathbf{E}(x_{j-1}^2) = \frac{1}{2}\int_{-\infty}^{+\infty} y_{j-1}^2 \cdot f(y_{j-1})dy.$$

We also know that $\mathbf{E}(y_{j-1}) = 0$, therefore:

$$\mathbf{E}(x_{j-1}^2) = \frac{1}{2}\int_{-\infty}^{+\infty} \Big(y_{j-1} - \mathbf{E}(y_{j-1})\Big)^2 \cdot f(y_{j-1})dy.$$

By definition, the above equation is half the expected value of $\Big(y_{j-1} - \mathbf{E}(y_{j-1})\Big)^2$:

$$\mathbf{E}(x_{j-1}^2) = \frac{1}{2} \cdot \mathbf{E}\bigg(\Big(y_{j-1} - \mathbf{E}(y_{j-1})\Big)^2\bigg).$$

Finally, using the definition of a variance:

$$\mathbf{E}(x_{j-1}^2) = \frac{1}{2} \cdot Var(y_{j-1}). \tag{3.5}$$

By applying Equation 3.5 in Equation 3.4, we get the following expression:

$$Var(y_j) = \frac{1}{2} \cdot u_{j-1} \cdot Var(p_j) \cdot Var(y_{j-1}). \tag{3.6}$$

We can calculate the variance of the $(n+1)$–th layer:

$$Var(y_{n+1}) = Var(y_1)\left(\prod_{j=2}^{n+1} \frac{1}{2} \cdot u_{j-1} \cdot Var(p_j)\right). \tag{3.7}$$

Equation 3.7 is crucial to ensure that the input signals would not increase or decrease exponentially. We demand that

$$\forall j \in \{2, ..., n+1\} \;\; \frac{1}{2} \cdot u_{j-1} \cdot Var(p_j) = 1. \tag{3.8}$$

For simplicity we neglect the fact that for $j = 1$ we have $u_0 \cdot Var(p_1) = 1$ because the activation function is not used for the input layer [60]. Thus, we also apply Equation 3.8 for the first layer. To fulfill the condition of Equation 3.8 we can initialize weights of the j–th layer, $j \in \{1, ..., n+1\}$, from a Gaussian distribution with zero mean and standard deviation equal to $\sqrt{\frac{2}{u_j}}$. Furthermore, all biases are initialized with zeros.

We can perform similar considerations for the backward propagation case. Let $\tilde{\mathbf{P}}_{\mathbf{j}} \in \mathbb{R}^{u_j \times u_{j-1}}$ be the rearranged matrix of the network weights. $\Delta x_j \in \mathbb{R}^{u_j}$ represents the values of the gradients in neurons of the j–th layer while $\Delta y_j \in \mathbb{R}^{u_j}$ denotes the gradients in the outputs of neurons of the j–th layer. Similarly to previously, $\tilde{p_j}$ is a random variable of an element of the matrix $\tilde{\mathbf{P}}_{\mathbf{j}}$. Analogical inference as in the case of forward propagation leads to the following equation:

$$Var(\Delta x_1) = Var(\Delta x_{n+1})\left(\prod_{j=1}^{n} \frac{1}{2} \cdot u_j \cdot Var(\tilde{p_j})\right). \tag{3.9}$$

A requirement to keep the gradient stable is that the above product is equal to 1:

$$\forall j \in \{1, ..., n\} \quad \frac{1}{2} \cdot u_j \cdot Var(\tilde{p_j}) = 1. \tag{3.10}$$

It is not necessary to calculate the gradient for the input signal (Δx_1) but for simplicity we can also apply Equation 3.10 for the first layer. During the network initialization, it is enough to apply only one of the conditions from Equations 3.8 and 3.10 which was confirmed by the Authors in [60].

Glorot initialization [50]

Similar derivations were performed by the authors in [50] but they considered only the linear case, while ReLU is a non-linear function. After some derivations, they concluded that it is necessary to meet the following conditions:

$$\forall j \in \{1, ..., n+1\} \quad u_{j-1} \cdot Var(p_j) = 1, \tag{3.11}$$

$$\forall j \in \{1, ..., n+1\} \quad u_j \cdot Var(p_j) = 1. \tag{3.12}$$

To take into account the above conditions, for networks with all layers of the same width, we may require the following:

$$\forall j \in \{1, ..., n+1\} \quad Var(p_j) = \frac{2}{u_{j-1} + u_j}. \tag{3.13}$$

Including an additional property from Equation 15 in [50], we can finally present the Glorot weight initialization method:

$$\forall j \in \{1, ..., n+1\} \quad \mathbf{p_j} \sim U\left[-\sqrt{\frac{6}{u_{j-1} + u_j}}, \sqrt{\frac{6}{u_{j-1} + u_j}}\right]. \tag{3.14}$$

This approach can also be adopted for the Gaussian distribution with zero mean.

3.4 Methods for spectral unmixing

3.4.1 Linear Mixing Model

Spectral unmixing is a procedure of decomposition of a mixed pixel into a set of distinct pure spectra called endmembers and a set of fractional abundances indicating the proportion of consecutive endmembers in a given pixel [70, 139]. In the Linear Mixing Model (LMM) we assume that the surface area is divided proportionally between multiple components having consistent spectral properties [20, 70]. In this case, the spectra of pixels are a linear combination of such endmembers, while the coefficients are fractional abundances.

Let us suppose that $\mathbf{x} = [x_1, x_2, ..., x_B]^\top$ is a hyperspectral pixel of B spectral bands having S endmembers which can be expressed as follows:

$$\mathbf{x} = \sum_{i=1}^{S} a_i \cdot \mathbf{e_i} + \mathbf{w}. \tag{3.15}$$

The vector $\mathbf{e_i} = [e_1, e_2, ..., e_B]^\top$ represents the i–th endmember, $i \in \{1, 2, ..., S\}$, the value a_i is the i–th fractional abundance, $\mathbf{a} = [a_1, a_2, ..., a_S]^\top$, while

$\mathbf{w} = [w_1, w_2, ..., w_B]^\top$ is the noise vector. We require the physical properties of the abundances vector to be met:

$$\forall i \in \{1, 2, ..., S\} \quad a_i \geq 0 \quad \text{(nonnegativity constraint)}, \tag{3.16}$$

$$\sum\nolimits_{i=1}^{S} \quad a_i = 1 \quad \text{(sum to one constraint)}. \tag{3.17}$$

It is possible to rewrite Equation 3.15 for P pixels. Let $\mathbf{X} \in \mathbb{R}^{B \times P}$ be the matrix where columns contain the spectra of P pixels in the image, $\mathbf{E} \in \mathbb{R}^{B \times S}$ denotes the matrix of S endmembers, $\mathbf{A} \in \mathbb{R}^{S \times P}$ represents the matrix of fractional abundances corresponding to proportions of endmembers in P pixels and $\mathbf{W} \in \mathbb{R}^{B \times P}$ means the noise matrix for the considered image. Then, LMM can be expressed as follows:

$$\mathbf{X} = \mathbf{E}\mathbf{A} + \mathbf{W}. \tag{3.18}$$

Our goal is to reconstruct the endmember matrix $\mathbf{E}$ and the matrix of fractional abundances $\mathbf{A}$ based on the spectra of mixed pixels of the image.

3.4.2 Autoencoder architectures

For the hyperspectral unmixing task we selected autoencoders, i.e. neural network architectures described in Section 1.1.4. Autoencoders consist of an encoder and a decoder part. Their construction favors expressing consecutive pixels of the image as a product of fractional abundances and endmembers, according to Equation 3.18. Such networks learn data representation and create a latent space through data downscaling. The last encoder layer has the same size as the number of endmembers in a given dataset. Thus, the values of neurons in this layer represent fractional abundances for the input pixels. Finally, the decoder layer consists of one layer that creates a reconstruction. The weights of this layer correspond to endmembers. Then, the output of the autoencoder is a product of neurons in the last encoder layer and the weights of the decoder part. Such a reconstruction should be similar to input data and the loss function L assesses the distance between input and its reconstruction. It is a fully unsupervised approach because we need neither the ground truth of endmembers nor abundances during the learning process. The minor disadvantage is that the number of endmembers has to be known a priori, but it can be determined using, for instance, HySime algorithm [18].

We selected two linear autoencoder architectures for hyperspectral unmixing experiments:

1. *original*: the most efficient architecture from [114], depicted in Figure 3.2,
2. *basic*: the proposed simplified version of the *original*, depicted in Figure 3.3.

The encoder part of *original* consists of four hidden layers which have $9S$, $6S$, $3S$ and S neurons, respectively. For all encoder layers, the sigmoid activation function is used. During the next steps, four additional layers are applied: batch normalization, dynamical soft thresholding, sum-to-one constraint layer and Gaussian Dropout. Below, they will be described in detail.

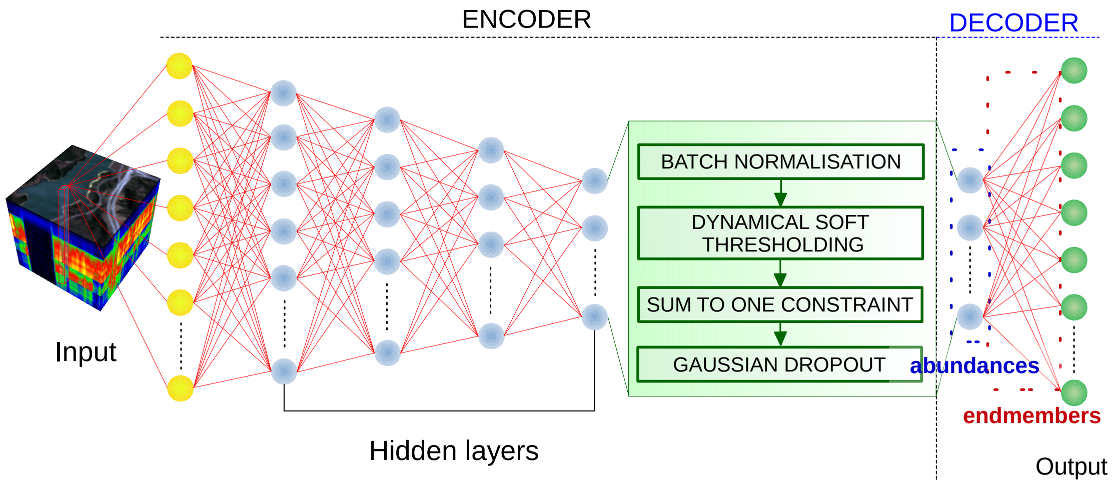


Figure 3.2: The scheme of the *original* architecture described in [114].

Let us suppose that $\mathcal{B} = \{\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_{b_s}}\}$ is a batch of B–dimensional vectors, i.e. $\mathbf{x_i} = [x_i^1, x_i^2, ..., x_i^B]$, where $i \in \{1, ..., b_s\}$. The batch normalization transform is performed over b_s elements of the batch $\mathcal{B}$ [67]. After forward propagation through some layers with batch $\mathcal{B}$ we have a set of b_s activation values for the k–th neuron: $\mathcal{B}_{act}^k = \{g_1^k, g_2^k, ..., g_{b_s}^k\}$, $k \in \{1, 2, ..., M\}$, where M is the number of neurons in a given network layer. During the batch normalization step, consecutive activation values are independently scaled and shifted according to learned parameters γ and β. Furthermore, the mean and variance of batch activations are calculated:

$$\tilde{g_i}^k = \frac{g_i^k - \mu_{\mathcal{B}}^k}{\sqrt{(\sigma_{\mathcal{B}}^k)^2 + \epsilon}}. \tag{3.19}$$

In Equation 3.19, $\mu_{\mathcal{B}}^k = \frac{1}{b_s}\sum_{i=1}^{b_s} g_i^k$ is a mean while $(\sigma_{\mathcal{B}}^k)^2 = \frac{1}{b_s}\sum_{i=1}^{b_s}\left(g_i^k - \mu_{\mathcal{B}}^k\right)^2$ is a variance calculated over batch activations and $\epsilon > 0$ is a constant value. Finally, the rescaled i–th activation value for the k–th neuron, ${}_{(BN)}\tilde{g_i}^k$, where $i \in \{1, ..., b_s\}$, can be expressed as follows:

$${}_{(BN)}\tilde{g_i}^k = \gamma \cdot \tilde{g_i}^k + \beta. \tag{3.20}$$

After the applying batch normalization, a soft thresholding ReLU with a learnable $\boldsymbol{\alpha}$ is used. Let us denote by ${}_{(BN)}\mathbf{\tilde{g}_i}$ a vector of transformed neurons values for the i–th batch sample: ${}_{(BN)}\mathbf{\tilde{g}_i} = [{}_{(BN)}\tilde{g}_i{}^1, {}_{(BN)}\tilde{g}_i{}^2, ..., {}_{(BN)}\tilde{g}_i{}^M]$. Then

$$ {}_{(ST)}\mathbf{\tilde{g}_i} = \max\Big(\mathbf{0}, {}_{(BN)}\mathbf{\tilde{g}_i} - \boldsymbol{\alpha}\Big), \tag{3.21}$$

where $\mathbf{0}$ is a zero vector while $\boldsymbol{\alpha} = [\alpha^1, \alpha^2, ..., \alpha^M]$ is a vector of trainable parameters. In the next step, to ensure that the sum-to-one constraint for abundances will be met (Equation 3.17), a normalization is performed:

$$\forall j \in \{1, 2, ..., M\} \quad {}_{(norm)}\tilde{g}_i{}^j = \frac{{}_{(ST)}\tilde{g}_i{}^j}{\sum\limits_{m=1}^{M} {}_{(ST)}\tilde{g}_i{}^m}. \tag{3.22}$$

We assume that such normalized values represent fractional abundances for the i–th pixel from batch $\mathcal{B}$. They are further multiplied by Gaussian noise [90], but only during the training phase:

$$\forall j \in \{1, 2, ..., M\} \quad {}_{(GD)}\tilde{g}_i{}^j = {}_{(norm)}\tilde{g}_i{}^j \cdot N(1, \alpha^2), \tag{3.23}$$

where $N(1, \alpha^2)$ is a pseudorandom value of a normal distribution with mean equal to 1 and variance equal to α^2 while $\alpha > 0$ is a hyperparameter, ${}_{(GD)}\mathbf{\tilde{g}_i} = [{}_{(GD)}\tilde{g}_i{}^1, {}_{(GD)}\tilde{g}_i{}^2, ..., {}_{(GD)}\tilde{g}_i{}^M]$.

Finally, in the decoder part, there is only one linear layer. Multiplication of ${}_{(GD)}\mathbf{\tilde{g}_i}$ with the weights of the layer represents the reconstruction of the i–th pixel while the weights are considered as endmembers.

We also prepared a simplified version of the *original* architecture called *basic*, which is presented in Figure 3.3. The encoder part of this network has only two hidden layers, where the first one has $p \cdot S$ neurons and p is a hyperparameter, while the second one has exactly S neurons, which correspond to the number of endmembers in a given dataset. In both layers, the ReLU activation function is used. From among the additional layers, we have only left the normalization layer. The output of this layer is an estimation of abundance values and it is multiplied by the weights of the single decoder layer. As previously, such a product is a pixel reconstruction.

3.5 Stability analysis of networks for unmixing

3.5.1 Datasets

We selected two datasets for hyperspectral unmixing – the Samson and the Jasper Ridge [174]. We could not choose a dataset discussed in Chapter 2 because we do not have ground truth labeling for mixed pixels.

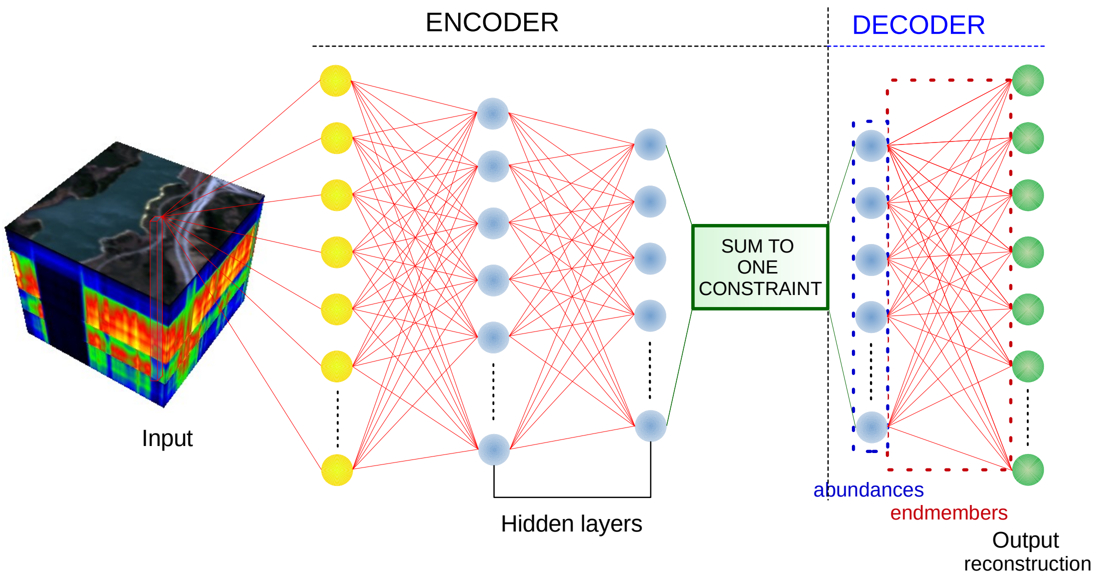
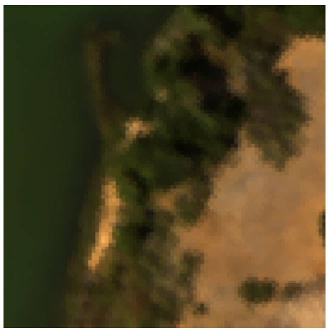
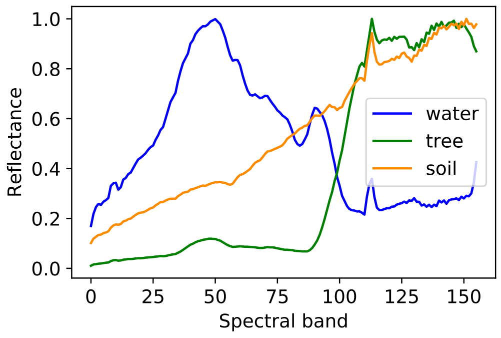


Figure 3.3: The scheme of the *basic* autoencoder architecture.

The Samson image was captured by the SAMSON (Spectroscopic Aerial Mapping System with On-board Navigation) sensor and has spatial resolution 95×95 pixels. Each pixel has 156 spectral bands ranging from 401 to 889 nm. This means that the spectral resolution is about 3.13 nm. It is a subimage of a larger dataset (952×952 pixels) so it is less computationally expensive and is used as a benchmark in many articles. The Samson dataset contains three endmembers: water, tree and soil. The ground truth of the endmember spectra with a false-color image of the analyzed area is presented in Figure 3.4.


(a) False-color image

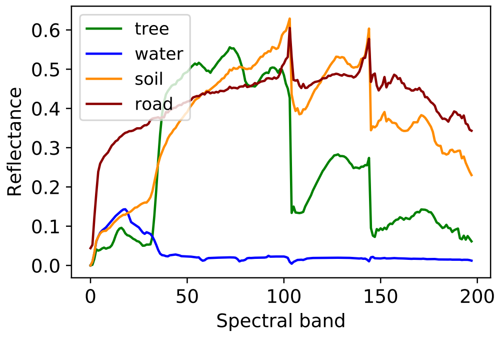


(b) Ground truth of endmembers

Figure 3.4: The presentation of the Samson dataset with the corresponding ground truth of the endmember spectra.

The Jasper Ridge is a second well-known hyperspectral dataset. Its acquisition was performed with the AVIRIS sensor (Airborne Visible/Infrared Imaging Spectrometer). The spatial resolution of this image is equal to 100×100 pixels and is a subimage of the dataset with 512×614 pixels. It covers the electromagnetic spectrum from 380 to 2500 nm. Originally, the dataset has 224 spectral bands which correspond to a spectral resolution of about 9.46 nm but 1-3, 108-112, 154-166 and 220-224 bands were dropped. This is a common practice due to water vapor absorption and atmospheric effects [113, 174, 175]. Four endmembers are present in the dataset: tree, water, soil and road. Their ground truth with a false-color image of the area considered is depicted in Figure 3.5.

(a) False-color image



(b) Ground truth of endmembers

Figure 3.5: The presentation of the Jasper Ridge dataset with the corresponding ground truth of the endmember spectra.

3.5.2 Schema of experiments

Let us suppose that K is a given architecture of the autoencoder designated for hyperspectral unmixing with a given set of hyperparameters. We have to check whether there is a statistically significant difference among results obtained by different weight initializations of models, i.e. sets of initial parameters of all network layers. We prepared 10 different experiment scenarios, including various architectures, datasets and sets of hyperparameters. Let us denote by K_j^i the j–th model of the i–th weight initialization in a given experiment scenario, where $i \in \{1, 2, ..., N\}$ and $j \in \{1, 2, ..., R\}$. N is a number of different initial sets of weight values for a given scenario and R is a number of training runs. A single model K_j^i represents the model after j–th training when the i–th set of initial values of weights was taken. This means that for each model initialization R different models are trained. In our experiments for each scenario, we generated $N = 50$ random initializations and performed $R = 50$ training runs to verify whether various initial sets of weights lead to significantly different results. Therefore, we received 2500 individual models. Several runs per each set of weights were performed due to randomness during the training process resulting from various orders of data batches. This leads to different weights' values after a backpropagation. We also applied four weight initialization methods and tested all approaches for each experimental setup. It leads to $10,000$ various models per scenario. In total, we conducted $100,000$ single experiment runs.

3.5.3 Metrics

Let us denote a hyperspectral dataset by a matrix $\mathbf{X} \in \mathbb{R}^{P \times B}$, where P is the number of pixels and B is the number of spectral bands. Each pixel of $\mathbf{X}$ is a B–element vector $\mathbf{x}_p = \left[x_p^1, x_p^2, ..., x_p^B\right]$, where $p \in \{1, ..., P\}$. Similarly, $\mathbf{Y} \in \mathbb{R}^{P \times B}$ is a matrix of the reconstructed image, where each element is a B–dimensional vector, $\mathbf{y_p} = \left[y_p^1, y_p^2, ..., y_p^B\right]$ and also $p \in \{1, ..., P\}$. We define the Root Mean Squared Error (RMSE) between $\mathbf{X}$ and $\mathbf{Y}$ for the model K_j^i as follows:

$$\text{RMSE}(K_j^i; \mathbf{X}, \mathbf{Y}) = \frac{1}{P} \sum_{p=1}^{P} \sqrt{\frac{1}{B} \sum_{l=1}^{B} (x_l^p - y_l^p)^2}. \tag{3.24}$$

For the calculation of endmember error, we use the Spectral Angle Distance (SAD) measure. Let us assume that the image $\mathbf{X}$ has S endmembers defined in the ground truth matrix $\mathbf{E_{GT}} = \left[\mathbf{e^1_{GT}}, \mathbf{e^2_{GT}}, ..., \mathbf{e^S_{GT}}\right]^\top$. A vector $\mathbf{e^k_{GT}} = \left[e_{GT}^{k,1}, e_{GT}^{k,2}, ..., e_{GT}^{k,B}\right]$ represents k–th endmember, i.e. $k \in \{1, 2, ..., S\}$. A matrix $\mathbf{E_{rec}}$ expresses the endmembers reconstructed by the model K_j^i, $\mathbf{E_{rec}} = \left[\mathbf{e^1_{rec}}, \mathbf{e^2_{rec}}, ..., \mathbf{e^S_{rec}}\right]^\top$. As previously, consecutive vectors are reconstructed endmembers, $\mathbf{e^k_{rec}} = \left[e_{rec}^{k,1}, e_{rec}^{k,2}, ..., e_{rec}^{k,B}\right]$. Finally, the endmember error is defined in the following way:

$$\text{SAD}(K_j^i; \mathbf{E_{GT}}, \mathbf{E_{rec}}) = \frac{1}{S} \sum_{s=1}^{S} \arccos\left(\frac{\mathbf{e^s_{GT}} \cdot \mathbf{e^s_{rec}}}{\|e^s_{GT}\|_2 \cdot \|e^s_{rec}\|_2}\right), \tag{3.25}$$

where $\mathbf{e^s_{GT}} \cdot \mathbf{e^s_{rec}}$ denotes a scalar product of the s–th reconstructed endmember and its corresponding ground truth vector. Furthermore, a symbol $\|\cdot\|_2$ is an Euclidean norm of a given vector.

We also calculate the abundance error between a reconstructed abundance matrix $\mathbf{A_{rec}} \in \mathbb{R}^{P \times B}$ and a ground truth matrix $\mathbf{A_{GT}} \in \mathbb{R}^{P \times B}$ using the Root Mean Square Error:

$$\text{RMSE}(K_j^i; \mathbf{A_{GT}}, \mathbf{A_{rec}}) = \frac{1}{P} \sum_{p=1}^{P} \sqrt{\frac{1}{S} \sum_{l=1}^{S} \left(a_{GT}^{p,l} - a_{rec}^{p,l}\right)^2}, \tag{3.26}$$

where $\mathbf{a}^{\mathbf{p}}_{\mathbf{GT}} = \left[a^{p,1}_{GT}, a^{p,2}_{GT}, ..., a^{p,S}_{GT}\right]$ is an abundance ground truth for the p–th vector of $\mathbf{A}_{\mathbf{GT}}$ and a vector $\mathbf{a}^{\mathbf{p}}_{\mathbf{rec}} = \left[a^{p,1}_{rec}, a^{p,2}_{rec}, ..., a^{p,S}_{rec}\right]$ expresses the reconstructed abundance coefficients for the p–th pixel.

3.5.4 Scenarios

We prepared 10 experiment scenarios for two autoencoder architectures: *original* and *basic*, two datasets: the Samson and the Jasper Ridge two loss functions: mean squared error (MSE, Equation 1.13) and Spectral Angle Distance (SAD, Equation 3.25). Moreover, experiments were performed for different sets of hyperparameters. Most of them were selected using the Ray Tune optimizer [92] while in a few cases they were chosen according to the Authors of the *original* architecture [114]. If the hyperparameters optimization procedure was performed, optimal values of batch size and learning rate were sought. Additionally, we searched for a number of neurons in the first encoder layer for the *basic* network and the Gaussian Dropout value for the *original* network. Detailed information on selected hyperparameters is presented in Table 3.1. Furthermore, in each experiment, four weight initialization methods have been applied: He [60] with a uniform / normal distribution (KHU / KHN) and Glorot [50] with a uniform / normal distribution (XGU / XGN)[1]. All training sessions were performed using the Adam optimizer. The source code for experiments was written in Python 3.7.6 and libraries like PyTorch [119], SciPy [154], Ray Tune [92], scikit-posthocs [148], NumPy [59], Pandas [128], Matplotlib [66], Seaborn [157] and others.

3.5.5 Statistical tests

We decided to perform statistical tests to check whether there exist statistically significant differences between various models trained on the same set of hyperparameters. We could not use ANOVA analysis due to failure to meet the assumptions about equality of variance in all model populations, according to the results of Levene's test [84, 108]. Instead, we selected Kruskal-Wallis H-test [78] for several independent samples which checks whether the expected value of at least one data sample (i.e. single population of models) is significantly different than the expected value of at least one other sample. If we reject the null hypothesis of the H-test we only know that there exists at least one pair of samples different from the others but we do not exactly know which one. It is possible to specify it by performing pairwise comparisons using the Conover-Iman post-hoc test [35, 156]. Both tests are described in detail in the Appendix.

[1]In the case of KHU approach we used a default PyTorch initialization. The main difference is that biases were not initialized with zero values.

Table 3.1: The list of experiments performed with selected hyperparameters. The column titled "encoder" is related to the *basic* architecture and contains information about the number of neurons in the first encoder layer. S denotes the number of endmembers in a given dataset. GD is an abbreviation of Gaussian Dropout and is only used in the *original* architecture.

exp. ID	architecture	hyperparameters	loss	dataset	encoder	batch size	learning rate	GD
1	*original*	Ray Tune	MSE	Samson	–	100	0.01	0.
2	*original*	Ray Tune	SAD	Samson	–	100	0.01	0.
3	*original*	article [114]	SAD	Samson	–	20	0.01	0.1
4	*basic*	Ray Tune	MSE	Samson	$10S$	4	0.0001	–
5	*basic*	Ray Tune	SAD	Samson	$20S$	4	0.0001	–
6	*original*	Ray Tune	MSE	Jasper Ridge	–	100	0.01	0.
7	*original*	Ray Tune	SAD	Jasper Ridge	–	100	0.01	0.
8	*original*	article [114]	MSE	Jasper Ridge	–	5	0.01	0.1
9	*original*	article [114]	SAD	Jasper Ridge	–	5	0.01	0.1
10	*basic*	Ray Tune	MSE	Jasper Ridge	$10S$	20	0.001	–

3.6 Results

The main results of this chapter are related to the study on the impact of weights initialization on network performance in the case of autoencoders for hyperspectral unmixing. They were provided by two statistical tests: the Kruskal-Wallis H-test and the Conover-Iman post-hoc test. The results of the experiments described in Section 3.5.4 are presented in Table 3.2. Each row corresponds to a single experiment scenario, while columns show scores for various weight initialization methods and the Kruskal Wallis H-test. Moreover, values of the *ph* metric are presented. *ph* can be expressed as the ratio of pairs that are statistically significantly different (in terms of the Conover-Iman test) to the total number of pairs that were compared. The lower the coefficient, the more consistent the results among different model populations.

It can be seen that in all except two cases (Experiment 9 with KHU and XGU initialization methods) the H_0 hypothesis of the Kruskal-Wallis H-test was rejected. This means that for almost all experimental setups statistically significant differences were noted between populations of model training results. On the basis of this observation, it is possible to conclude that weight initialization in the presented autoencoder networks has an important impact on the final reconstruction error. Despite the fact that in a given scenario, all models were trained using the same hyperparameters, various sets of initial weights lead to different results.

In addition to the results of statistical tests, we demonstrate box and whisker plots for all model reconstruction errors in terms of RMSE (Figures 3.6–3.7) and endmember errors in terms of the SAD function (Figures 3.8–3.9). A single plot ag-

Table 3.2: Results of experiments described in detail in Table 3.1. H-stat denotes the statistic of the Kruskal Wallis H-test while log p-val is the logarithm of the corresponding p-value. The significance level was set to 0.05. Cases where the p-value was greater than 0.05 were marked in bold. Very small p-values are replaced by '− inf'. The *ph* values represent the ratio of pairs that are statistically significantly different from the total number of pairs compared.

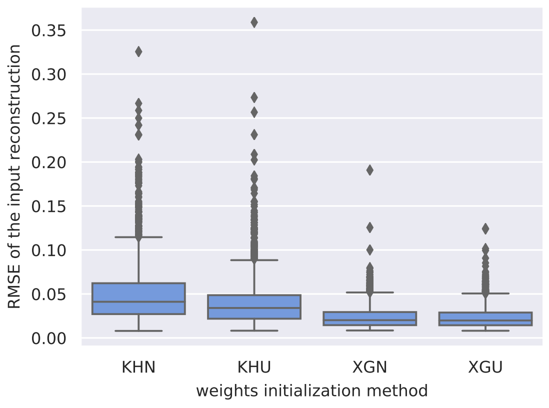
init.	He normal (KHN)			He uniform (KHU)			Glorot normal (XGN)			Glorot uniform (XGU)		
exp. ID	H-stat	log p-val	*ph*	H-stat	log p-val	*ph*	H-stat	log p-val	*ph*	H-stat	log p-val	*ph*
1	200.9	−45.07	0.33	243.2	−61.76	0.42	78.8	−5.41	0.12	70.4	−3.72	0.10
2	891.2	−355.38	0.70	443.2	−147.76	0.56	625.9	−231.04	0.64	660.3	−246.96	0.66
3	763.8	−295.33	0.67	267.8	−71.82	0.41	239.1	−60.11	0.39	180.9	−37.49	0.33
4	2185.8	− inf	0.88	2025.3	− inf	0.72	1997.6	− inf	0.75	2141.5	− inf	0.76
5	1954.3	− inf	0.88	2093.4	− inf	0.90	1840.8	− inf	0.87	1777.2	− inf	0.85
6	134.0	−20.95	0.24	75.8	−4.79	0.11	76.3	−4.90	0.12	98.8	−10.32	0.16
7	903.8	−361.36	0.71	761.1	−294.07	0.67	953.8	−385.10	0.70	871.0	−345.85	0.69
8	77.3	−5.09	0.12	75.4	−4.71	0.11	78.4	−5.33	0.12	93.3	−8.88	0.16
9	69.2	−3.50	0.10	**66.2**	**−2.98**	–	74.2	−4.46	0.11	**47.9**	**−0.65**	–
10	1767.3	− inf	0.85	2041.9	− inf	0.82	1344.6	−572.45	0.70	1155.1	−481.29	0.73

gregates results from 10,000 model instances, 2500 per each initialization approach. The box covers the area between the 25th (Q_1) and 75th (Q_3) percentile of the data and a horizontal line marks the 50th percentile. The whiskers contain observations from the range $[Q_1 - 1.5 \cdot IQR, Q_3 + 1.5 \cdot IQR]$, where $IQR = Q3 - Q1$ [73, 4]. Other samples are marked separately as gray diamonds. One can see differences between the results from different weight initialization methods.

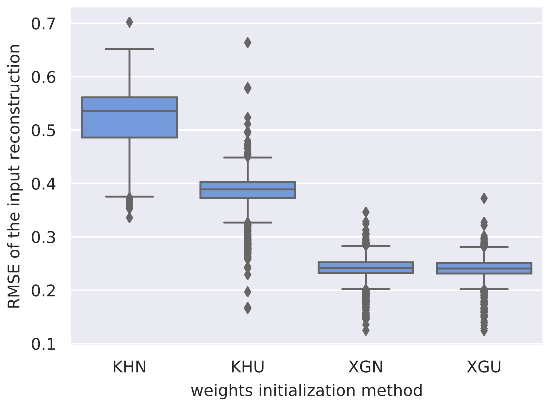
We notice a high variability in reconstruction errors in terms of RMSE, especially in the case of Experiments 2, 3 and 7. Furthermore, for these scenarios, the KHN method achieved not only the largest errors values but also the highest variance of RMSE scores. The most stable reconstructions were performed by autoencoders in Experiments 1, 6, 8 and 9 which coincide with corresponding low *ph* coefficient values. Similarly for the endmember reconstruction metric, in the case of some scenarios, especially for Experiments 1, 4 and 7, there are important differences between results for various initialization techniques. The plots presented indicate that the most consistent SAD metric values are achieved in Experiments 8 and 9. Furthermore, in the case of Experiment 3, for all methods, endmember errors are minor and only outlying results exceed 0.01. It should be noted that Experiments 1–5 concern the same dataset while consequently the models of Experiment 3 significantly minimized the SAD metric values, unlike the other scenarios. It means that network hyperparameters, as well as architectures and loss functions, have a substantial impact on the quality of pixel reconstruction and variance of results. The differences are noticed even in situations where only the loss function has changed as in Experiments 1–2.

We also prepared a post-hoc analysis for reconstruction errors using the Conover-Iman test in the cases for which the H_0 hypothesis of the Kruskal-Wallis H-test was rejected, i.e. for all scenarios except KHU and XGU in Experiment 9. The results of the statistical tests for models pairs from different scenarios and initialization methods are depicted in Figures 3.10–3.12. In each plot, consecutive squares represent the results of a single Conover-Iman comparison between given pairs of models (from 1 to 50). Moreover, it is a symmetric matrix. A light yellow square means that the difference between the selected model populations is not statistically significant. Blue squares denote cases in which the H_0 hypothesis was rejected in favor of the alternative hypothesis that a given pair of model populations is different, in terms of the Conover-Iman test (the darker the blue, the lower p-value of the performed post-hoc test). Furthermore, these plots correspond to the *ph* scores in Table 3.2. It can be noticed that the most consistent results come from Experiments 6, 8 and 9. In these cases, models were trained on the Jasper Ridge dataset using the *original* architecture. In Experiment 1, a major difference can be seen between the He and Glorot initialization methods. The values of *ph* coefficient are around 0.1 for the Glorot approach, while they achieve 0.33 and even 0.42 in the case of the He technique. The largest number of differences among models can be observed in Experiments 4 and 5 which were made using the Samson dataset and the *basic* architecture. In these scenarios *ph* scores range between 0.72 and 0.9 which proves the great diversity of results. Large *ph* values were also achieved in Experiment 10 which means that the *basic* architecture leads to the highest variations in results. On the other hand, the *original* architecture, especially with the MSE loss function, yields the most stable results.

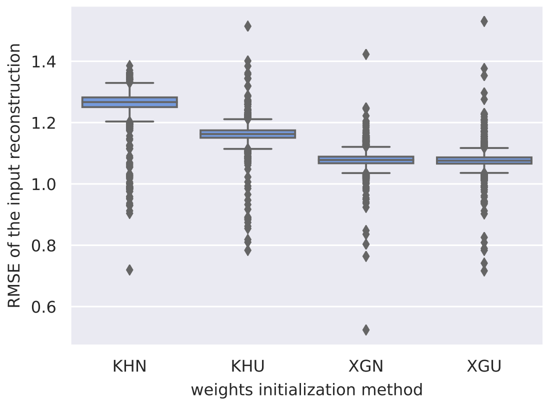
In addition to the autoencoder stability analysis, we also present the results of hyperspectral unmixing for all the experiments performed. Table 3.3 shows mean abundance and endmember errors, according to Equations 3.26, 3.25, respectively. The lowest values of errors were achieved mainly by the Glorot initialization approach, especially with a uniform distribution. For both Samson and Jasper Ridge datasets, the best setup from the unmixing results' point of view was the *original* architecture with the SAD loss function (Experiments 3 and 7). We demonstrate the reconstruction of abundance maps and endmember spectra for two exemplary models. Figure 3.14 depicts a comparison of reproduced abundance maps with ground truth for a single model from Experiment 3 while Figure 3.15 presents normalized spectra of reconstructed endmembers (blue) with their ground truth counterparts (red). Generally, consecutive endmembers and abundance maps were reproduced similarly good. Analogous conclusions can be drawn on the basis of Figures 3.16–3.17 prepared for a model selected from Experiment 9. The biggest



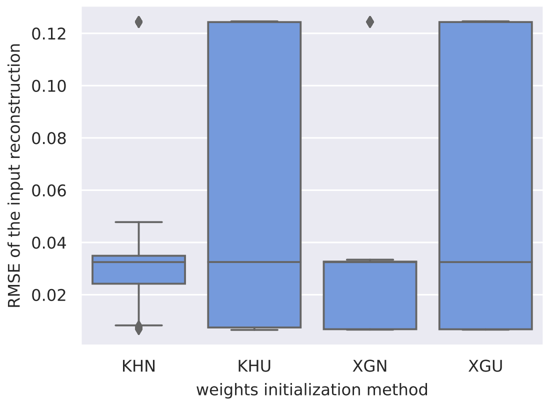
(a) Experiment 1.



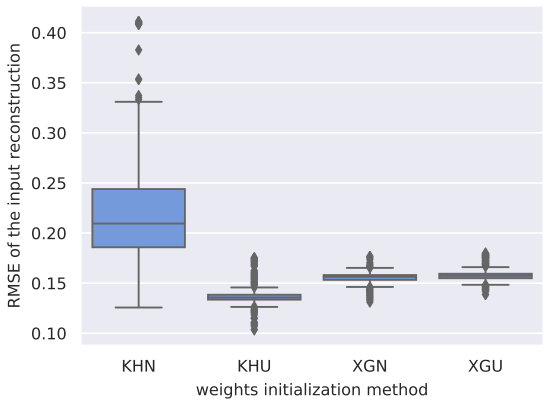
(b) Experiment 2.



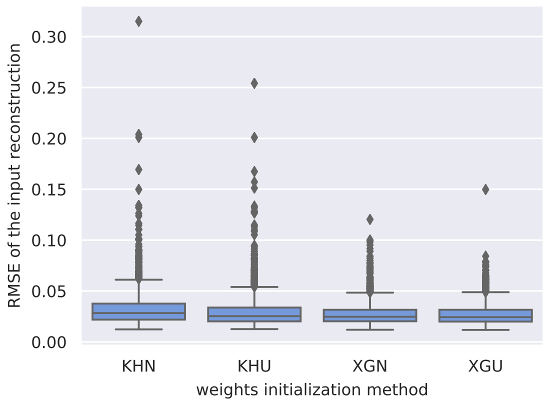
(c) Experiment 3.



(d) Experiment 4.

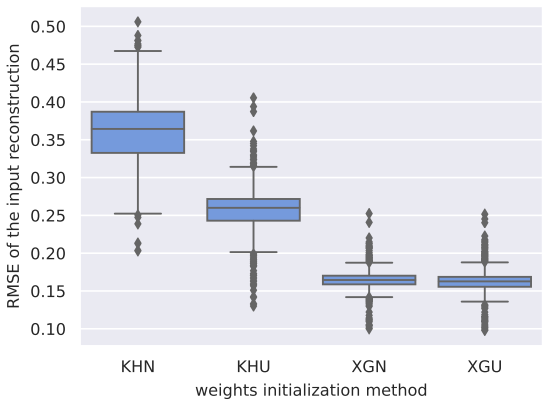


(e) Experiment 5.

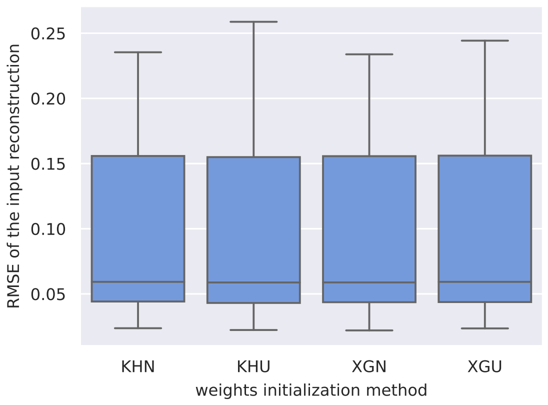


(f) Experiment 6.

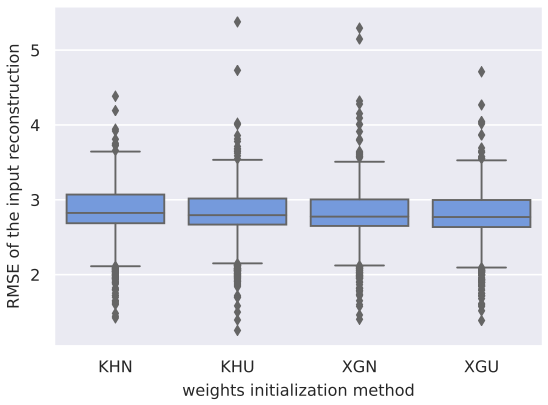
Figure 3.6: Box and whisker plots for Root Mean Square Error (RMSE) of the input reconstruction for Experiments 1-6.



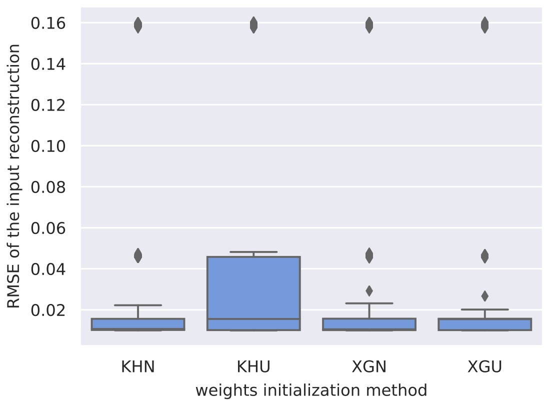
(a) Experiment 7.



(b) Experiment 8.

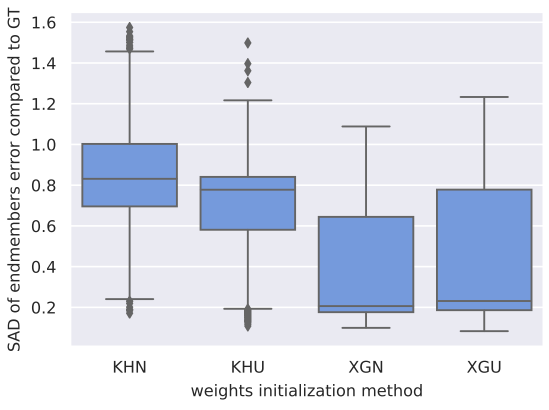


(c) Experiment 9.

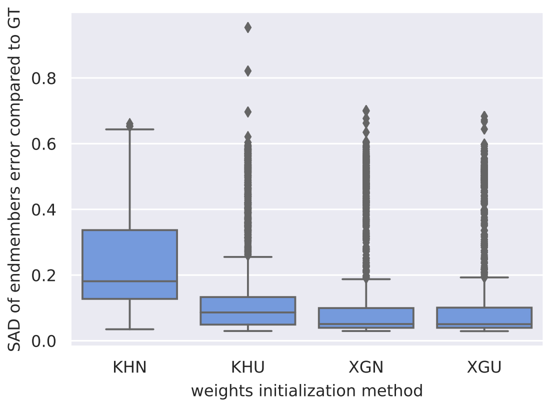


(d) Experiment 10.

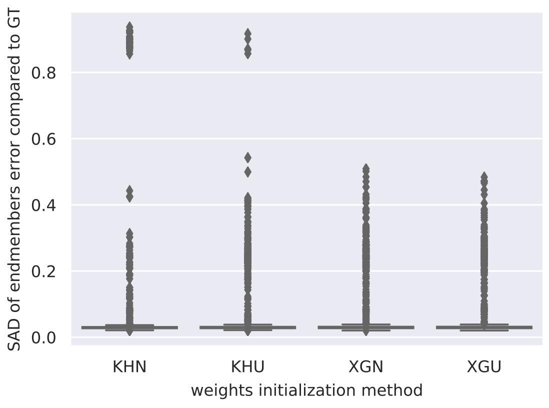
Figure 3.7: Box and whisker plots for Root Mean Square Error (RMSE) of the input reconstruction for Experiments 7-10.



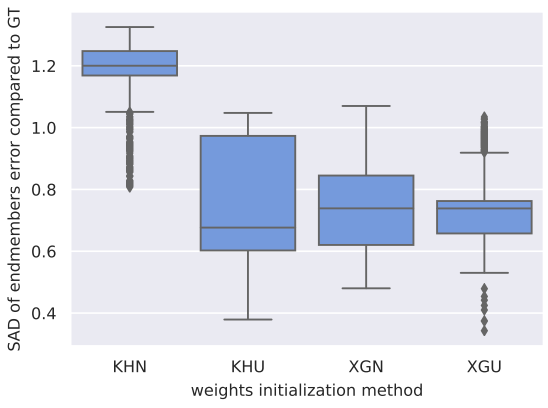
(a) Experiment 1.



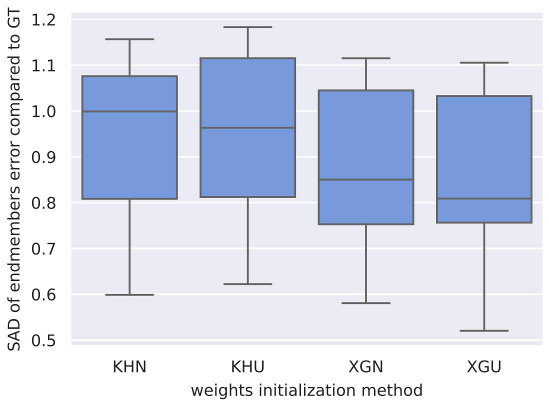
(b) Experiment 2.



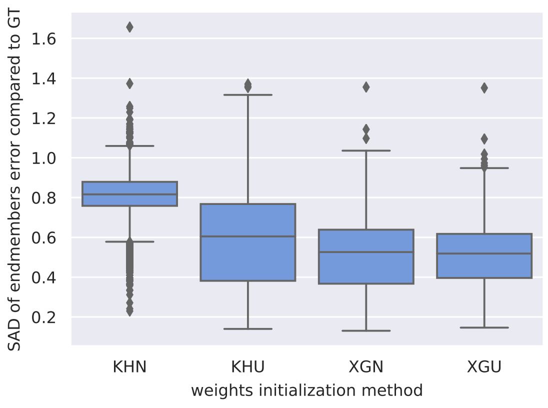
(c) Experiment 3.



(d) Experiment 4.

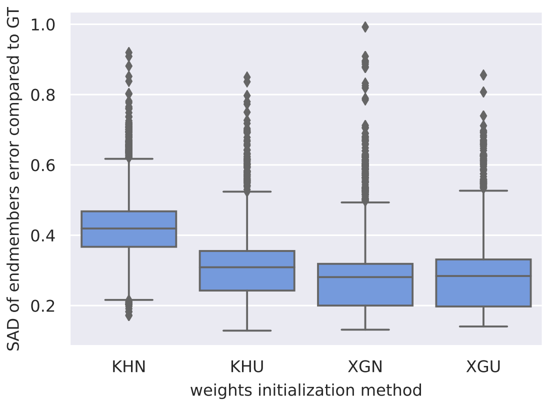


(e) Experiment 5.

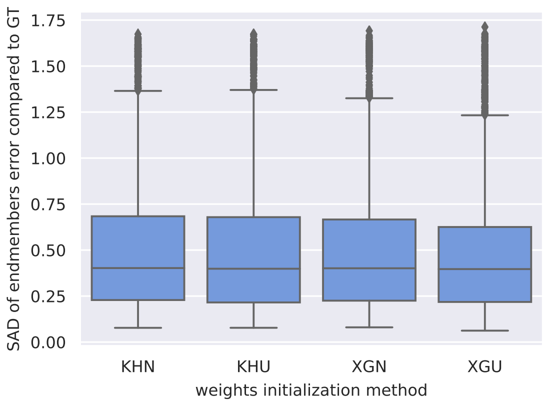


(f) Experiment 6.

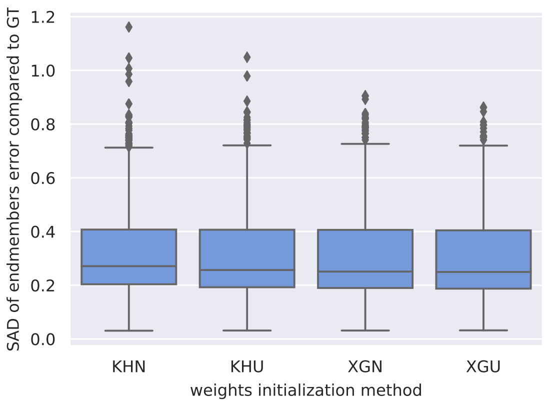
Figure 3.8: Box and whisker plots for Spectral Angle Distance (SAD) of endmember error compared to the ground truth for Experiments 1-6.



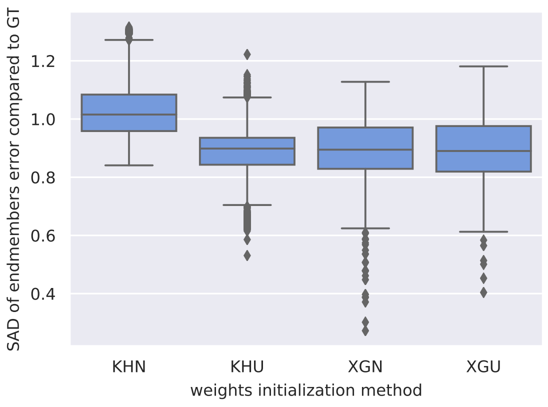
(a) Experiment 7.



(b) Experiment 8.

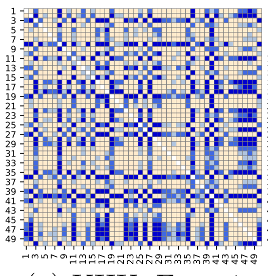


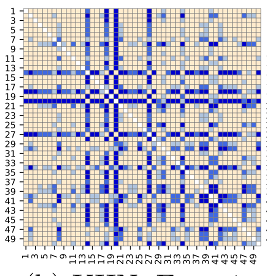
(c) Experiment 9.

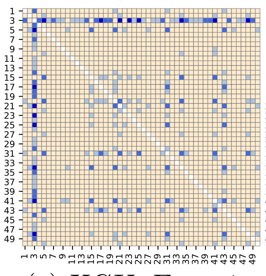


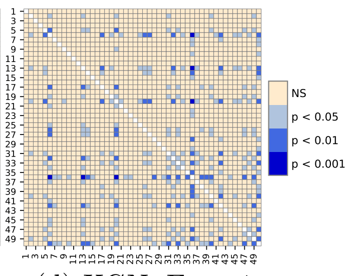
(d) Experiment 10.

Figure 3.9: Box and whisker plots for Spectral Angle Distance (SAD) of endmembers error compared to the ground truth for Experiments 7-10.

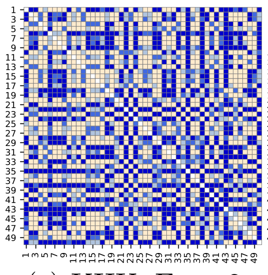

(a) KHU, Exp. 1

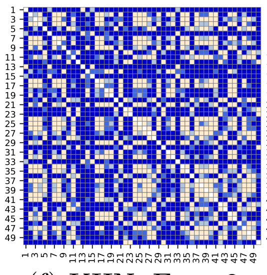

(b) KHN, Exp. 1

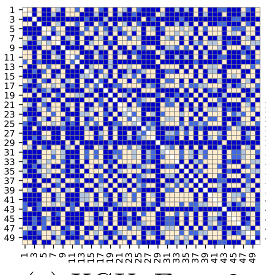

(c) XGU, Exp. 1

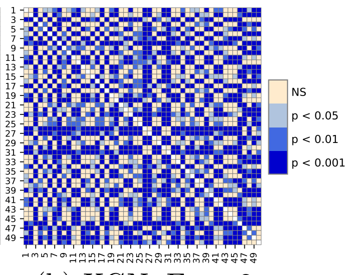


(d) XGN, Exp. 1

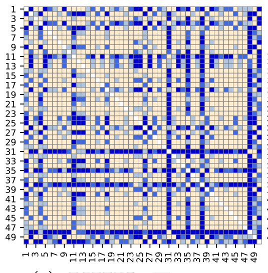

(e) KHU, Exp. 2

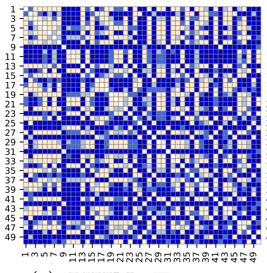

(f) KHN, Exp. 2

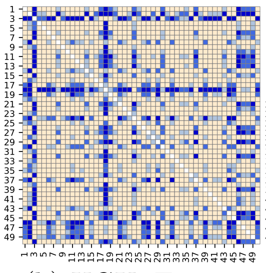

(g) XGU, Exp. 2

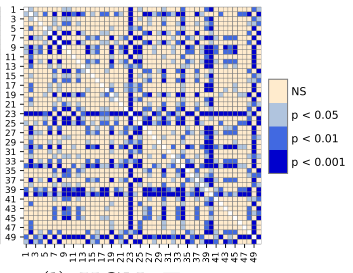


(h) XGN, Exp. 2


(i) KHU, Exp. 3


(j) KHN, Exp. 3


(k) XGU, Exp. 3



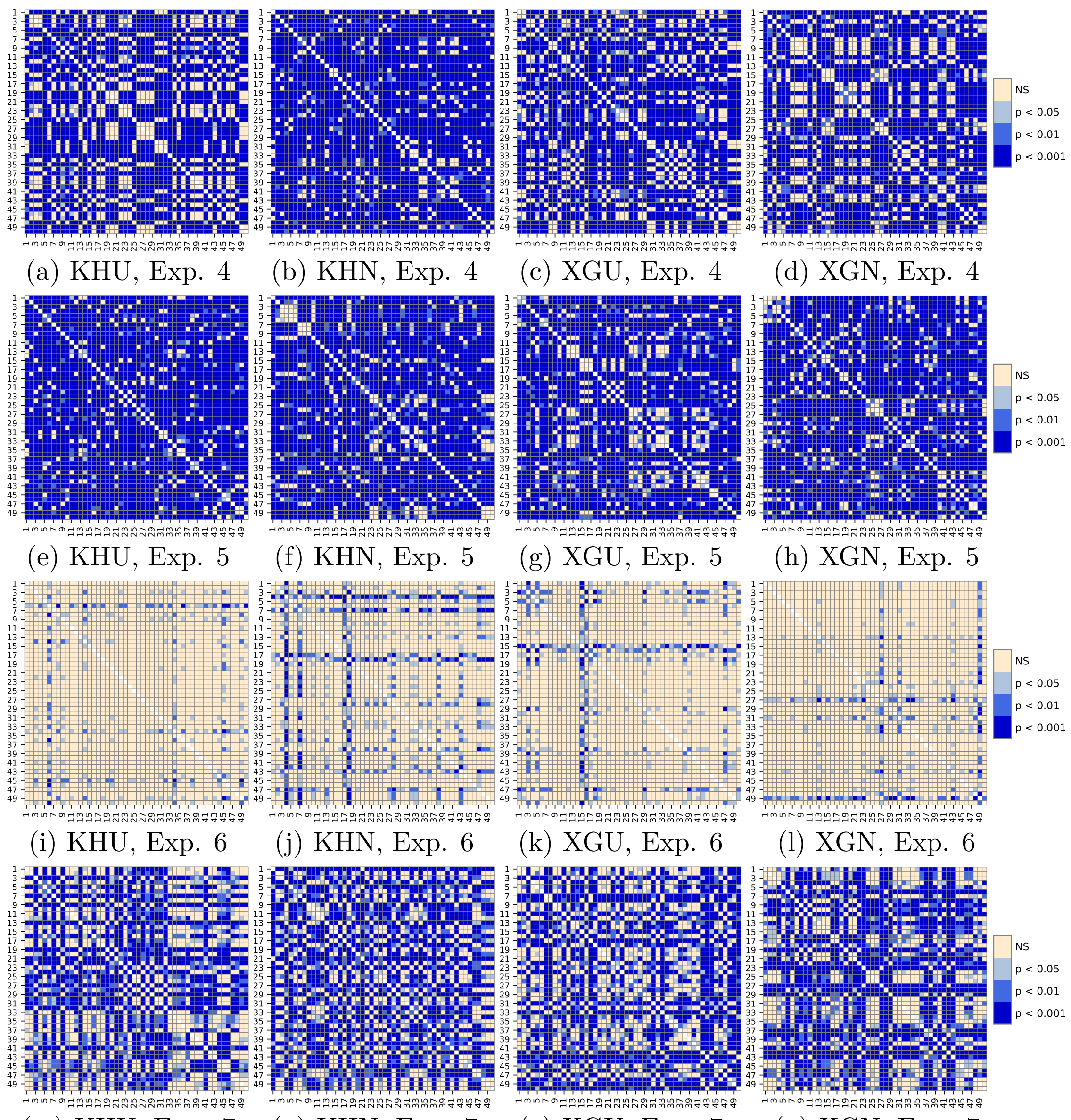
(l) XGN, Exp. 3

Figure 3.10: Results of Conover-Iman post-hoc test for consecutive pairs of models from Experiments 1–3.

differences between the spectra are visible for the soil endmember but the SAD error, similarly to other endmembers, does not exceed 0.008. It is due to the fact that the SAD metric is scale invariant because it only measures the spectral angle between the reconstructed and the real endmember. Thus, one does not have to reduce the Euclidean distance between spectra.

3.7 Discussion

Based on the results of the experiments, we indicate that initial values of weights in presented autoencoder networks have a significant impact on the final reconstruction error. Unfavorable sets of weights can lead to lower-performing models. The dependency between the initial model parameters and the quality of the pixel



(a) KHU, Exp. 4 (b) KHN, Exp. 4 (c) XGU, Exp. 4 (d) XGN, Exp. 4

(e) KHU, Exp. 5 (f) KHN, Exp. 5 (g) XGU, Exp. 5 (h) XGN, Exp. 5

(i) KHU, Exp. 6 (j) KHN, Exp. 6 (k) XGU, Exp. 6 (l) XGN, Exp. 6

(m) KHU, Exp. 7 (n) KHN, Exp. 7 (o) XGU, Exp. 7 (p) XGN, Exp. 7

Figure 3.11: Results of Conover-Iman post-hoc test for consecutive pairs of models from Experiments 4–7.

reconstruction after training has been statistically confirmed. This phenomenon may have a negative influence on the optimization of network hyperparameters. There is a possibility that a good set of hyperparameters will be rejected because the weights drawn are unfortunate. It can be misleading for such algorithms as grid search, which can select a suboptimal solution.

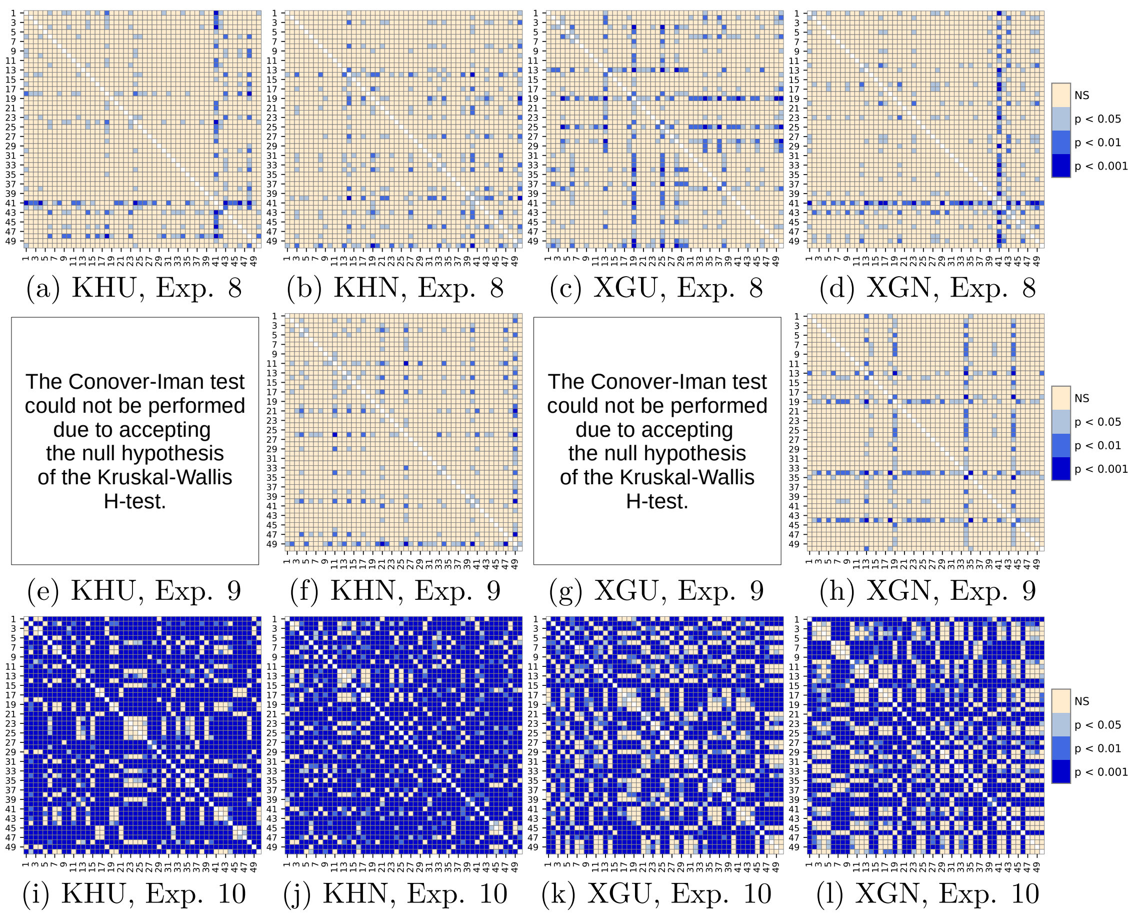


Figure 3.12: Results of Conover-Iman post-hoc test for consecutive pairs of models from Experiments 8–10.

We can compare the results of hyperspectral unmixing for corresponding pairs of experimental settings, i.e. with the same dataset, architecture and hyperparameters except for loss function (MSE or SAD were selected). For Experiments 1 and 2, performed on the Samson dataset, the *original* architecture and hyperparameters optimized using Ray Tune, the SAD loss function led to considerably lower values of both abundance and endmember errors. A similar situation is related to Experiments 6 and 7, where hyperparameters were prepared as previously, but the dataset was Jasper Ridge. In the case of Experiments 8 and 9, in which network settings came from the article [114], also the SAD function achieved better results, but the differences in results were lower, especially for abundance errors. Furthermore, one can notice that standard deviations in endmember errors for results with MSE loss were very high and almost equal to the mean values. The situation is different for Experiments 4 and 5 on the Samson dataset and the *basic*

Table 3.3: Results of mean endmember error in terms of SAD (Equation 3.25) and mean abundance error in terms of RMSE (Equation 3.26) with standard deviations for different experiment scenarios and weight initialization approaches: He [60] and Glorot [50] with normal or uniform distribution. The results with the smallest error are shown in bold.

init.	He normal (KHN)		He uniform (KHU)		Glorot normal (XGN)		Glorot uniform (XGU)	
Exp. ID	abundances	endmembers	abundances	endmembers	abundances	endmembers	abundances	endmembers
1	0.34 ± 0.0	0.84 ± 0.2	0.36 ± 0.0	0.68 ± 0.2	$\mathbf{0.32 \pm 0.0}$	$\mathbf{0.36 \pm 0.3}$	0.33 ± 0.1	0.43 ± 0.3
2	0.19 ± 0.1	0.23 ± 0.1	0.13 ± 0.1	0.13 ± 0.1	0.10 ± 0.1	0.10 ± 0.1	$\mathbf{0.10 \pm 0.1}$	$\mathbf{0.10 \pm 0.1}$
3	$\mathbf{0.06 \pm 0.1}$	0.05 ± 0.1	0.07 ± 0.1	0.04 ± 0.1	0.07 ± 0.1	0.05 ± 0.1	0.07 ± 0.1	$\mathbf{0.04 \pm 0.1}$
4	0.36 ± 0.0	1.17 ± 0.1	0.36 ± 0.0	0.76 ± 0.2	0.35 ± 0.0	0.75 ± 0.1	$\mathbf{0.34 \pm 0.0}$	$\mathbf{0.73 \pm 0.1}$
5	0.35 ± 0.0	0.95 ± 0.1	0.35 ± 0.0	0.97 ± 0.2	$\mathbf{0.34 \pm 0.0}$	0.88 ± 0.2	0.34 ± 0.0	$\mathbf{0.87 \pm 0.1}$
6	0.29 ± 0.0	0.80 ± 0.1	0.24 ± 0.1	0.58 ± 0.2	0.22 ± 0.0	0.51 ± 0.2	$\mathbf{0.22 \pm 0.0}$	$\mathbf{0.50 \pm 0.2}$
7	0.20 ± 0.0	0.42 ± 0.1	0.17 ± 0.0	0.31 ± 0.1	$\mathbf{0.16 \pm 0.0}$	$\mathbf{0.28 \pm 0.1}$	0.16 ± 0.0	0.28 ± 0.1
8	0.29 ± 0.1	0.51 ± 0.4	$\mathbf{0.29 \pm 0.1}$	0.49 ± 0.4	0.29 ± 0.1	0.48 ± 0.3	0.29 ± 0.1	$\mathbf{0.48 \pm 0.4}$
9	0.27 ± 0.1	0.30 ± 0.1	0.27 ± 0.1	0.29 ± 0.1	0.27 ± 0.1	0.28 ± 0.1	$\mathbf{0.26 \pm 0.1}$	$\mathbf{0.28 \pm 0.1}$
10	0.30 ± 0.0	1.04 ± 0.1	$\mathbf{0.27 \pm 0.0}$	$\mathbf{0.89 \pm 0.1}$	0.29 ± 0.0	0.89 ± 0.1	0.28 ± 0.0	0.89 ± 0.1

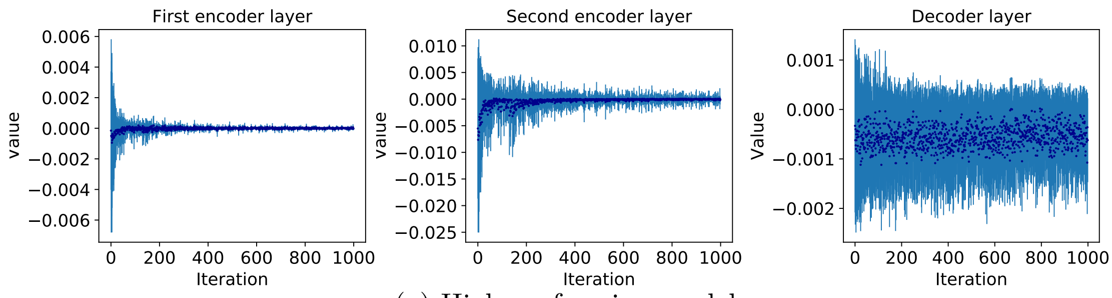
architecture. In this case, the abundance error values were very similar for both loss functions. Moreover, for KHU, XGN and XGU initialization approaches, the reconstruction of endmembers was done better for the MSE loss function. Despite the fact that, in general, models trained with SAD loss were more efficient in the unmixing task than those with MSE, the SAD function has some undesirable properties. One has to remember that after training with this function, all or almost reconstructed points lie outside of the simplex whose vertices are defined by endmembers. We also do not know exact reflectance values because only the spectral angle between vectors is minimized, so, in this case, we rather focus on the normalized reflectance. Furthermore, optimization using the MSE function also reduces the reconstruction error in terms of the SAD loss, but the opposite statement is not true due to the scale invariance of the SAD function.

In general, results for unmixing metrics show a slight advantage of Glorot initialization techniques over He methods, both in terms of abundance and endmember errors. This conclusion can be drawn on the basis of an overall view of the 10 experiments carried out. In the case of the reconstruction of fractional abundances, Glorot approaches achieved the lowest mean errors in 7 out of 10 experiment scenarios while for endmembers, they were the most competitive in 9 out of 10 settings.

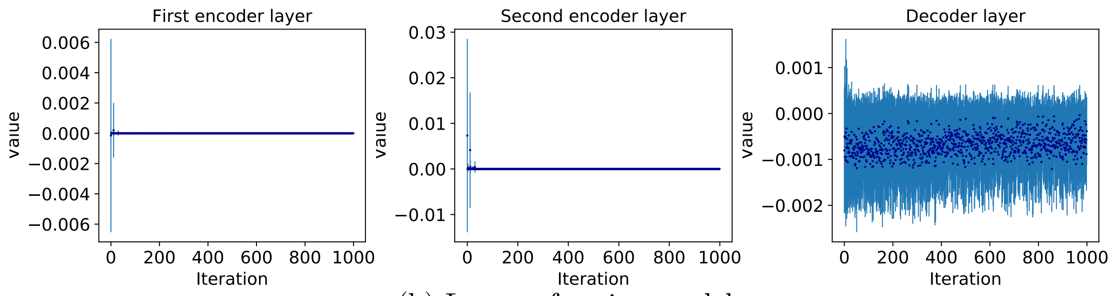
3.7.1 Vanishing gradient

After conducting the experiments, we tried to understand why some models perform worse than others. We focused on the simpler architecture (*basic*) and the

Samson dataset; thus we decided to get an insight into networks from Experiment 4. We observed that during the backpropagation process for some low-performing models, gradients vanished, even during the first iterations of training. Figure 3.13 depicts the gradient values during initial learning iterations of two exemplary models. In this experimental setup, one epoch corresponds to 2257 iterations; therefore the plots present the initial part of the first training epoch. One can observe a vanishing gradient problem in two encoder layers for the model with a higher value of the reconstruction error. The decoder layer gradients values are very similar for the two compared models. Furthermore, in the case of the low-performing model, even during the first iteration, some neurons in the encoder part are not active. We decided to investigate this phenomenon in more detail. The results of the study, as well as network reinitialization methods for improving bad-performing models, will be presented in Section 4.



(a) High-performing model



(b) Low-performing model

Figure 3.13: Mean gradient values with standard deviations for two exemplary models of the *basic* autoencoder, trained in Experiment 4, using KHU initialization. Model (a) achieved RMSE of 0.0067 while model (b) achieved RSME equal to 0.1244.

3.8 Conclusions

In Chapter 2 we performed the optimization of deep learning networks in terms of various architectures for hyperspectral data classification. We identified a problem with the spectral mixing of pixels. Therefore, as a continuation of research, in this chapter, we considered autoencoder networks which are state-of-the-art architectures for hyperspectral unmixing [113, 114, 142]. We evaluated 10 experiment scenarios using different sets of hyperparameters, including diverse architectures, weight initialization methods and loss functions. The remaining hyperparameters, like batch size, learning rate, the number of neurons in the first layer of the *basic* architecture and the parameter of Gaussian Dropout for the *original* were optimized using Ray Tune. They were also compared with the hyperparameters from Palsson's article [114]. Totally, we trained $100,000$ individual models for two well-known datasets for hyperspectral unmixing: Samson and Jasper Ridge. In the example of the spectral unmixing problem, we showed that the choice of architectures and hyperparameters used for model training has an important impact on network performance, which partially confirms the dissertation thesis.

We identified a problem with the stability of autoencoders, i.e. for the same set of hyperparameters some models achieved lower performance compared to the other ones. We performed the Kruskal-Wallis H-test to verify whether there exists a dependency between weights initialization and the final reconstruction error of a neural network, on the example of autoencoders for hyperspectral unmixing. For cases in which a difference between model populations has been confirmed, we included the Conover-Iman post-hoc test to check which pairs of models are statistically significantly different. We proved that for almost all considered scenarios, weight initialization has a significant impact on the further network performance, despite the differences in the severity of this phenomenon. For some experiments, only a few model populations were outlying while for another subset of scenarios, especially related to the *basic* architecture, most pairs of models were different, according to the results of the Conover-Iman test.

Finally, in some autoencoder models, we found a problem with vanishing gradients. We noticed that in some of the models, gradients were not propagated starting from the first training iterations. We will continue the investigation in the next chapter and we will propose network reinitialization methods that can alleviate the negative impact of this phenomenon.

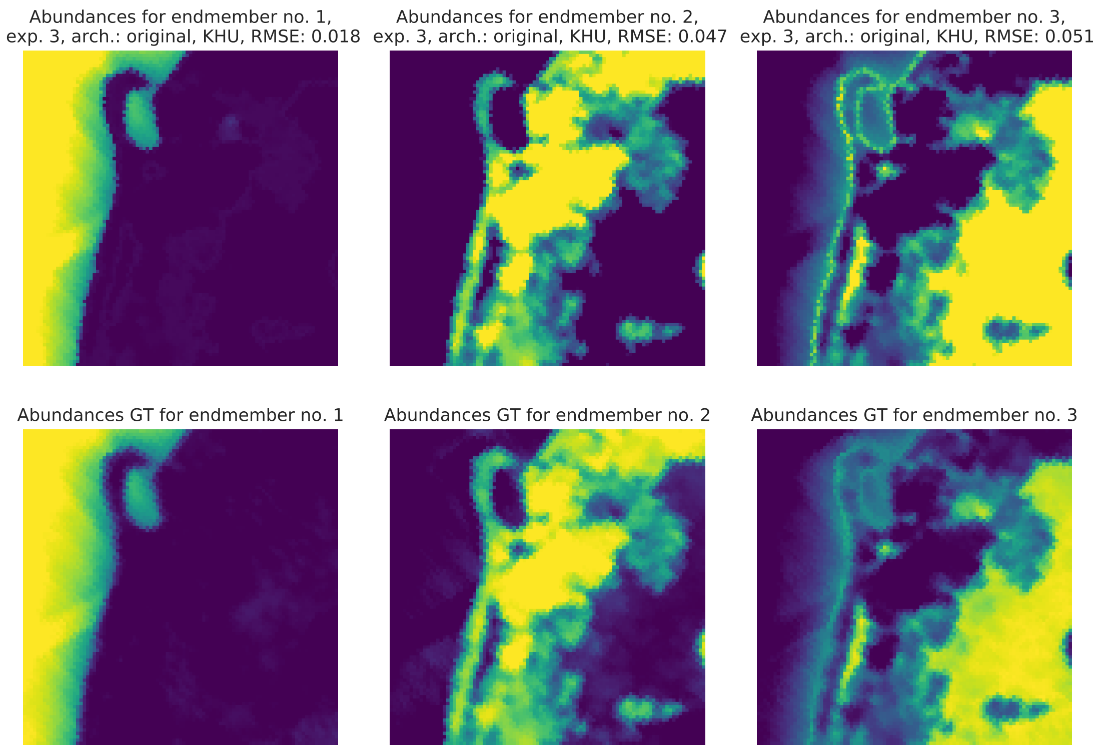


Figure 3.14: Reconstruction of fractional abundances (with ground truth) for a model with the lowest value of endmember error in terms of SAD metric for Experiment 3 with the Samson dataset. The subsequent plots represent abundance coefficients for water, tree and soil.

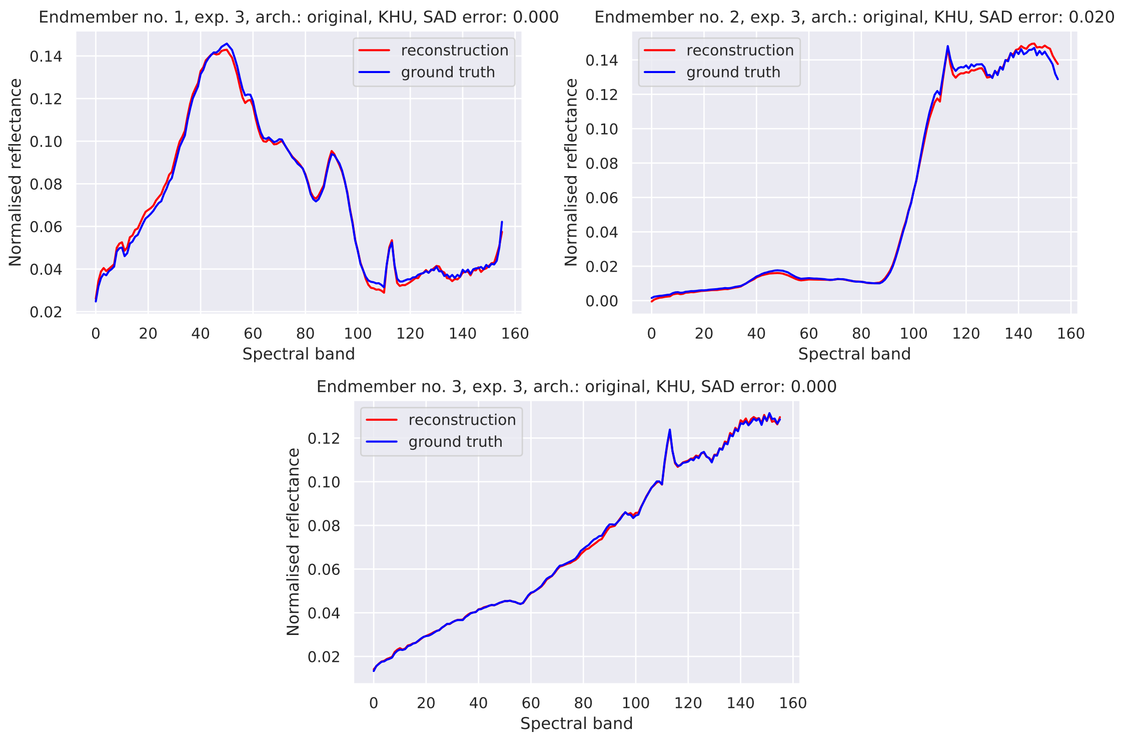


Figure 3.15: Reconstruction of endmembers (with ground truth) for a model with the lowest value of SAD error for Experiment 3 with the Samson dataset. Plots depict normalized reflectance for consecutive spectral bands, i.e. each vector was divided by the value of its Euclidean norm. The subsequent endmembers represent water, tree and soil.

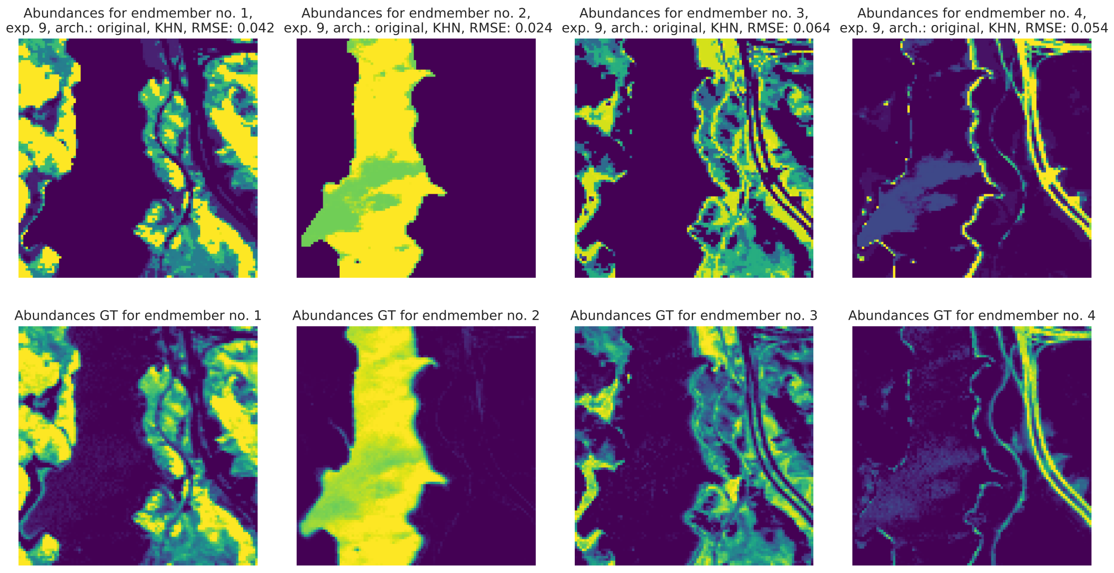


Figure 3.16: Reconstruction of fractional abundances (with ground truth) for a model with the lowest value of abundance error in terms of RMSE metric for Experiment 9 with the Jasper Ridge dataset. The subsequent plots represent abundance coefficients for tree, water, soil and road.

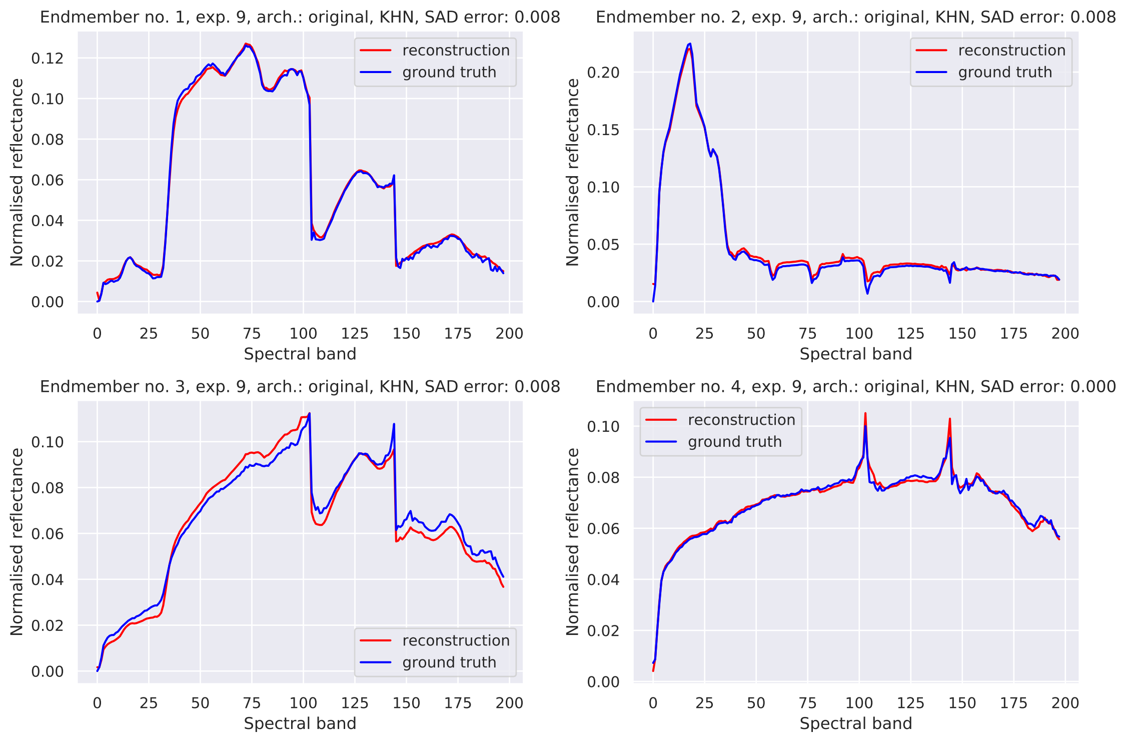


Figure 3.17: Reconstruction of endmembers (with ground truth) for a model with the lowest value of abundance error in terms of RMSE for Experiment 9 with the Jasper Ridge dataset. Plots depict normalized reflectance for consecutive spectral bands, i.e. each vector was divided by the value of its Euclidean norm. The subsequent endmembers represent tree, water, soil and road.

Chapter 4

Network reinitialization methods for improving the autoencoders' performance

4.1 Introduction

In this chapter, we try to find a solution of problems with selected models trained in hyperspectral unmixing experiments described in Chapter 3. We propose network reinitialization methods intended to minimize the risk of a failed training session. In Chapter 3 we empirically proved that weight initialization in autoencoders has a significant influence on network training, which can result in performance degradation in a pessimistic case. For further investigation, we selected a subset of experiments from Chapter 3 to make their detailed analysis. Despite the fact that in a given scenario, models were trained with the same hyperparameters set, a part of training runs was underperformed based on others. We identified that for some models the gradient has vanished during the first iterations of the training session. This was due to the dead activations' and dead neurons' phenomena which are related to the ReLU activation function [97, 131]. In such cases, the output of some neurons is zero for some or all of the input data. If it occurs for many neurons and many (or all) input values, the network learning process may be inhibited or even stopped. As a consequence, the output of the network can degenerate. The results of the analysis of the chosen autoencoder models prompted us to propose several network reinitialization methods designed for architectures with ReLU activation function. The presented methods help to reduce the negative effect of the dead activations' phenomenon.

This chapter contains and extends the work presented in the paper *Improving Autoencoder Training Performance for Hyperspectral Unmixing with Network*

Reinitialisation [79] published in the proceedings of the 21st International Conference on Image Analysis and Processing – ICIAP 2022 in Lecce. The following topics are covered in the chapter:

1. We present definitions related to dead activations and dead neurons in networks with the ReLU activation function. To measure the scale of this phenomenon in a given neural network model, we introduce dead activations' coefficients.

2. We investigate a relationship between the number of dead activations and the reconstruction error for a subset of experiments from Chapter 3.

3. Based on conclusions from the previous point, we introduce three network reinitialization techniques which are presented in the Algorithm 1. They work by replacing all or some weights of a given model. Their purpose is to alleviate the negative impact of dead activations on network performance.

4. The proposed network reinitialization methods are evaluated in three experiment scenarios for hyperspectral unmixing, using different datasets and loss functions. We perform a comparison between the baseline from Chapter 3 and results obtained from the proposed techniques.

5. Finally, we test our approaches using deeper and wider autoencoders with different hyperparameter values on the well-known MNIST dataset which generalizes the topic beyond the scope of hyperspectral unmixing.

4.2 Definitions

Let $\mathfrak{U}$ be a neural network of n hidden layers for which the number of neurons in consecutive layers is defined by a vector $\mathbf{u} = [u_0, u_1, u_2, ..., u_n, u_{n+1}]$. For each $j \in \{1, 2, ..., n+1\}$, u_j is the number of neurons in the j–th layer of the network $\mathfrak{U}$ while u_0 represents the input size. Let us assume that $\mathbf{P_j}$ is a matrix of network parameters (weights of neurons), while $\mathbf{b_j}$ is a bias vector of the j–th network layer where $j \geq 1$, i.e. $\mathbf{P_j} \in \mathbb{R}^{u_{j-1} \times u_j}$, $\mathbf{b_j} \in \mathbb{R}^{u_j}$.

Let $\mathbf{g_j}(\mathbf{x})$ be a vector of values of the ReLU activation function for the j–th layer, $j \geq 1$. Its values are calculated as follows:

$$\mathbf{g_j}(\mathbf{x}) = \max\{\mathbf{x} \cdot \mathbf{P_j} + \mathbf{b_j}, \mathbf{0}\}, \tag{4.1}$$

where $\mathbf{x} \in \mathbb{R}^{u_0}$ for $j = 1$ and $\mathbf{x} = \mathbf{g_{j-1}} \circ \mathbf{g_{j-2}} \circ ... \circ \mathbf{g_1}(\mathbf{x})$ for $j \geq 2$. We call the consecutive values of the vector $\mathbf{g_j}(\mathbf{x}) = [g_{j,1}, g_{j,2}, ..., g_{j,u_j}]$ as neurons values of the j-th layer of $\mathfrak{U}$. The maximum in Eq. 4.1 is calculated element-wise. Let us also define a matrix of neurons activations from the whole network for an input $\mathbf{x}$, that is $\mathbf{G}(\mathbf{x}) = [\mathbf{g_0}, \mathbf{g_1}(\mathbf{x}), \mathbf{g_2}(\mathbf{x}), ..., \mathbf{g_{n+1}}(\mathbf{x})]$, where for each $i \in \{0, 1, 2, ..., n+1\}$, $\mathbf{g_i} \in \mathbb{R}^{u_i}$. For the clarity of the description, we assume that $\mathbf{g_0} = \mathbf{x}$ because in the case of input values no activation function is applied. We also denote by $g_{j,i}$ the i–th element of the vector $\mathbf{g_j}(\mathbf{x})$, that is, the activation value of the i–th neuron of the j–th layer of $\mathfrak{U}$ for $\mathbf{x}$ as an input vector, where $j \in \{1, ..., n+1\}$ and $i \in \{1, ..., u_j\}$.

Definition 4.2.1. *The i–th neuron of the j–th layer is called 'dead for given input data' $\mathbf{x} \in \mathfrak{D}$, $\mathbf{x} \neq \mathbf{0}$, if for $\mathbf{x}$ as input to the network $g_{j,i} = 0$.*

We call such an inactive neuron for a given input data as dead activation.

Definition 4.2.2. *[11] Let $\mathbf{x} = [x_1, x_2, ..., x_m]$ and $\mathbf{y} = [y_1, y_2, ..., y_m]$ be nonzero vectors in $\mathbb{R}^m$, that is $\mathbf{x} \neq \mathbf{0}$ and $\mathbf{y} \neq \mathbf{0}$. The angle θ between $\mathbf{x}$ and $\mathbf{y}$, denoted as $\sphericalangle(\mathbf{x}, \mathbf{y})$, is expressed by the following formula:*

$$\sphericalangle(\mathbf{x}, \mathbf{y}) = \arccos\left(\frac{\mathbf{x} \bullet \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|}\right), \tag{4.2}$$

where $\mathbf{x} \bullet \mathbf{y}$ is a scalar product between $\mathbf{x}$ and $\mathbf{y}$.

Proposition 4.2.3. *Let the bias vector of the j–th network layer be the zero vector, that is $\mathbf{b_j} = \mathbf{0}$. Assume also that the weights matrix of the j–th layer, $\mathbf{P_j}$, is nonzero, that is $\mathbf{P_j} \neq \mathbf{0}$, as well as the activation vector $\mathbf{g_{j-1}}(\mathbf{x})$ is nonzero, $\mathbf{g_{j-1}}(\mathbf{x}) \neq \mathbf{0}$. Then, the i–th neuron of the j–th layer is dead for a given input data $\mathbf{x} \in \mathfrak{D}, \mathbf{x} \neq \mathbf{0}$, if*

$$\sphericalangle(\mathbf{g_{j-1}}, \mathbf{P_j}^\top[\cdot, \mathbf{i}]) \geq \frac{\pi}{2}, \tag{4.3}$$

where

$$\mathbf{g_{j-1}}(\mathbf{x}) = [g_{j-1,1}, g_{j-1,2}, ..., g_{j-1,u_{j-1}}],$$

$$\mathbf{P_j} = \begin{bmatrix} p^j_{1,1} & p^j_{1,2} & \cdots & p^j_{1,u_j} \\ p^j_{2,1} & p^j_{2,2} & \cdots & p^j_{2,u_j} \\ \vdots & \vdots & \ddots & \vdots \\ p^j_{u_{j-1},1} & p^j_{u_{j-1},2} & \cdots & p^j_{u_{j-1},u_j} \end{bmatrix}$$

and

$$\mathbf{P_j}[\cdot, \mathbf{i}] = [p^j_{1,i}, p^j_{2,i}, ..., p^j_{u_{j-1},i}]^\top.$$

$\mathbf{P_j}[\cdot, \mathbf{i}]$ represents the weights (parameters) between all neurons in the $(j-1)$–th layer and the i–th neuron of the j–th layer.

Proof. By the definition of the ReLU activation function, if the argument of this function is less than or equal to zero, the result will be zero. It means that such a neuron is dead for given input data. It follows that if this neuron is alive, the value of $\mathbf{g_{j-1}}(\mathbf{x}) \cdot \mathbf{P_j^\top}[\cdot, \mathbf{i}]$ has to be positive. Using the definition of the dot product:

$$\mathbf{g_{j-1}}(\mathbf{x}) \cdot \mathbf{P_j^\top}[\cdot, \mathbf{i}] = \|\mathbf{g_{j-1}}(\mathbf{x})\| \cdot \|\mathbf{P_j^\top}[\cdot, \mathbf{i}]\| \cdot \cos \sphericalangle(\mathbf{g_{j-1}}(\mathbf{x}), \mathbf{P_j^\top}[\cdot, \mathbf{i}]).$$

By the definition of the norm and the assumptions of this proposition, the norms of the vectors $\mathbf{g_{j-1}}(\mathbf{x})$ and $\mathbf{P_j^\top}[\cdot, \mathbf{i}]$ are positive numbers. The angle between these two vectors is a number from the interval $[0, \pi)$. The value of $\cos \sphericalangle(\mathbf{g_{j-1}}(\mathbf{x}), \mathbf{P_j^\top}[\cdot, \mathbf{i}])$ within this range is less than or equal to zero only when $\sphericalangle(\mathbf{g_{j-1}}(\mathbf{x}), \mathbf{P_j^\top}[\cdot, \mathbf{i}]) \geq \frac{\pi}{2}$ which ends the proof. $\square$

Definition 4.2.4. *The i–th neuron of the j–th layer is called 'dead' if for each* $\mathbf{x} \in \mathfrak{D}$ *as input of the network,* $\mathbf{x} \neq \mathbf{0}$*,* $g_{j,i} = 0$*.*

Definition 4.2.5. *The input vector* $\mathbf{x} \in \mathfrak{D}$ *is called dead in layer j, where* $j \in \{1, 2, ..., n+1\}$*, if* $\mathbf{g_j}(\mathbf{x}) = \mathbf{0}$*. It also means that if $j < n+1$, then* $\mathbf{x}$ *is dead for all subsequent layers, that is, for the $(j+1)$–th, the $(j+2)$–th, ..., and the $(n+1)$–th layer.*

Definition 4.2.6. *The network parameters of the i–th neuron of the j–th layer,* $\mathbf{P_j}[\cdot, \mathbf{i}]$*, where $j \in \{1, 2, ..., n+1\}$ and $i \in \{1, 2, ..., u_i\}$, are within a functionally dead zone if for any input vector* $\mathbf{x} \in \mathfrak{D}$*, $g_{j,i} = 0$.*

From Definition 4.2.6 it follows that if network parameters lie inside a functionally dead zone, their product with previous neurons' values will always lead to dead activations in the next layer. Furthermore, if the angle between all the neurons' output vectors of the previous layer and the parameters of the subsequent layer is less than $\dfrac{\pi}{2}$, all the neurons of the next layer will be alive, according to the Proposition 4.2.3. In intermediate states, some activations are dead while the remaining ones are alive.

Proposition 4.2.7. *For the j–th layer of the network $\mathfrak{U}$, $j \geq 1$, the maximum area of the functionally dead zone is $2^{-u_{j-1}}$ where u_{j-1} is the number of neurons in the $(j-1)$–th layer of $\mathfrak{U}$.*

Because we require that each coordinate be within the range $[0, 1]$, if a given dataset $\mathfrak{D}$ has no vectors lying on the boundaries of the domain (for example $\mathbf{x} = [1, 1, ..., 1]$), the area of the functionally dead zone is smaller. Therefore, Proposition 4.2.7 expresses the upper bound of the zone.

Definition 4.2.8. *The network $\mathfrak{U}$ is called 'dead' if there exists a layer $i \in \{1, 2, ..., n+1\}$ such that all neurons in the i–th layer are dead [131].*

The dead network is not trainable because the gradient cannot be back-propagated.

We assume that $\mathfrak{D}$ is a P–element dataset of B–dimensional vectors $\mathbf{x_i} \in \mathfrak{D}$, $i \in \{1, ..., P\}$. All elements of $\mathfrak{D}$ can be written as a matrix $\boldsymbol{X} \in \mathbb{R}^{P \times B}$. Let $\mathfrak{G}$ be an array of network activations for P input vectors, that is $\mathfrak{G} = [\mathbf{G}(\mathbf{x_1}), \mathbf{G}(\mathbf{x_2}), ..., \mathbf{G}(\mathbf{x_P})]$. Furthermore, we denote by $\mathfrak{G}^j$ a matrix of activations of the j–th layer for all input instances, that is $\mathfrak{G}^j = [\mathbf{g_j}(\mathbf{x_1}), ..., \mathbf{g_j}(\mathbf{x_P})]$. For all $i \in \{1, ..., P\}$, $\mathbf{g_j}(\mathbf{x_i}) \in \mathbb{R}^{u_j}$. This means that after a given training epoch, we have $u_j \cdot P$ neuron activations for the j–th layer.

Based on the above notation, one can determine some coefficients connected with the rate of dead activations for a given layer or for consecutive neurons of the layer. They will allow one to determine a 'vitality' of a given network element.

Definition 4.2.9. *A dead activations' coefficient for the j–th network layer, d^j_{dead}, is the ratio of a number of zero activations for this layer, denoted by $\mathcal{N}^j_0$, to the number of neurons' activations in a given training epoch ($u_j \cdot P$):*

$$d^j_{dead} = \frac{\mathcal{N}^j_0}{u_j \cdot P} \in [0, 1]. \tag{4.4}$$

$\mathcal{N}^j_0$ is counted on all neurons in the j–th network layer in a given training epoch.

Definition 4.2.10. *A dead activations' coefficient for the i–th neuron of the j–th network layer, $d^{j,i}_{dead}$, where $i \in \{1, ..., u_j\}$, is the ratio of a number of zero activations of this neuron to the number of training points in a given training epoch:*

$$d^{j,i}_{dead} = \frac{\mathcal{N}^{j,i}_0}{P} \in [0, 1]. \tag{4.5}$$

In addition, assume that all the vectors in the dataset $\mathfrak{D}$ have values of their coordinates within the range $[0, 1]$. It means that

$$\forall p \in \{1, 2, ..., P\} \quad \forall b \in \{1, 2, ..., B\} \quad x_{p,b} \in [0, 1], \tag{4.6}$$

where $x_{p,b}$ is the b–th coordinate of the vector $\mathbf{x}_p \in \mathfrak{D}$.

4.3 Related works

In recent years, some theorems, propositions and algorithms have been proposed on the topic of dying ReLU networks, i.e. dying networks with the ReLU activation function. The Authors in [131] designate lower and upper bounds of the probability of the network death, i.e. mapping all input data points to a constant function.

Furthermore, they propose a method called sign-flipping which aims to prevent the network death. It is based on the calculation of the number of dead points layer by layer. However, the authors do not perform an analysis of how this approach affects the classification accuracy or the regression error. In [97] the Authors conduct a theoretical analysis of dying ReLU networks; they prove that the probability of the ReLU network death increases as its depth goes to infinity. They also present other theoretical properties; e.g. they show why deep and narrow networks are not efficient in practice. Furthermore, they indicate obstacles related to the training of very deep networks. The Authors propose a randomized asymmetric initialization of neural network weights which aims to minimize the risk of network death. They perform basic experiments comparing their idea with the well-known He initialization approach [60].

Some Authors suggest network reinitialization methods to improve network training, but in many cases, existing solutions are designed for convolutional neural networks or they are connected with a transfer learning approach. The Authors in [58] demonstrate DSD, i.e. a dense-sparse-dense training pipeline. It consists of three parts: during the first phase, a network is trained as usual. In the second step, pruning is performed, which removes unimportant model weights. Then, a training of a sparse network is conducted. During the final phase, all previously removed connections are restored and initialized to zero. Furthermore, a dense network is fine-tuned with a smaller learning rate. The performance of this approach is verified on different datasets and CNN, LSTM and RNN networks. The authors in [9] propose a reinitialization method for convolutional neural networks. They rescale convolutional blocks and add a normalization layer. They study the performance of this approach on 12 image classification datasets and claim that the presented approach gives an advantage for small datasets. In [89] the Authors present a method called RIFLE designed for fine-tuning CNNs in a transfer learning scenario. They propose a cyclic reinitialization of fully-connected layers. Furthermore, in [171] a method based on ensemble learning and multiple reinitialization of the last layer of consecutive networks is presented. A simple approach based on weight reinitialization of randomly selected neurons is described in [152]. The probability of choice a given neuron is independent of the other neurons.

The current works on dying ReLU neurons are, in our opinion, incomplete. We know that the dead neurons phenomenon may lead to weak performance or even network death. Nevertheless, the relationships between dead activations, neurons, weight reinitialization and their impact on network performance are not clear. It is not obvious how many dead activations or neurons significantly affect the classification accuracy or the regression error. An in-depth understanding of the dying ReLU networks will make it easier to gather a solution aimed at solving this problem. Later, in this chapter, we present the analysis of the occurrence of dead activations and dead neurons in the example of selected network models described in Chapter 3 and its impact on network performance. Moreover, we propose network reinitialization methods to mitigate the negative effects of dead activations.

4.4 Network activations analysis

4.4.1 Rank correlation test

In further analysis we will use a Spearman's rank correlation coefficient which is described in more detail in the Appendix. It can be applied to verify a null hypothesis H_0 claiming that random variables $\mathcal{X}$ and $\mathcal{Y}$ are independent against an alternative hypothesis H_1 which states that there is a positive or negative dependence between $\mathcal{X}$ and $\mathcal{Y}$. In our case, we will check whether the occurrences of dead activations or dead input points before or after the training and values of the final reconstruction error are independent (a null hypothesis H_0) or if there exists a relationship between them (an alternative hypothesis H_1).

4.4.2 Correlational study

In Section 3.7.1 we performed a detailed analysis of selected autoencoder models from Experiment 4 described in Section 3.5.4. We focused on models which achieved the highest and the lowest values of the reconstruction error, claiming that we should observe key differences leading to understanding problems in low-performance models. The crucial observation was related to the vanishing gradient in low-performing models from the first training iterations. It was briefly described in Section 3.7.1. Furthermore, we found out that a subset of neurons is constantly not active, i.e. always returns zero values. This discovery prompted us to look at the magnitude of this phenomenon and investigate the impact of dead neurons' on network performance what we analyze in this chapter. It turns out that in neural networks with the ReLU activation function the number of dead neurons can be so large that the network will stop working properly, i.e. the forward and backward propagation will be disturbed. In extreme cases, a whole network layer can be dead, and, consequently, all subsequent layers will also be dead. It leads to the degenerated models, i.e. returning a constant value for all data samples. Similar topics were also recently raised in some papers, e.g. in [97] and [131]. The Authors derived there the estimation of the probability of the network death dependent on the number of layers and the number of neurons in consecutive layers. However, they did not study deeply the impact of dead neurons on network performance, i.e. they did not consider a relationship between the number of dead neurons and classification accuracy or regression error. We derive such a study on the basis of autoencoders for hyperspectral unmixing.

Based on the definitions from Section 4.2 we proposed experiments which were supposed to investigate a relationship between the number of dead activations (or dead points) and a reconstruction error of the autoencoder network, according to Definitions 4.2.1 and 4.2.5. We wanted to verify whether any of the following hypotheses is true:

1. The number of dead activations or dead input points just after initialization of all network weights (before training) is related to the reconstruction error at the end of the training.

2. The number of dead activations or dead input points at the end of the training is related to the final reconstruction error.

If any of these hypotheses are confirmed, the way of decreasing the network reconstruction error should be more clear because we would know which coefficient informs the potential of a current training run. Therefore, we could prepare a method which modifies the network state in a way that the value of such a coefficient will be more advantageous and, in consequence, the final reconstruction error will be minimized. Calculations to verify the above hypotheses were made for three experiments on *basic* architecture: Experiment 4 (Samson, MSE loss), Experiment 5 (Samson, SAD loss) and Experiment 10 (Jasper Ridge, MSE loss); see Table 3.1. We selected these experiments because in the *basic* architecture the ReLU activation function was used. For each model tested, all elements of the dataset were propagated forward through the network to obtain the output values. During this process, the number of dead activations and the number of dead input points were calculated for each of the two encoder layers.

By model, we understand a set of network weights. First, all parameters were randomly initialized and in the next step, during the training process, weights were optimized to minimize a loss function. In experiments described in Section 3.5.4 we randomly generated 50 models and each model was separately trained 50 times for each scenario and weight initialization approach.

To check the first hypothesis, for each of the 50 initialized models in a given scenario:

I. The number of dead activations and the number of dead input points before the start of the training were calculated.

II. The reconstruction error was averaged over 50 training runs. More specifically, the mean RMSE for the i–th model ($\overline{RMSE_i}$), where $i \in \{1, 2, ..., 50\}$, was calculated in the following way:

$$\overline{RMSE}_i = \sum_{j=1}^{50} RMSE_{i,j}(\mathbf{X}, \tilde{\mathbf{X}}).$$

$\mathbf{X}$ represents the input vectors, $\tilde{\mathbf{X}}$ their reconstructions, while $RMSE_{i,j}(\cdot,\cdot)$ denotes the mean reconstruction error for the j–th training run of the i–th model, calculated at the end of the training session of the j–th run. After the calculations were completed, there were 50 mean RMSE values for consecutive models.

III. The Spearman's rank correlation coefficient between 50 pairs of numbers (separately per each experiment setup) was designated in two cases, i.e. between the number of dead activations just after initialization and the mean final reconstruction error as well as between the number of dead points before the start of the training and the mean final reconstruction error.

To verify the second hypothesis, for each of the 50 models:

I. One out of 50 training runs was selected.

II. At the end of the learning process of a given run, the numbers of dead activations and dead input points were calculated.

III. Similarly as previously, Spearman's rank correlation coefficient between the collected numbers and the final reconstruction error for corresponding network runs was computed.

Table 4.1 presents correlation coefficients for models just after weights initialization, while Table 4.2 shows results for trained autoencoders, in both cases for dead activations. Furthermore, Table 4.3 contains statistics for dead input points for the same models as presented in Tables 4.1–4.2 but only in the case of the second encoder layer. The values of the Spearman's rank correlation coefficient were calculated separately for each initialization method (KHN and KHU denote the Kaiming He technique [60] with normal or uniform distribution, respectively, while XGN and XGU mean the Xavier Glorot approach [50], also with normal or uniform distribution). An analysis of the results presented in Tables 4.1–4.3 leads to the conclusion that for Experiments 4 and 10 there is a strong or very strong correlation between the number of dead activations in the second encoder layer at the end of the training process and the network reconstruction error. This relationship is much more visible in the case of trained models than for models just after initialization. It means that neurons die during training and this phenomenon has an important impact on the final network score. This dependency is not observable for Experiment 5 (with SAD loss function). Furthermore, one can conclude that

Table 4.1: Results of Spearman's rank correlation coefficient (with p-value) between the number of dead activations for models after initialization and the mean final reconstruction error. Correlation values above 0.7 are shown in bold.

		first encoder layer		second encoder layer	
exp. ID	init. method	correlation	p-value	correlation	p-value
4	KHN	-0.065	0.654	0.698	1.80e-08
4	KHU	-0.008	0.958	**0.796**	5.01e-12
4	XGN	0.019	0.897	0.693	2.50e-08
4	XGU	0.362	0.010	**0.783**	1.79e-11
5	KHN	0.147	0.309	0.032	0.825
5	KHU	-0.144	0.319	-0.090	0.535
5	XGN	-0.082	0.571	0.402	0.004
5	XGU	-0.105	0.470	0.090	0.532
10	KHN	0.234	0.102	0.578	1.33e-05
10	KHU	0.083	0.567	**0.870**	2.57e-16
10	XGN	0.106	0.463	0.578	1.12e-05
10	XGU	-0.160	0.266	0.607	2.92e-06

the second encoder layer is crucial in terms of the training success because similar reasoning done for the first encoder layer leads to weak correlation values. It is probably due to the fact that the second encoder layer is a "bottleneck" because it has only 3 or 4 neurons (3 for Samson and 4 for Jasper Ridge dataset), so the death of each particular neuron can be costly. It is a known fact from the literature that the longer and narrower the network, the more likely neurons will die [97, 131]. Our networks are not so long (only a few layers), but narrow, especially in the last encoder layer. Furthermore, correlations between dead input points and reconstruction errors are not as significant as for dead activations. One can conclude that a weak performing network run may be easier detected using dead activations' statistics than the dead input points' metric. It can be surprising, but one possible reason is that the input data point may not die but be "barely alive" which has a negative impact on further reconstruction. In such a case, it will not be counted as a dead input point but still, it contributes to the number of individual dead activations.

4.5 Network reinitialization methods

Based on conclusions presented in Section 4.4 we prepared several network reinitialization methods which take into account the numbers of dead activations. When a weak performing model is identified, some or all network weights are changed, depending on the adopted strategy. The Algorithm 1 introduces a general descrip-

Algorithm 1: Network reinitialization methods.

```
Input: $n$–hidden layer neural network $\mathfrak{U}$, numbers of neurons in
       consecutive layers: $[u_0, u_1, u_2, ..., u_n, u_{n+1}]$, a batch of input data:
       $\boldsymbol{X}$, $\boldsymbol{X} \in \mathbb{R}^{b_s \times B}$, $b_s$: a batch size, $B$: a number of dimensions of a
       single data point, $t \in (0, 1)$: threshold, $method$: a selected
       reinitialization method (whole network, single layer or
       partial reinitialization)
Output: $n$–hidden layer neural network $\mathfrak{U}$
Perform a forward propagation and calculate $\mathbf{G}(\boldsymbol{X})$;
for $i \leftarrow 1$ to $n + 1$ do
    if $method =$ whole network or single layer then
        Calculate $d^i_{dead}$ based on $\mathbf{G}(\boldsymbol{X})$;
        if $d^i_{dead} > t$ then
            if $method =$ whole network then
                Reinitialize all weights of the network $\mathfrak{U}$.
            end
            else
                Reinitialize all weights of the $i$–th layer of the network $\mathfrak{U}$.
            end
            Interrupt the calculations and return the reinitialized network
              $\mathfrak{U}$.
        end
    end
    else if $method =$ partial reinitialization then
        Calculate $d^{i,1}_{dead}, d^{i,2}_{dead}, ..., d^{i,u_i}_{dead}$ based on $\mathbf{G}(\boldsymbol{X})$;
        for $j \leftarrow 1$ to $u_i$ do
            if $d^{i,j}_{dead} > t$ then
                Reinitialize all weights connected with the $j$–th neuron of
                  the $i$–th layer of the network $\mathfrak{U}$.
            end
        end
        /* Only neurons of one layer can be reinitialized in a single
           iteration. If at least once a threshold has been exceeded,
           some weights were reinitialized, further calculations of dead
           activations' statistics are interrupted and the training is
           continued.                                                  */
        if $\max_{j \in \{1,...,u_i\}} d^{i,j}_{dead} > t$ then
            Interrupt the calculations and return the reinitialized network
              $\mathfrak{U}$.
        end
    end
end
```

Table 4.2: Results of Spearman's rank correlation coefficient (with p-value) between the number of dead activations for a selected run of trained models and the final reconstruction error. Correlation values above 0.7 are shown in bold.

		first encoder layer		second encoder layer	
exp. ID	init. method	correlation	p-value	correlation	p-value
4	KHN	-0.171	0.236	**0.893**	2.72e-18
4	KHU	-0.046	0.750	**0.859**	1.49e-15
4	XGN	-0.125	0.389	**0.847**	8.79e-15
4	XGU	0.239	0.095	**0.889**	7.33e-18
5	KHN	0.142	0.325	-0.270	0.058
5	KHU	0.019	0.897	-0.251	0.078
5	XGN	-0.211	0.141	0.174	0.226
5	XGU	0.038	0.795	0.029	0.840
10	KHN	0.049	0.736	**0.762**	1.27e-10
10	KHU	0.089	0.541	**0.887**	9.50e-18
10	XGN	0.068	0.640	**0.774**	4.38e-11
10	XGU	-0.010	0.947	**0.813**	7.46e-13

tion of the proposed methods. A reinitialization method is run during a forward propagation, after each processed batch of data. Let us suppose that we have a network $\mathfrak{U}$ where a vector $\mathbf{u} = [u_0, u_1, u_2, ..., u_n, u_{n+1}]$ represents numbers of neurons in consecutive layers. We start the network analysis from the first hidden layer, u_1 (u_0 is an input). One of the three approaches is applied:

- *Whole network reinitialization*: if there exists a layer $i \in \{1, ..., n+1\}$ such that $d^i_{dead} > t$ then all the weights of the network are drawn randomly according to the given initialization scenario (e.g. KHU, XGU, etc.). It is similar to the situation before the training of the model, when all network weights are randomly generated.

- *Single layer reinitialization*: if there exists a layer $i \in \{1, ..., n+1\}$ such that $d^i_{dead} > t$ then all weights of the i–th layer are drawn randomly according to the given initialization scenario, while other weights of the network remain unchanged.

- *Partial network reinitialization*: if there exists a layer $i \in \{1, ..., n+1\}$ and a neuron $j \in \{1, ..., u_i\}$ (or more neurons) such that $d^{i,j}_{dead} > t$ then all weights connected to the j–th neuron of the i–th layer are drawn randomly according to the given initialization scenario. Reinitialization concerns only neurons of one layer selected in a current iteration (single or more neurons, up to u_i) while other weights remain the same.

Table 4.3: Results of Spearman's rank correlation coefficient (with p-value) between the number of dead input points and the final reconstruction error for the second encoder layer in two scenarios. Correlation values above 0.7 are shown in bold.

		initialized models		trained models	
exp. ID	init. method	correlation	p-value	correlation	p-value
4	KHN	0.323	0.022	0.693	2.45e-08
4	KHU	0.593	5.69e-06	**0.794**	6.08e-12
4	XGN	0.246	0.085	0.635	7.27e-07
4	XGU	0.695	2.11e-08	**0.808**	1.30e-12
5	KHN	-0.005	0.975	-0.441	0.001
5	KHU	0.010	0.491	-0.246	0.085
5	XGN	0.383	0.006	0.300	0.035
5	XGU	0.286	0.044	0.028	0.847
10	KHN	0.291	0.040	0.339	0.016
10	KHU	0.338	0.016	0.563	2.08e-05
10	XGN	0.299	0.035	0.339	0.016
10	XGU	0.431	0.002	0.339	0.016

For each of the presented methods, calculations of dead activations' coefficients were performed in the order from the first to the last layer and possibly can be omitted for the last layer. It is due to the fact that this is often a classification or regression layer and zero activations are in many cases desired there. The final decision can be made by the network designer. If a threshold for the dead activations' coefficient is exceeded, calculations for the following layers are abandoned. It means that *single layer reinitialization* method can change weights of only one selected layer while *partial reinitialization* approach only affects one or more neurons of the selected network layer. Dead activations of the i–th layer contribute to the death of neurons in subsequent layers of the network $(i+1, i+2, ..., n+1)$. It means that preventing neurons from dying in earlier layers has a positive effect on the next layers. It is also consistent with conclusions from the paper [131] in which authors claim that a probability of the network death increases as the number of network layers is getting higher.

4.6 Reinitialization of autoencoders for hyperspectral unmixing

4.6.1 Experiments' scenario

The methods described in Section 4.5 were applied to improve the performance of autoencoders for hyperspectral unmixing. Models' reinitialization was performed for Experiments 4, 5 and 10 from Section 3.5, i.e. for all experiments in which the *basic* architecture with the ReLU activation function was used. In the case of Experiments 4 and 5, unmixing was made on the Samson dataset for the architecture with $(156, 30, 3, 156)$ neurons in consecutive layers. 30 neurons in the first hidden layer were chosen during hyperparameter optimization using Ray Tune [92], as presented in Section 3.5.4. 156 is a number of spectral bands, while 3 is a number of endmembers in the Samson dataset. For Experiment 10, the Jasper Ridge dataset and the architecture with $(198, 40, 4, 198)$ neurons in the following layers was used. Similarly as before, 198 is a number of spectral bands, 40 was chosen with Ray Tune while 4 is a number of endmembers in the dataset. In both cases, bias was only attached to two hidden layers of the networks. Furthermore, for Experiments 4 and 10, the MSE loss function was selected, while for Experiment 5 the SAD loss function was chosen. Detail information about the rest of the hyperparameters is contained in Table 3.1.

In each scenario, 200 models generated in experiments described in Section 3.5 were used. As previously, per each weights initialization scenario (KHU, KHN, XGU and XGN), 50 models were prepared. In Chapter 3, for each model, 50 separate training sessions were performed while in this case, for simplicity, one training run per model with a selected setting was carried out. Such a setup should still be sufficient to reliably assess the differences between methods.

The main difference between the previous and the current series of experiments is that for each model one training run with a selected setting was carried out (not 50 runs for each model, as in the former case).

For each of the 200 models, four independent training sessions with different scenarios were performed: using a standard training technique (denoted as baseline), *whole network reinitialization* approach, *single layer reinitialization* and *partial reinitialization*. The Algorithm 1, which presented three network reinitialization methods, was adapted to the *basic* architecture. It means that, according to the results presented in Section 4.4, calculations of dead activations coefficients' were performed only for the second encoder layer, i.e. for a layer with 3 neurons in the case of Experiments 4 and 5, or for a layer with 4 neurons for Experiment 10. It is due to the fact that such a layer is a ”bottleneck” of this architecture and Spearman's rank correlation coefficients between the number of dead activations and reconstruction errors were the highest for this layer. In all experiments a threshold t was set using a rule of thumb to 0.6 (i.e. at least 60% of dead activations is necessary for reinitialization). In the case of the Samson dataset, the threshold set implies that in some cases, all 3 neurons' activations in the second encoder layer had to be zero, while for the Jasper Ridge dead should be at least 3 from 4 activations.

4.6.2 Wilcoxon signed-rank test

We would like to verify whether a reconstruction error has decreased significantly after the application of the network reinitialization method compared to the original approach. Therefore, we selected the Wilcoxon signed-rank test (see the Appendix for more details). We could not use a parametric t-test for paired samples due to failure to meet the condition of the normality of the data, according to results of the Shapiro-Wilk test. We have a set of M pairs of the form (x_i, y_i), where $i \in \{1, 2, ..., M\}$, M is the number of models. In our case, the first coordinate of pairs represents a reconstruction error for the baseline approach, while the second coordinate is a reconstruction error for a training run starting from the same set of initial weights, with the application of one of the presented reinitialization methods. In our experiments $M = 200$. We expect that y_i's will be generally lower than x_i's, i.e. reinitialization methods improve network performance. In such a case, the null hypothesis of the Wilcoxon test should be rejected in favor of the alternative hypothesis.

4.6.3 Results

Table 4.4 presents results of weight reinitialization experiments for the Samson dataset with the reconstruction and abundance errors in terms of RMSE and endmember errors in terms of the SAD function as well. Table 4.5 shows analogous results for the Jasper Ridge dataset.

In the case of Experiment 4 using the Samson dataset and the MSE loss function, differences between model performances for Glorot and He initialization methods can be observed. In the first situation, *whole network reinitialization* technique achieved the lowest reconstruction and abundance errors while *partial reinitialization* was the most promising in terms of endmembers' spectra reproduction. Furthermore, differences between baseline (i.e. models without weight reinitialization during the training) and proposed reinitialization methods were statistically significant, according to the Wilcoxon signed-rank test with a p-value threshold of 0.05. In the He scenario, *partial reinitialization* approach was the most effective in almost all cases, except for the endmember error for KHU technique.

In this situation the improvements were also statistically significant. Looking at the average scores, the lowest error values were achieved with XGU approach.

Regarding Experiment 5, in which a loss function has been changed from MSE to SAD, *single layer* and *partial reinitialization* were the most efficient methods. They statistically outperformed the baseline scores for abundance and endmember errors. The results for these metrics seem to be more stable than for Experiment 4 but still the lowest mean scores were achieved for the MSE loss function. However, the reconstruction of individual pixels is burdened with a greater error than in the case of Experiment 4. Furthermore, for almost all setups, the differences in this case were not statistically significant. It should be mentioned that the *whole network* approach also surpassed baseline scores, but generally, it was not the best solution.

For the Jasper Ridge dataset, for which calculations were performed in Experiment 10, the lowest error values were achieved by models initialized according to XGU approach. In all cases but one, *single layer* reinitialization approach led to the smallest abundance and endmember errors. The reconstruction of individual pixels was best prepared by *partial reinitialization* method. It can be seen that the improvements compared to the baseline results were statistically significant for a lower number of cases than for experiments on the Samson dataset.

4.6.4 Discussion

A more detailed analysis of the results for the *basic* architecture presented in Chapter 3 uncovered the problem with dead activations. Its investigation by the use of dead activations' coefficients (Definitions 4.2.9–4.2.10) and a Spearman's rank correlation measure led to three methods aimed at improving training of the network models. There were proposed solutions customized to this architecture that minimize the negative impact of the dead activations' phenomenon on training performance. Furthermore, the presented methods can easily be generalized. The results contained in Tables 4.4–4.5 clearly indicate that the scores of the baseline approach had improved. This applies both for unmixing metrics (i.e. endmember and abundance errors) and the reconstruction error. In many cases, new results statistically outperformed the initial approach, in terms of the Wilcoxon signed-rank test. The p-value significance threshold was set to 0.05.

It is also interesting that despite the fact that for models from Experiment 5 (i.e. trained using the SAD function) there was no significant correlation between numbers of dead activations and the network reconstruction error in terms of RMSE, proposed reinitialization approaches still reduced values of endmember and abundance errors. It is probably due to the fact that the SAD function is scale invariant. This means that it does not necessarily minimize the reconstruction error in terms of mean or root mean squared error.

Comparing the results in Tables 3.3 and 4.4-4.5 it can be noticed that errors achieved by the *original* architecture are lower than in the case of the *basic* autoencoder, even after the application of reinitialization methods. The most favorable autoencoder's settings, by the endmember or abundance errors' point of view, were achieved for Experiments 3 and 7, for the Samson and the Jasper Ridge, respectively. The *basic* architecture applies only the most common components of neural networks, i.e. linear layers, the ReLU activation function and a normalization layer to fulfill the abundance requirements. The *original* architecture is more sophisticated than the *basic*: it has more layers in the encoder part, another activation function and additional layers like Gaussian Dropout. However, we do not always want to use complicated architectures due, e.g. the increase in computational costs. Furthermore, it is natural to start from simpler networks. The problem in the *basic* architecture has been identified and its significance has been diminished thanks to the proposed weights reinitialization methods. In the next section, the performance of these solutions will be evaluated for both other architectures and datasets, not related to the hyperspectral unmixing task.

4.7 Generalization of results beyond the scope of hyperspectral images

4.7.1 MNIST dataset

To verify the performance of network reinitialization methods beyond the scope of hyperspectral images, a MNIST handwritten digits dataset [82] was chosen. It consists of thousands of gray scale images of size 28×28 which depict digits 0, 1, 2, ..., 9. The samples were prepared by Census Bureau employees and high-school students. The dataset was divided into a training part containing 60,000 examples and a test part with 10,000 digits' samples. This collection has achieved great popularity and it is one of the basic datasets for testing the performance of new architectures and methods of deep learning, like in [46], [127] or [138]. Selected images from the MNIST dataset are presented in Figure 4.1.

4.7.2 Experiments

We carried out experiments using the MNIST dataset on several different autoencoder architectures and sets of hyperparameters with a grid search technique. It means that we tested each combination of the following hyperparameters:

1. Network architecture:

   $arch \in \{[500, 500], [100, 100], [100, 100, 100, 100], [100, 100, 100, 100, 100, 100]\}$

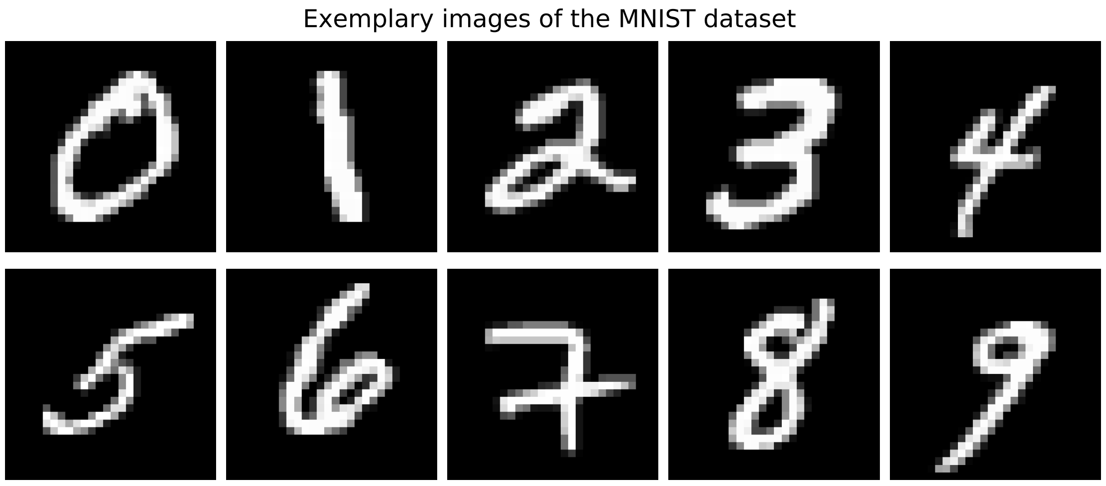


Figure 4.1: A presentation of selected images of the MNIST handwritten digits dataset.

neurons, both for the encoder and the decoder part of the network; an autoencoder had a total of $4, 4, 8$ or 12 hidden layers, respectively.

2. Learning rate:
$$l_r \in \{0.0001, 0.001, 0.01, 0.1\}.$$
3. Batch size:
$$b_s \in \{4, 16, 32, 64, 128\}.$$

For each set of hyperparameters $\{arch, l_r, b_s\}$ experiments were repeated 5 times, for different seed values. A validation set was also separated, which represented 10% of the training set. The maximum number of training epochs has been set to 50 but a training run was early stopped if a validation loss did not decrease below the minimum achieved so far through 5 consecutive epochs. As a weight initialization method, KHU (Kaiming He with uniform distribution) was selected because that it was the default approach in the PyTorch library for linear layers. We carried out experiments for several reinitialization scenarios:

- *Whole network reinitialization* with thresholds $t \in \{0.75, 0.9, 0.95\}$.
- *Single layer reinitialization* with thresholds $t \in \{0.75, 0.9, 0.95\}$.
- *Partial network reinitialization* with threshold $t = 0.99$.

The threshold values were selected as a rule of thumb. For comparison purposes, we also prepared baseline experiments, i.e. without weight reinitialization during the training. It means that for each of the 8 network training approaches we have 400 model instances.

4.7.3 Results

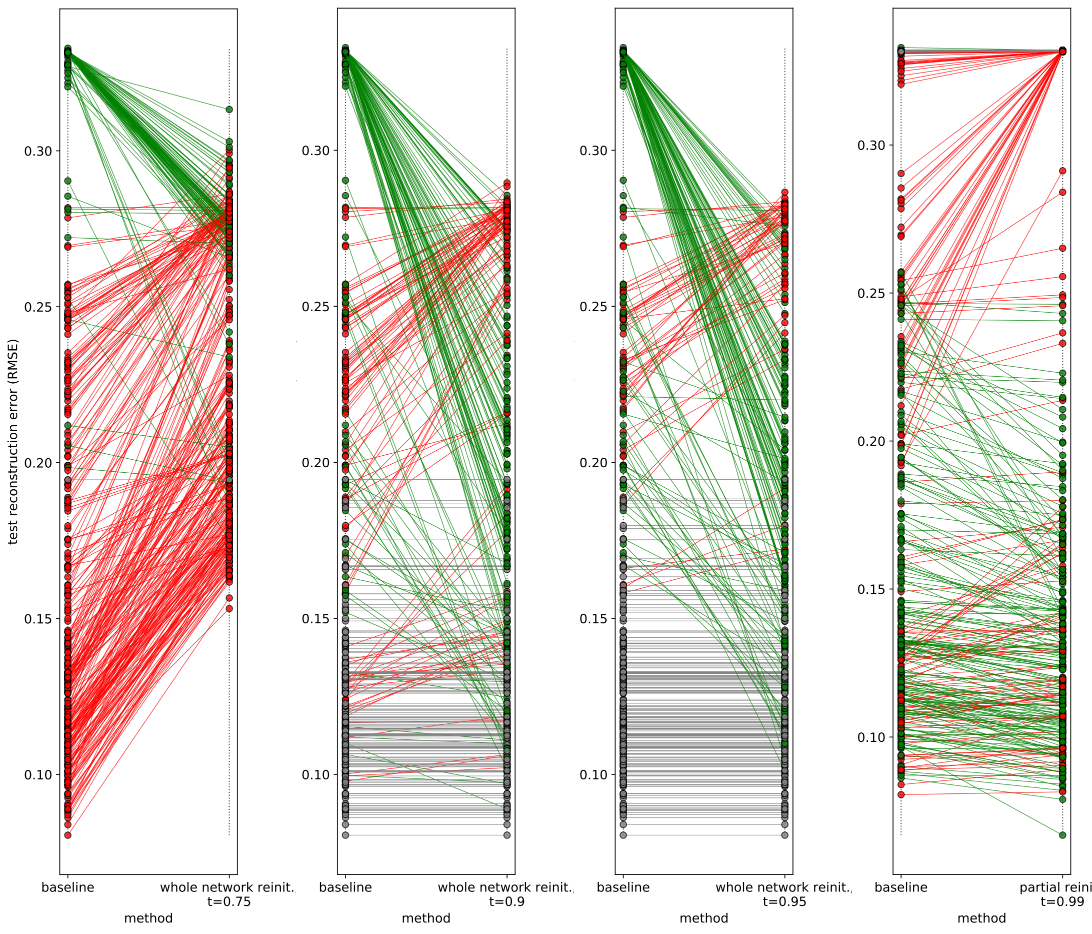


Figure 4.2: Comparison of test reconstruction errors (in terms of RMSE) between baseline and reinitialization methods for consecutive models (starting from the left side: *whole network reinit.* method with thresholds $t \in \{0.75, 0.9, 0.95\}$ and *partial reinit.* method with $t = 0.99$). Each line represents one training run. Green means that RMSE decreased after the application of reinit. method, red indicates increasing of RMSE while gray marks models whose RMSE has not changed.

Figures 4.2–4.3 depict slope charts that compare test reconstruction errors between baseline models (i.e. without reinitializations) and networks after the application of reinitialization methods. Results are presented for all training runs.

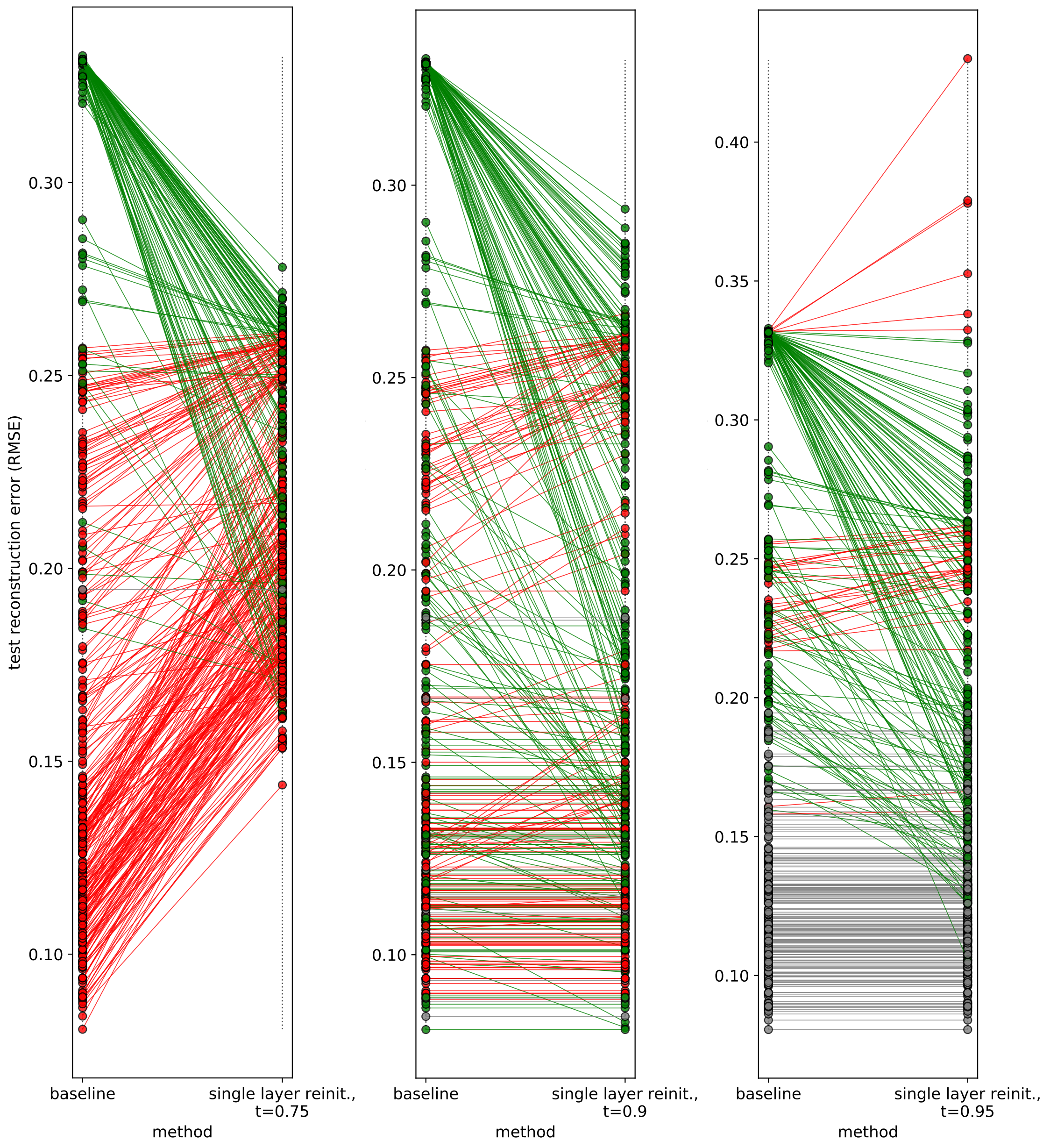


Figure 4.3: Comparison of test reconstruction errors (in terms of RMSE) between baseline and *single layer reinitialization* method for consecutive models and thresholds $t \in \{0.75, 0.9, 0.95\}$. Each line represents one training run. Green means that RMSE decreased after the application of reinit. method, red indicates increasing of RMSE while gray marks models whose RMSE has not changed.

Green denotes the cases where the test reconstruction error has decreased after using a reinitialization approach. The red color means the opposite case, while gray color is related to the models whose error has not changed. It can be seen that for *whole network* and *single layer reinitialization* approach with threshold $t = 0.75$ the results form two groups. Models that initially were low-performing have improved, while models that originally had smaller error values have deteriorated. The situation changes with the increase of the dead activations' threshold value. Models that initially had small RMSE slightly changed or remained the same, while in most cases low-performing models have improved. For both *whole network* method and for *single layer reinitialization* technique the number of deteriorated models for $t = 0.95$ decreased compared to $t = 0.9$ and $t = 0.75$. The results for *partial reinitialization* method are slightly different than for other approaches. The weakest models continued to deteriorate, while in most cases mid-range and high-performing models have improved. Furthermore, the best individual result was achieved with this approach.

We also present results for the best 3 out of the 80 trained models for each of the network reinitialization methods and we compare them with baseline. Model selection was carried out based on RMSE on the validation set. Results for such models are presented in Table 4.6 with corresponding network hyperparameters. We can observe that the three best individual results have been achieved by *partial reinitialization* method. The next few high-performing models included in Table 4.6 did not need weight reinitializations, but they were not harmed by applied techniques. The weakest results have been achieved by *whole network* and *single layer* methods with threshold $t = 0.75$. In these cases, the test reconstruction errors were almost 2 times higher than in the case of the best models (0.085 for the best individual) and they exceeded 0.155. Also, the slope charts depicted in Figures 4.2–4.3 indicate that high-performance models in the baseline approach were spoiled by reinitialization methods with the lowest threshold. This observation confirms that $t = 0.75$ is not a good option for calculations on this dataset. However, to fully understand their potential, it is worth performing a more in-depth analysis of the results.

4.7.4 Discussion

Due to the fact that experiments were prepared for different sets of network hyperparameters, we can analyze their impact on network performance, i.e. the reconstruction error. Figure 4.5 depicts results of network reconstruction in terms of RMSE depending on the values of learning rate, batch size and type of architectures, i.e. number of hidden layers and number of neurons in consecutive layers. The influence of the learning rate on the final results appears to be significant. Models with the highest reconstruction error had a value of 0.01 or 0.1 while well-performing models had lower learning rate's values like 0.0001 or 0.001. This phenomenon occurred despite different architectures and batch sizes. The

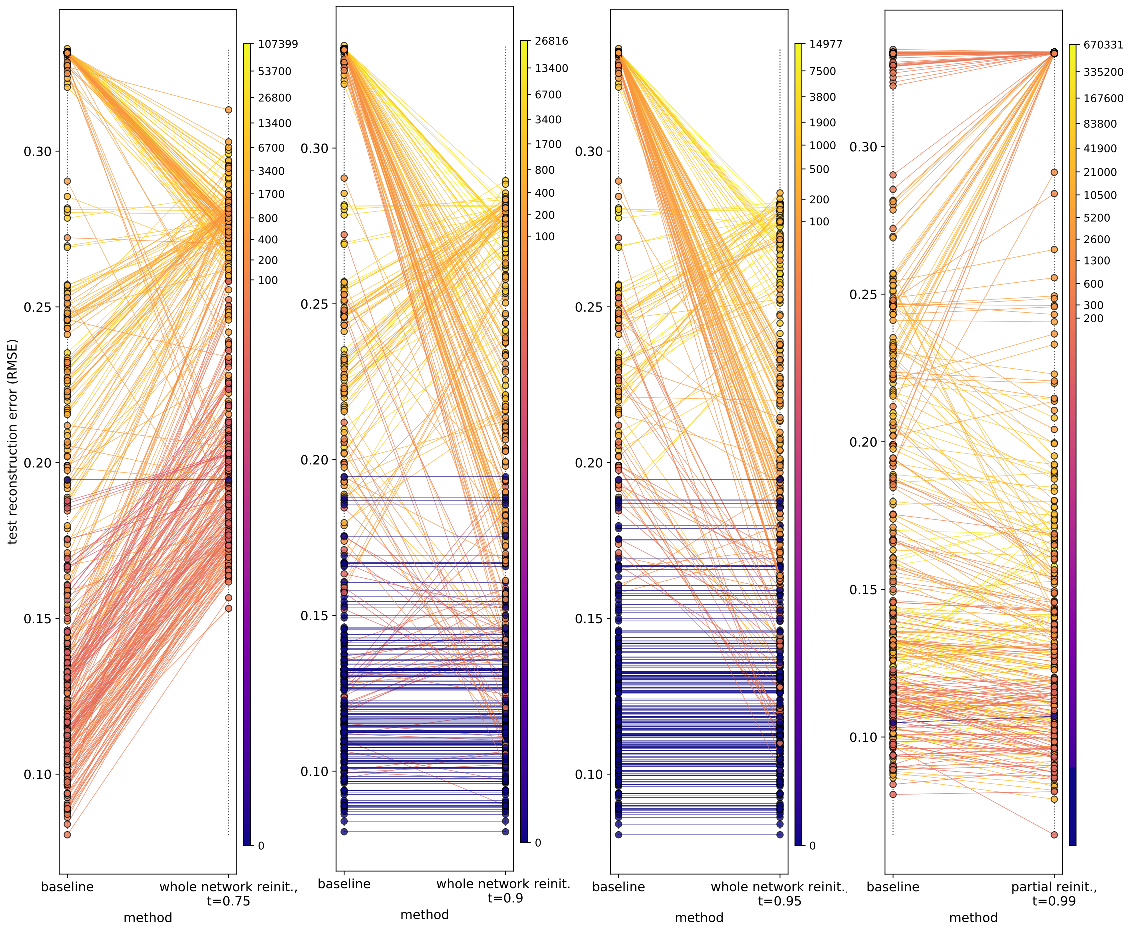


Figure 4.4: Numbers of iterations with the reinitialization of at least one network weight for consecutive models, according to the selected reinitialization method. Plots depict results for, starting from the left side, *whole network reinit.* method with thresholds $t \in \{0.75, 0.9, 0.95\}$ and *partial reinit.* approach with $t = 0.99$. Each line represents one training run while points correspond to test reconstruction errors before and after the application of reinitialization methods. The color scale was transformed using a power-law normalization. The darker the color, the fewer reinitializations were performed.

impact of these two hyperparameters seems to be not as high as in the case of learning rate, although the most promising results were achieved by networks with two hidden layers of 500 neurons on each. Similarly, models with greater batch sizes (like 64 or 128) achieved lower reconstruction errors than models with smaller values of this hyperparameter, i.e. 4 or 16. Overall, the learning rate seems to have the greatest influence on network performance.

In Figures 4.4–4.6 there is presented a dependency between the numbers of

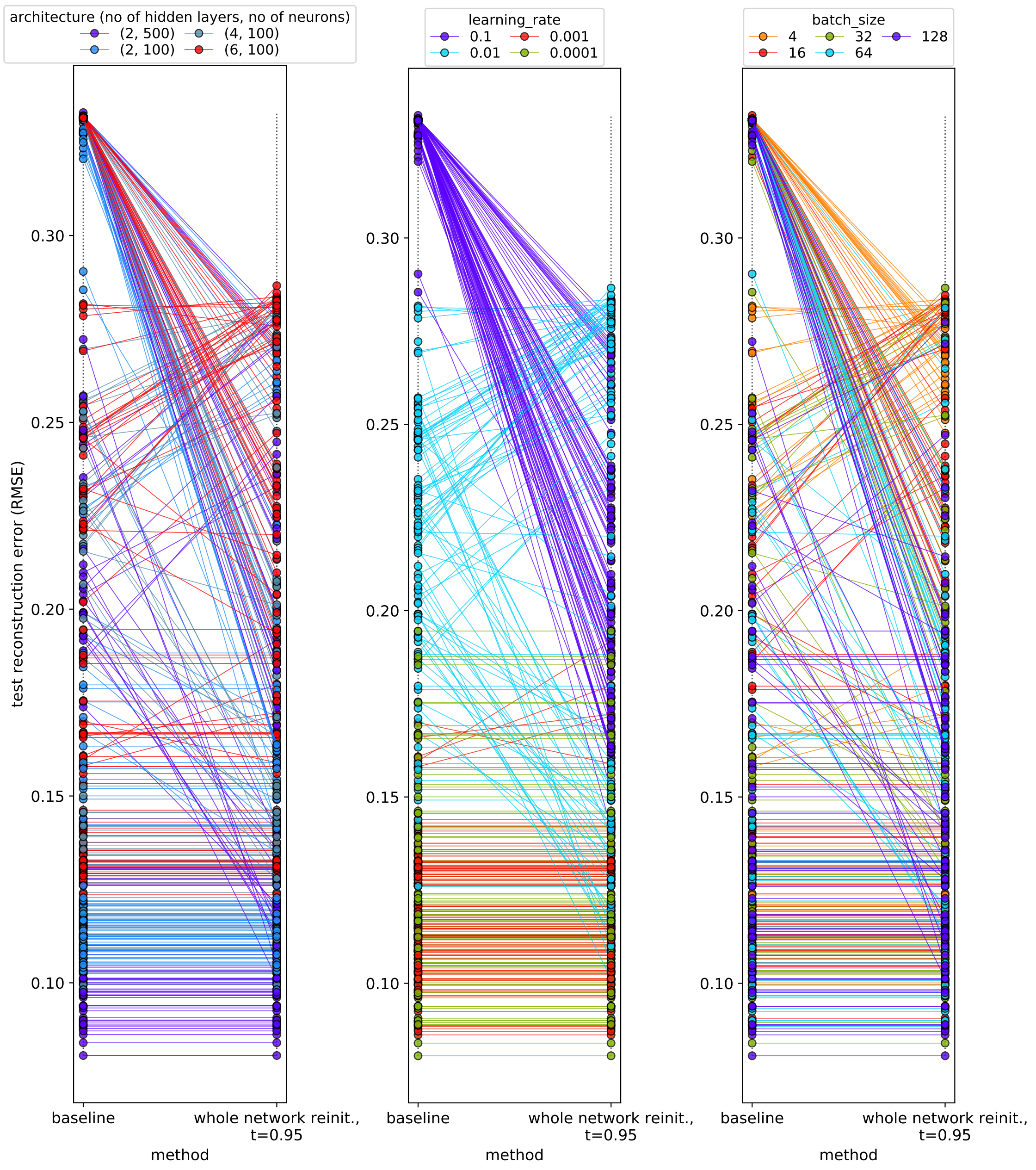


Figure 4.5: The impact of different network hyperparameters on the test reconstruction error for *whole network reinitialization* method with threshold $t = 0.95$. Each line represents one training run while points correspond to test reconstruction errors before and after the application of reinitialization methods. Plots depict results for, starting from the left side, different network architectures, learning rates and batch sizes.

network reinitializations and reconstruction errors on the test set. Based on the above plots, it can be deduced that, generally, the number of iterations in which at least one weight has been changed, is the highest in the case of low-performing models. For efficient models, weight reinitializations were not as frequent as for models with the greatest values of the reconstruction error, or they even did not occur. It means that reinitialization methods are not harmful for good network models, while they can improve weaker model instances. The number of iterations with weight modifications is greater for *partial reinit.* technique due to the fact that the reinitialization may be performed for a smaller number of neurons (in an extreme case only for weights of one neuron), but even such a small change increments the variable counting weight changes.

Furthermore, Table 4.7 presents results of Spearman's rank correlation coefficient between test reconstruction errors from baseline experiments and numbers of network reinitializations using different methods and threshold values. It is possible to conclude that for 5 out of 7 approaches, there is a strong correlation between tested variables and it exceeds 0.7. For *single layer reinit.* with a threshold $t = 0.75$ the correlation coefficient is only slightly below 0.7 and it is equal to 0.672 while for *partial reinit.* technique, the result is not significant. Probably it is due to the fact that for this method the way of performing reinitializations is different than in the other cases. We would obtain additional information by calculating the correlation using the number of reinitialized neurons instead of the number of iterations with modification of the weights of at least one neuron. On the basis of presented results, we can conclude that the values of Spearman's rank correlation coefficient are getting higher as the threshold value t increases. The presented results suggest that reinitialization methods can be considered as a *watchdog*, indicating that a given set of hyperparameters is far from optimal and that a given training session can be earlier stopped. Indeed, a wrong selection of hyperparameters leads to high reconstruction errors, so a relationship between the number of reinitializations and the quality of reconstruction opens the direction for future research. This property may be used for grid or random search algorithms for the interruption of badly-performing training sessions to save calculation time.

Computational times

Figure 4.7 presents histograms of model training times for various learning scenarios, including different reinitialization approaches and the baseline approach. Furthermore, the kernel density estimation plot of model training times for all the considered methods is depicted in Figure 4.8. It was prepared to compare model training times for different reinitialization methods in a more convenient way. Due to the fact that calculations were performed for various hyperparameters and architectures, and since the final number of training epochs was dependent on the

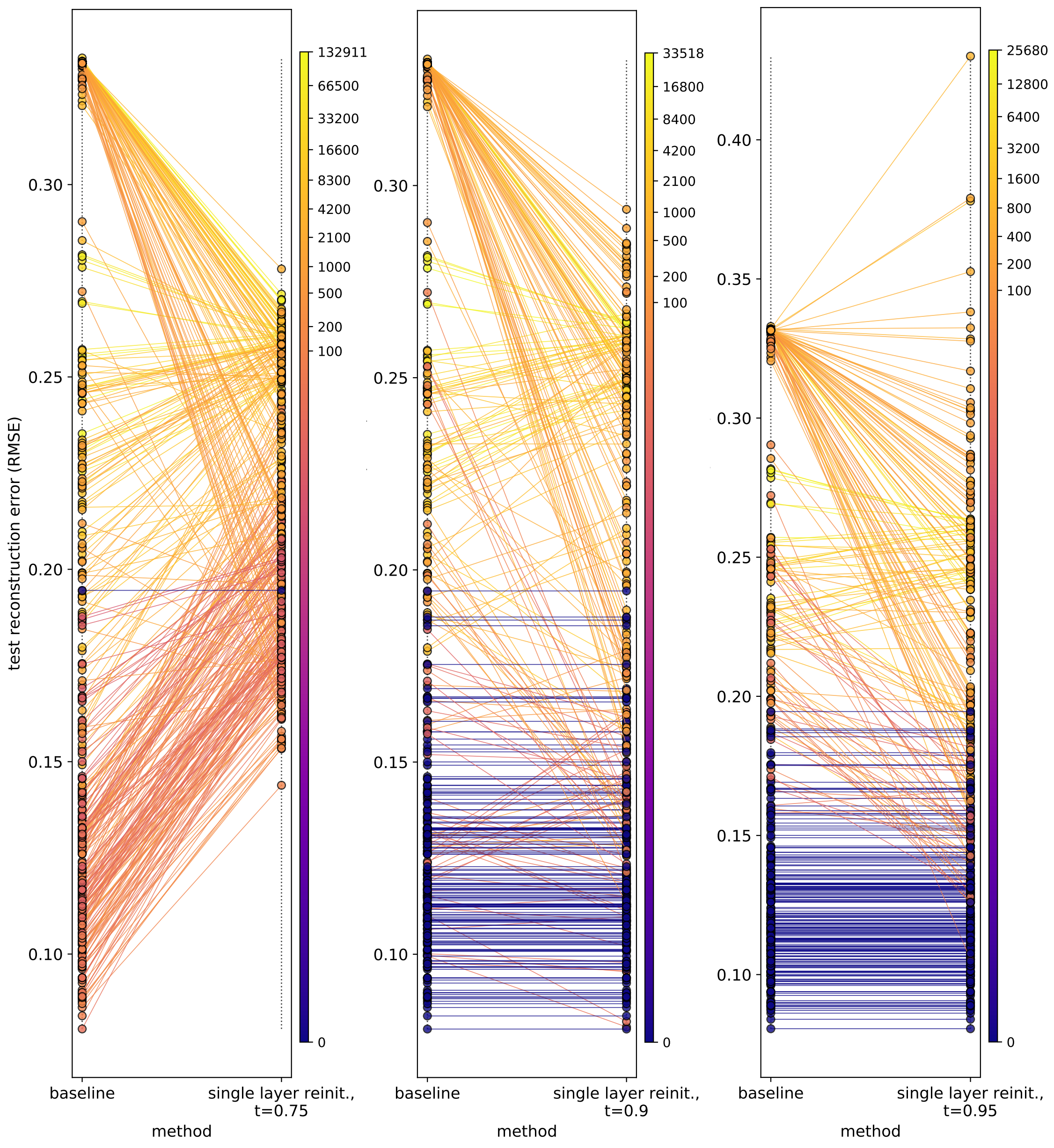


Figure 4.6: Numbers of iterations with the reinitialization of at least one network weight for consecutive models and *single layer reinitialization* method with thresholds $t \in \{0.75, 0.9, 0.95\}$. Each line represents one training run while points correspond to test reconstruction errors before and after the application of reinitialization methods.

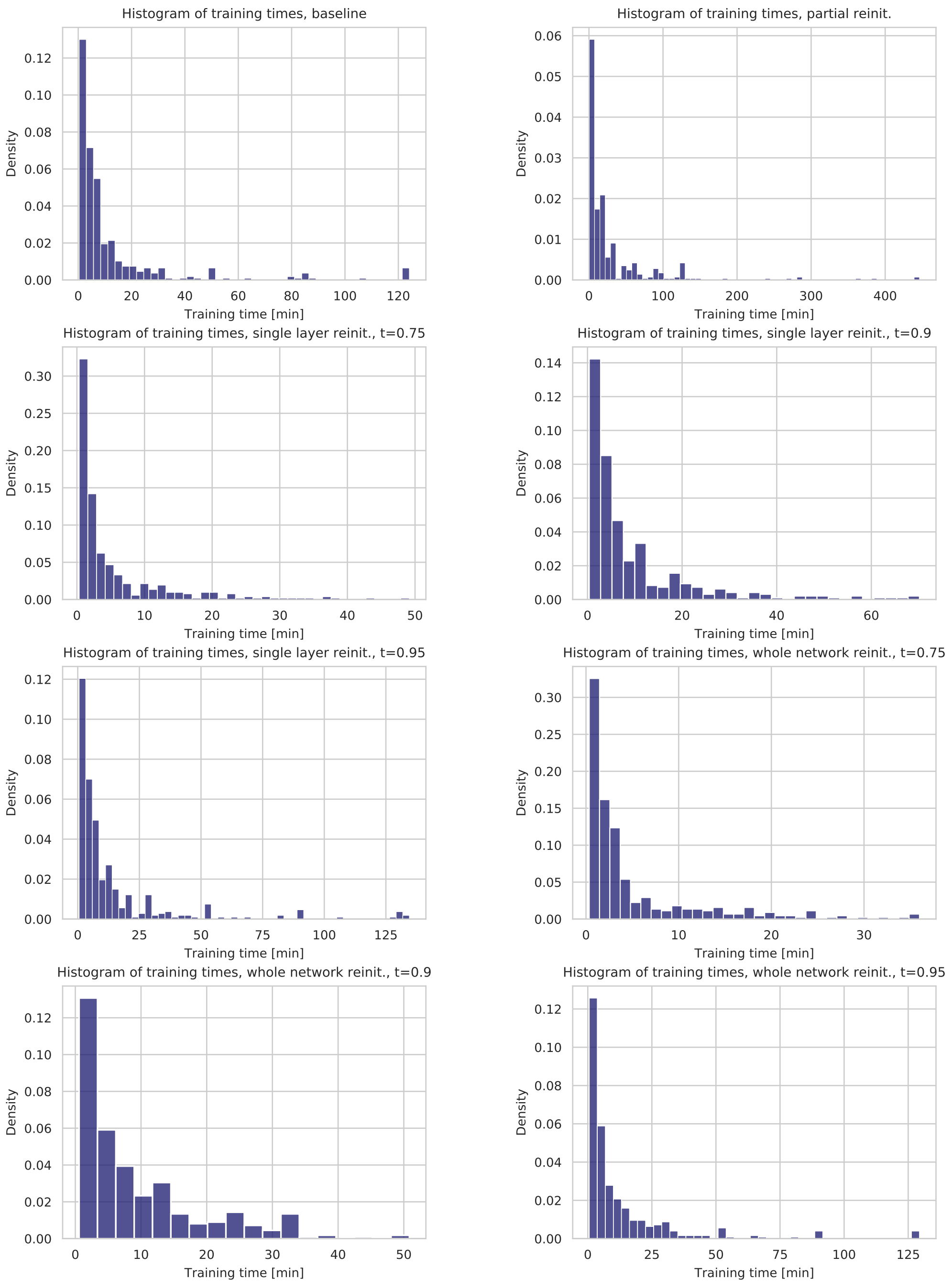


Figure 4.7: Histograms of model training times, for all runs and hyperparameter selections, according to the given reinitialization scenario.

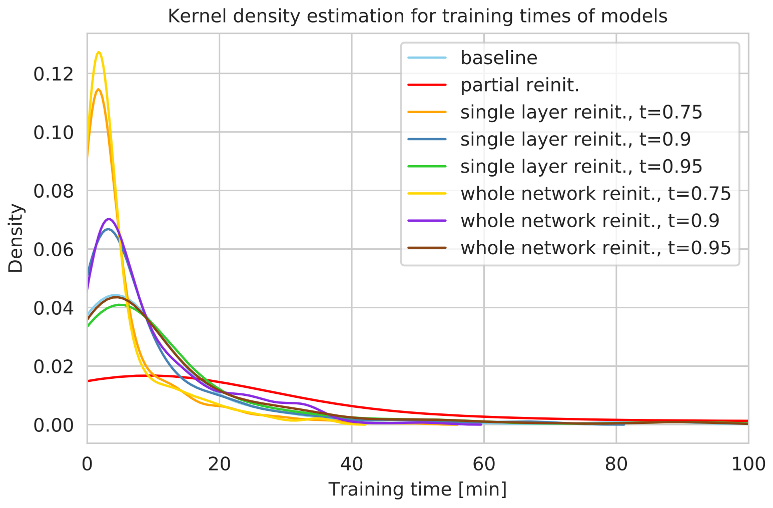


Figure 4.8: Kernel density estimation plot of model training times for different reinitialization scenarios.

validation loss over time, a single histogram aggregates results for all observations of a given experiment. It would not be sufficient to show only means with corresponding standard deviations because in these cases the variance of the training duration is very high.

Based on these comparisons, it is possible to conclude that the fastest methods have the reinitialization threshold set to 0.75 (both *whole network* and *single layer* approaches). In the case of higher thresholds, the training times have increased, but for methods with the threshold $t = 0.9$ the computational times were still lower than for the baseline. The duration of training of the baseline approach was comparable to methods with a threshold set to 0.95 which achieved more stable results than methods with lower thresholds. Definitely, the longest training times were related to *partial reinitialization*, although it was the most efficient method in terms of the reconstruction error of the best individuals. One of the reasons is that it is necessary to calculate the dead activations' coefficient for each particular neuron, not only for consecutive network layers, which extends the computation time, especially for wider networks.

4.8 Conclusions

In this chapter, we investigated selected autoencoder models, and on the basis of the results, proposed three network reinitialization methods that aim to alleviate the identified problem. We tested them using different datasets, also beyond the scope of hyperspectral images. We demonstrated that our approaches are able to improve network performance through minimization of the reconstruction error. For hyperspectral images, in many cases they statistically outperformed the baseline results. In the case of experiments on the MNIST dataset, we used more complex architectures and different hyperparameters values. We noticed that models that initially had low values of reconstruction errors, after the application of reinitialization methods, remained high-performing while low-performing models have been in many situations improved. It is promising that the presented methods may repair bad models without spoiling good ones. In the future, one can apply the reinitialization techniques in a classification scenario, for more complex architectures like convolutional neural networks and other datasets.

It should be emphasized that the problem of dying ReLU neurons is related to the suboptimal choice of network hyperparameters, especially the learning rate. Since in many cases the cost of network training is very high, it is not possible to perform a full optimization using many sets of hyperparameters. Hence an algorithm minimizing the negative impact of the dying neurons phenomenon on network performance would be useful. Furthermore, it is worth considering the reinitialization approach as a *watchdog* during the search for optimal hyperparameters and for a selection or early rejection of unpromising sets of networks hyperparameters.

The initial results of experiments on the FashionMNIST [161] and the CIFAR10 [76] datasets are consistent with those previously with the MNIST. We can see that neurons trained with ”weak” sets of hyperparameters can be improved, but we also notice that some models were deteriorated through reinitialization methods. The proposed solutions need further investigation in different configurations. Moreover, an important direction of future research is related to optimization of threshold values for the considered methods. We showed that it has an important impact on the performance of reinitialization methods and a more detailed analysis may lead to their improvements. Another option is related to adding random noise to the existing weights instead of generating entirely new values. These propositions need to be revised through a series of additional experiments in the future.

Table 4.4: Mean results of the network reinitialization methods and the baseline approach (with the corresponding standard deviation) for Experiments 4 and 5. A filled circle (●) indicates that a reinitialization method significantly outperforms baseline, according to the results of the Wilcoxon signed-rank test ($p < 0.05$).

init. method	reinitialization method	RMSE	abundance error	endmember error
Experiment 4 – Samson dataset, loss function: MSE				
XGN	baseline	0.036 ± 0.04	0.349 ± 0.04	0.744 ± 0.14
	partial reinitialization	0.089 ± 0.03	0.413 ± 0.08	**0.451 ± 0.24** ●
	single layer	0.019 ± 0.03 ●	0.310 ± 0.06 ●	0.777 ± 0.19
	whole network	**0.007 ± 0.00** ●	**0.276 ± 0.03** ●	0.687 ± 0.11 ●
XGU	baseline	0.052 ± 0.05	0.344 ± 0.05	0.723 ± 0.12
	partial reinitialization	0.089 ± 0.03	0.407 ± 0.08	**0.375 ± 0.03** ●
	single layer	0.020 ± 0.04 ●	0.307 ± 0.05 ●	0.707 ± 0.16
	whole network	**0.007 ± 0.00** ●	**0.270 ± 0.03** ●	0.695 ± 0.11
KHN	baseline	0.047 ± 0.04	0.365 ± 0.03	1.176 ± 0.13
	partial reinitialization	**0.024 ± 0.01** ●	**0.330 ± 0.03** ●	**1.100 ± 0.13** ●
	single layer	0.028 ± 0.01 ●	0.355 ± 0.04	1.153 ± 0.11
	whole network	0.027 ± 0.01 ●	0.355 ± 0.04	1.134 ± 0.13 ●
KHU	baseline	0.053 ± 0.05	0.357 ± 0.03	0.758 ± 0.19
	partial reinitialization	**0.007 ± 0.00** ●	**0.308 ± 0.04** ●	0.733 ± 0.12
	single layer	0.014 ± 0.01 ●	0.347 ± 0.03	0.732 ± 0.15
	whole network	0.008 ± 0.01 ●	0.316 ± 0.04 ●	**0.663 ± 0.17** ●
Experiment 5 – Samson dataset, loss function: SAD				
XGN	baseline	0.155 ± 0.01	0.378 ± 0.04	0.882 ± 0.15
	partial reinitialization	0.158 ± 0.00	**0.289 ± 0.02** ●	0.755 ± 0.09 ●
	single layer	**0.154 ± 0.01**	0.304 ± 0.04 ●	**0.730 ± 0.13** ●
	whole network	0.159 ± 0.01	0.317 ± 0.03 ●	0.823 ± 0.10 ●
XGU	baseline	0.158 ± 0.01	0.392 ± 0.04	0.879 ± 0.14
	partial reinitialization	0.159 ± 0.00	**0.283 ± 0.02** ●	**0.733 ± 0.10** ●
	single layer	**0.156 ± 0.01**	0.316 ± 0.04 ●	0.752 ± 0.10 ●
	whole network	0.157 ± 0.01	0.313 ± 0.04 ●	0.795 ± 0.09 ●
KHN	baseline	0.223 ± 0.05	0.389 ± 0.04	0.939 ± 0.15
	partial reinitialization	**0.182 ± 0.03** ●	**0.300 ± 0.02** ●	0.822 ± 0.07 ●
	single layer	0.240 ± 0.06	0.327 ± 0.04 ●	**0.797 ± 0.09** ●
	whole network	0.221 ± 0.06	0.329 ± 0.03 ●	0.906 ± 0.12
KHU	baseline	**0.138 ± 0.01**	0.385 ± 0.04	0.977 ± 0.14
	partial reinitialization	0.140 ± 0.01	**0.301 ± 0.02** ●	0.831 ± 0.08 ●
	single layer	**0.138 ± 0.01**	0.328 ± 0.03 ●	**0.827 ± 0.09** ●
	whole network	0.138 ± 0.02	0.329 ± 0.04 ●	0.851 ± 0.11 ●

Table 4.5: Mean results of the network reinitialization methods and the baseline approach (with the corresponding standard deviation) for Experiment 10. A filled circle (●) indicates that a reinitialization method significantly outperforms baseline, according to the results of the Wilcoxon signed-rank test ($p < 0.05$).

init. method	reinitialization method	RMSE	abundance error	endmember error
Experiment 10 – Jasper Ridge dataset, loss function: MSE				
XGN	baseline	0.020 ± 0.03	0.285 ± 0.04	0.894 ± 0.11
	partial reinitialization	**0.011 ± 0.00** ●	0.289 ± 0.03	0.893 ± 0.08
	single layer	0.014 ± 0.01	**0.259 ± 0.04** ●	**0.714 ± 0.10** ●
	whole network	0.119 ± 0.10	0.316 ± 0.09	1.016 ± 0.27
XGU	baseline	0.019 ± 0.03	0.281 ± 0.04	0.899 ± 0.11
	partial reinitialization	**0.011 ± 0.00** ●	0.281 ± 0.02	0.863 ± 0.08 ●
	single layer	0.017 ± 0.01	**0.252 ± 0.05** ●	**0.685 ± 0.13** ●
	whole network	0.112 ± 0.10	0.307 ± 0.10	1.017 ± 0.28
KHN	baseline	0.021 ± 0.03	0.300 ± 0.02	1.033 ± 0.11
	partial reinitialization	**0.011 ± 0.00** ●	0.300 ± 0.02	1.006 ± 0.06
	single layer	0.015 ± 0.01	0.301 ± 0.02	**0.999 ± 0.10**
	whole network	0.038 ± 0.04	**0.298 ± 0.02**	1.098 ± 0.12
KHU	baseline	0.037 ± 0.05	0.276 ± 0.04	0.881 ± 0.10
	partial reinitialization	**0.010 ± 0.00** ●	0.273 ± 0.03	0.827 ± 0.09 ●
	single layer	0.018 ± 0.03 ●	**0.261 ± 0.04** ●	**0.795 ± 0.14** ●
	whole network	0.028 ± 0.04	0.265 ± 0.05	0.909 ± 0.14

Table 4.6: Results of network reinitialization methods for 3 best models for each of the approaches applied. Models were selected according to RMSE on the validation set. Experiments were performed on the MNIST dataset. Results are averaged over 5 runs. Mean reconstruction errors are presented with the corresponding standard deviations. In the column entitled "no. of reinit." are presented mean numbers of iterations in which the reinitialization of neurons occurred while t denotes a threshold value of the dead activations' coefficient.

no.	method	t	architecture	batch size	learning rate	test RMSE	no. of reinit.
1	*partial reinit.*	0.99	2 layers, 500 neurons	64	0.0001	0.0854 ± 0.01	98.4
2	*partial reinit.*	0.99	2 layers, 500 neurons	128	0.001	0.0865 ± 0.01	15958.8
3	*partial reinit.*	0.99	2 layers, 500 neurons	64	0.001	0.0871 ± 0.00	34772.8
4	*single layer*	0.9	2 layers, 500 neurons	128	0.001	0.0902 ± 0.01	0
5	*whole network*	0.95	2 layers, 500 neurons	128	0.001	0.0902 ± 0.01	0
6	*baseline*	–	2 layers, 500 neurons	128	0.001	0.0902 ± 0.01	0
7	*whole network*	0.9	2 layers, 500 neurons	128	0.001	0.0902 ± 0.01	0
8	*single layer*	0.95	2 layers, 500 neurons	128	0.001	0.0902 ± 0.01	0
9	*single layer*	0.9	2 layers, 500 neurons	4	0.0001	0.0905 ± 0.01	4.6
10	*whole network*	0.95	2 layers, 500 neurons	128	0.0001	0.0909 ± 0.01	0
11	*whole network*	0.9	2 layers, 500 neurons	128	0.0001	0.0909 ± 0.01	0
12	*baseline*	–	2 layers, 500 neurons	128	0.0001	0.0909 ± 0.01	0
13	*single layer*	0.95	2 layers, 500 neurons	128	0.0001	0.0909 ± 0.01	0
14	*single layer*	0.9	2 layers, 500 neurons	128	0.0001	0.0909 ± 0.01	0
15	*baseline*	–	2 layers, 500 neurons	64	0.0001	0.0942 ± 0.00	0
16	*whole network*	0.9	2 layers, 500 neurons	64	0.0001	0.0942 ± 0.00	0
17	*whole network*	0.95	2 layers, 500 neurons	64	0.0001	0.0942 ± 0.00	0
18	*single layer*	0.95	2 layers, 500 neurons	64	0.0001	0.0942 ± 0.00	0
19	*single layer*	0.75	2 layers, 500 neurons	64	0.001	0.1554 ± 0.00	226.6
20	*single layer*	0.75	2 layers, 500 neurons	128	0.001	0.1564 ± 0.01	116
21	*single layer*	0.75	2 layers, 500 neurons	32	0.001	0.1627 ± 0.01	441.4
22	*whole network*	0.75	2 layers, 500 neurons	64	0.0001	0.1671 ± 0.01	23.8
23	*whole network*	0.75	2 layers, 500 neurons	64	0.001	0.1693 ± 0.01	77.8
24	*whole network*	0.75	2 layers, 100 neurons	128	0.001	0.1701 ± 0.00	20

Table 4.7: Results of Spearman's rank correlation coefficient (with p-value) between the reconstruction error on the test set without reinitialization method and the number of network reinitializations after the application of a selected method. Correlation values above 0.7 are shown in bold.

reinitialization method	correlation	p-value
whole network, $t = 0.75$	**0.717**	$1.97 \cdot 10^{-64}$
whole network, $t = 0.9$	**0.796**	$1.04 \cdot 10^{-88}$
whole network, $t = 0.95$	**0.843**	$2.15 \cdot 10^{-109}$
single layer, $t = 0.75$	0.672	$6.10 \cdot 10^{-54}$
single layer, $t = 0.9$	**0.787**	$2.01 \cdot 10^{-85}$
single layer, $t = 0.95$	**0.844**	$5.71 \cdot 10^{-110}$
partial reinit, $t = 0.99$	-0.037	0.46

Chapter 5

Conclusions

The following chapter summarizes the work described in this dissertation. We present a short discussion of topics and results from previous chapters, including a proof of the thesis. Finally, we outline a few ideas for further work.

5.1 Summary of the work carried out

The overview of neural network architectures used in further experiments was presented in Chapter 1. Furthermore, we introduced hyperspectral imaging and related challenges. In Chapter 2 we applied different algorithms for the classification of blood and blood-like substances using hyperspectral data. In order to study the problem of optimization of network architectures, we analyzed various state-of-the-art architectures, i.e. a recurrent network, 1D, 2D and 3D convolutional neural networks and a multilayer perceptron. We performed experiments according to the two approaches: *Hyperspectral Transductive Scenario* (HTC) and *Hyperspectral Inductive Scenario* (HIC). The first one is the most common in machine learning; i.e. the source image of both the training and the test datasets is the same. The second scenario is more demanding and, depending on the differences between datasets, can lead to low-performing models. We have made an exhaustive discussion of architectures and their results. While in the HTC scenario the overall accuracy exceeded 90% for the best methods, the network performance in the HIC scenario fluctuated between 57% and 100%.

The difficulties in HIC arise from, e.g. the differences in scene composition and sample characteristics as well as mixed spectra [80]. As a consequence, we studied a special type of neural network, i.e. autoencoders designed for spectral unmixing, and presented the results in Chapter 3. We performed several experiments using well-known HSI datasets and identified that the results of some runs are inconsistent and that the final pixel reconstruction is strongly dependent on the initial set of weights. We verified this hypothesis through the Kruskal-Wallis H-test [78] and the Conover-Iman post-hoc test [35] based on the results of a series of training sessions using different sets of hyperparameters, loss functions, datasets, architectures and weight initialization approaches. We conducted a total of 100,000 single autoencoder training sessions. Our aim was to determine whether at least one of the series of trained models is statistically significantly different from the others. In most of the considered cases, the test hypothesis about the equality of populations was rejected, which means that at least one population is significantly different from the others. In these situations, we also checked which pairs stand out. The results indicate that the network reconstruction error is dependent on the weight initialization. Furthermore, we presented results of spectral unmixing for each considered network setup and dataset used. We discussed the experiments carried out and the research problems encountered.

In Chapter 4, after the analysis of a subset of models from Chapter 3, we identified the dead activations' and dead neurons' phenomena in neural networks with ReLU activation function. This led us to the proposition of several network reinitialization methods that aim to mitigate the negative impact of dead activations and counteract them. We observed that in some cases it is possible to notice many zero activations, especially in the bottleneck of the tested autoencoders. We verified this remark using the Spearman's rank correlation coefficient and confirmed a strong relationship between the number of dead activations and the final network reconstruction error, for one of the network's layers in particular. We proposed three network reinitialization methods which are designed to alleviate the dead activations phenomenon: *whole network reinitialization* in which all network weights are drawn from scratch, *single layer reinitialization* performed for all weights of the one selected layer and *partial reinitialization* that changes only some weights of the selected layer. We evaluated the described methods for a subset of experiments from Section 3.5.4 and, in many cases, we got a statistically significant improvement in the network performance when compared to the baseline, i.e. the network without reinitialization. Furthermore, we applied the approach beyond the scope of hyperspectral images and performed experiments on the well-known MNIST dataset. The results obtained are promising,; however, further study is required to fully assess the potential of the presented methods beyond the scope of hyperspectral datasets.

5.1.1 Proving the dissertation thesis

In this dissertation, we discussed the topic of optimization of deep learning architectures for hyperspectral data classification. In Chapter 2, through an extensive study with different neural network architectures and various experiment scenarios on a representative dataset with blood and blood-like substances, we demonstrated the performance gain resulting from choosing an efficient architecture. We also identified a problem related to the HIC scenario which led to spectral unmixing, which was considered in the next chapter.

In Chapter 2 we also identified a network stability problem that relies on the fact that for a given set of hyperparameters some network training runs lead to scores of a lower level of performance compared to other runs. We observed that simpler architectures may still achieve competitive results while they remain easier to analyze. Thus, we selected linear autoencoders as proper architectures for further experiments described in Chapter 3, as the stability problem was also present in the case of autoencoders. Therefore, we performed an extensive statistical study and we stated that weight initialization has a statistically significant impact on network reconstruction error after the training session. We conducted experiments for different architectures, datasets, loss functions, weight initialization methods, etc. The results obtained in Chapters 2–3 confirm the first part of the dissertation thesis, i.e. **Optimization of deep learning network architectures improves the performance of neural networks for hyperspectral data**.

We decided to investigate selected network models trained in experiments from Chapter 3. We discovered that some models suffer from vanishing gradients. Furthermore, we identified that low-performing networks struggle with dead activation problems. Therefore, we proposed three network reinitialization methods that mitigate the negative impact of this phenomenon and, in many scenarios of the considered hyperspectral unmixing experiments, lead to statistically significantly better results. This confirms the second part of the thesis statement, i.e. **Weight reinitialization methods improve the performance of neural networks for hyperspectral data**. Therefore, the results presented in this dissertation fully confirm the thesis.

5.2 Future research

Based on the performed study, we can specify some promising research directions that can be considered as future topics:

- Checking whether the application of reinitialization methods improves the classification accuracy of multilayer perceptrons when compared to the baseline, i.e. networks trained without reinitialization steps. This idea can be verified e.g. through addition of a classification layer at the end of the initial pipeline and its training with the remaining layers being frozen. Finally, the whole network may be fine-tuned with a small value of the learning rate. Such an approach can be performed for all the reinitialization methods presented and compared to networks without these adjustments.

- Analysis of the application of reinitialization methods in convolutional neural networks. The number of dead activations may be calculated based on feature maps and some filters may be reinitialized. It is necessary to mention that this process has to be performed carefully because zero activations may be just connected with the lack of individual features, e.g. the absence of horizontal lines in a given sample when a filter is designed for their detection. Probably, a broader perspective for the quality assessment of filter responses for given samples will be useful.

- Using reinitialization methods as a *watchdog* indicating whether a given set of hyperparameters leads to high-performing models or not. When a number of reinitialization becomes large, then the training should be stopped and a current set of hyperparameters be rejected. This technique can save calculation time in a complex hyperparameter optimization process.

- Considering the addition of a random noise instead of a full replacement of weights in the case of models for which the number of dead activations exceeds the given threshold value. It could be a less disruptive approach for networks than the reinitialization of weights.

- Blood age estimation using the deep learning approach. In Chapter 2 we analyzed a dataset with *frame* images from 5 different moments of time and *comparison* scenes from 3 various days after substance spilling. We discussed the classification of several substances, but one should consider focusing only on blood and only assessing its age. This task has the potential for application in forensic science by crime investigators.

Appendix

Kruskal-Wallis H-test

Let us suppose that we have K data samples $\{X_1, X_2, ..., X_K\}$, where the i–th random sample, X_i, contains k_i observations, $i \in \{1, 2, ..., K\}$. We want to verify whether all samples come from the same population or if there exists at least one sample that is significantly different from the others. To use a parametric method like a one-way analysis of variance (ANOVA) a variance among all populations has to be equal. In the opposite case, a nonparametric test such as the Kruskal-Wallis H-test for several independent samples has to be chosen. It is an extension of the Mann-Whitney approach for two independent samples [78, 156]. Let $F(X_i)$ denotes the distribution of the i–th random sample. We assume that if the distribution of at least one random sample differs from the others, it is due to translation or shift, i.e. if for $i, j, \in \{1, 2, ..., K\}$ $F(X_i) \neq F(X_j)$, then $F(X_i) = F(X_j + C)$, where C is a constant value. We also denote by $\mathbf{E}(X_i)$ the expected value of the random sample X_i while $M = \sum_{i=1}^{K} k_i$ is the total number of observations. The test hypotheses can be described as follows [156]:

H_0: $\mathbf{E}(X_1) = \mathbf{E}(X_2) = ... = \mathbf{E}(X_K)$, i.e. the expected values (means) of all K samples are equal.

H_1: $\exists\ i, j \in \{1, 2, ..., K\}$ $\mathbf{E}(X_i) \neq \mathbf{E}(X_j)$, that is, the expected value of at least one sample is different from the expected value of at least one other sample.

To perform the Kruskal-Wallis one-way analysis of variance, it is necessary to replace original M observations by their ranks. More specifically, observations from all samples must be sorted in ascending order. Then, the smallest observation has to be replaced by 1, the second smallest by 2, etc., up to the largest one, which is to be assigned rank M. If there are ties, i.e. observations having the same values, there are replaced by the mean of the ranks which would have been assigned if all observations were different. After that, one has to compute the sum of the ranks for each sample. If there are no tied observations among the samples, the test statistic is expressed by the following pattern:

$$H = \frac{12}{M(M+1)} \sum_{i=1}^{K} \frac{R_i^2}{k_i} - 3(M+1), \tag{1}$$

where R_i is the sum of the ranks for the observations from the i–th sample. Assuming that the H_0 hypothesis is true, the distribution of the H-test statistic can be approximated by the chi-square distribution with $K - 1$ degrees of freedom. We reject the null hypothesis at significance level $\alpha > 0$ if H is greater than the value of its $(1 - \alpha)$-th quantile.

Suppose that we have $t > 0$ groups of tied observations. In this case, Equation 1 must be divided by the value of $1 - \frac{\sum_{i=1}^{t} T_i}{M^3 - M}$, where $T_i = t_i(t_i - 1)(t_i + 1)$ while t_i is the number of ties in the i–th group of ties. Finally, the H-test statistic for samples with tied observations is derived as follows:

$$H = \frac{\frac{12}{M(M+1)} \sum_{i=1}^{K} \frac{R_i^2}{k_i} - 3(M+1)}{1 - \frac{\sum_{i=1}^{t} T_i}{M^3 - M}.}. \tag{2}$$

Conover-Iman post-hoc test

The Conover-Iman post-hoc test is the Fisher's Significant Difference (FSD) procedure applied for ranks instead of original observations [35, 156]. The main assumption is that this test can be performed if and only if the H_0 hypothesis of the Kruskal-Wallis test is rejected. In such a situation, means of the i–th and the j–th samples are different if the following condition is met:

$$\left| \frac{R_i}{k_i} - \frac{R_j}{k_j} \right| > t_{1-\frac{\alpha}{2}} \sqrt{\left(S^2 \frac{M - 1 - H}{M - K} \right) \left(\frac{1}{k_i} + \frac{1}{k_j} \right)}, \tag{3}$$

where $t_{1-\frac{\alpha}{2}}$ is the $(1 - \frac{\alpha}{2})$-quantile of the t-distribution with $M - K$ degrees of freedom, while S^2 is the variance of ranks for populations with ties:

$$S^2 = \frac{1}{M-1} \left(\sum_{m=1}^{K} \sum_{n=1}^{k_m} R_{m,n}^2 - \frac{M(M+1)^2}{4} \right), \tag{4}$$

where $R_{m,n}$ is the rank of the n–th observation of the m–th sample. If there are no tied observations, S^2 reduces to $\frac{M(M+1)}{12}$. The application of the Conover-Iman test is simpler than multiple repetitions of the Mann-Whitney test due to the fact that there is no need for multiple ranking recalculations to compare each pair [35].

Spearman's rank correlation coefficient

Suppose that we have a k–element simple random sample $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$ selected from the random variables $\mathcal{X}$ and $\mathcal{Y}$. For $j \in \{1, ..., k\}$, we replace (x_j, y_j) by (q_j, r_j), where q_j is the rank of observation x_j in the sample $x_1, x_2, ..., x_k$ while r_j is the rank of observation y_j in the sample $y_1, y_2, ..., y_k$. A rank of x_j, i.e. q_j, is its position in the ordered sequence of observations.

A Spearman's rank correlation coefficient, r_s, between two random variables $\mathcal{X}$ and $\mathcal{Y}$ is expressed by the following formula [73]:

$$r_s = \frac{12}{k(k^2-1)} \sum_{j=1}^{k} q_j \cdot r_j - \frac{3(k+1)}{k-1}. \tag{5}$$

Wilcoxon signed-rank test

The Wilcoxon signed-rank test is applied for the comparison of paired sample observations. This test is a non-parametric analogue of the two-sample t-test [73, 74]. Originally, it was described by Frank Wilcoxon in his paper [158].

Suppose that we have two random variables $\mathcal{X}$ and $\mathcal{Y}$ with continuous cumulative distribution functions F and G, respectively. Let us also assume that we have k pairs of observations $(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)$ where x_i, $i \in \{1, 2, ..., k\}$, is sampled from $\mathcal{X}$ while y_j, $j \in \{1, 2, ..., k\}$, is sampled from $\mathcal{Y}$. Pairs have to be mutually independent, but two variables of a given pair may be dependent. Furthermore, we require that pairs have the same two-dimensional distribution. For each pair, we calculate the difference $z_i = x_i - y_i$, $i \in \{1, 2, ..., k\}$. We do not assume that a distribution of differences is normal. We have to verify a null hypothesis H_0 that $F(t) = G(t)$ for each $t \geq 0$, against the alternative hypothesis H_1 which states that $F(t) \geq G(t)$ for each $t \geq 0$ and $F \neq G$. Similarly as in the case of the Spearman's rank correlation coefficient, during the calculation of the test statistic, we rank absolute differences $|x_i - y_i|$. Then, we sum the ranks corresponding to positive differences.

Bibliography

[1] GRU. `https://pytorch.org/docs/stable/generated/torch.nn.GRU.html`. In: PyTorch Docs, Accessed: 2022-06-22.

[2] Local Response Normalization. `https://pytorch.org/docs/stable/generated/torch.nn.LocalResponseNorm.html#torch.nn.LocalResponseNorm`. In: PyTorch Docs, Accessed: 2022-08-01.

[3] Image recognition with deep neural networks and its use cases. `https://www.altexsoft.com/blog/image-recognition-neural-networks-use-cases/`, December 2019. In: AltexSoft blog, Accessed: 2022-04-22.

[4] Plotting with categorical data. `https://seaborn.pydata.org/tutorial/categorical.html#categorical-tutorial`, 2022. In: Seaborn user guide and tutorial, Accessed: 2022-05-31.

[5] M. Aalders and L. Wilk. *Investigating the Age of Blood Traces: How Close Are We to Finding the Holy Grail of Forensic Science?*, pages 109–128. Springer International Publishing, Cham, 2019.

[6] C. C. Aggarwal. *Neural Networks and Deep Learning.* Springer, 2018.

[7] M. Ahmad, S. Shabbir, S. K. Roy, D. Hong, X. Wu, J. Yao, A. M. Khan, M. Mazzara, S. Distefano, and J. Chanussot. Hyperspectral image classification—traditional to deep models: A survey for future prospects. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 15:968–999, 2022.

[8] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[9] I. Alabdulmohsin, H. Maennel, and D. Keysers. *The Impact of Reinitialization on Generalization in Convolutional Neural Networks*, 2021. arXiv:2109.00267.

[10] Y. Amit and P. Felzenszwalb. *Object Detection*, pages 537–542. Springer US, Boston, MA, 2014.

[11] S. Andrilli and D. Hecker. A light-hearted look at linear algebra terms. In *Elementary Linear Algebra (Fifth Edition)*, page xix. Academic Press, Boston, 2016.

[12] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christe, and S. Mougiakakou. Lung pattern classification for interstitial lung diseases using a deep convolutional neural network. *IEEE Transactions on Medical Imaging*, 35(5):1207 – 1216, 2016.

[13] N. Audebert, B. L. Saux, and S. Lefèvre. Deep learning for classification of hyperspectral data: A comparative review. *IEEE Geoscience and Remote Sensing Magazine*, 7:159–173, 2019.

[14] O. K. Baskurt. Mechanisms of blood rheology alterations. In O. K. Baskurt, M. R. Hardeman, M. W. Rampling, and H. J. Meiselman, editors, *Handbook of Hemorheology and Hemodynamics*, pages 170–190. IOS PRESS, 2007.

[15] A. Ben Hamida, A. Benoit, P. Lambert, and C. Ben Amar. 3-D deep learning approach for remote sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 56(8):4420–4434, 2018.

[16] J. S. Bhatt and M. V. Joshi. Deep learning in hyperspectral unmixing: A review. In *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, pages 2189–2192, 2020.

[17] J. M. Bioucas-Dias. A variable splitting augmented lagrangian approach to linear spectral unmixing. In *2009 First Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing*, pages 1–4, 2009.

[18] J. M. Bioucas-Dias and J. M. P. Nascimento. Estimation of signal subspace on hyperspectral data. In L. Bruzzone, editor, *Image and Signal Processing for Remote Sensing XI*, volume 5982, pages 191 – 198. International Society for Optics and Photonics, SPIE, 2005.

[19] J. M. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. Nasrabadi, and J. Chanussot. Hyperspectral remote sensing data analysis and future challenges. *IEEE Geoscience and Remote Sensing Magazine*, 1(2):6–36, 2013.

[20] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot. Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 5(2):354–379, 2012.

[21] P. Bizopoulos and D. Koutsouris. Deep learning in cardiology. *IEEE Reviews in Biomedical Engineering*, 12:168–193, 2019.

[22] J. W. Boardman, F. A. Kruse, and R. O. Green. Mapping target signatures via partial unmixing of AVIRIS data. In *Summaries of the Fifth Annual JPL Airborne Earth Science Workshop. Volume 1: AVIRIS Workshop*, 1995.

[23] R. A. Borsoi, T. Imbiriba, and J. C. M. Bermudez. Deep Generative Endmember Modeling: An Application to Unsupervised Spectral Unmixing. *IEEE Transactions on Computational Imaging*, 6:374–384, 2020.

[24] R. H. Bremmer, G. Edelman, T. D. Vegter, T. Bijvoets, and M. C. Aalders. Remote spectroscopic identification of bloodstains. *Journal of Forensic Sciences*, 56(6):1471–1475, 2011.

[25] R. H. Bremmer, A. Nadort, T. G. van Leeuwen, M. J. van Gemert, and M. C. Aalders. Age estimation of blood stains by hemoglobin derivative determination using reflectance spectroscopy. *Forensic Science International*, 206(1):166–171, 2011.

[26] S. Broedbeck. Introduction to bloodstain pattern analysis. In S. H. James, P. E. Kish, and T. P. Sutton, editors, *Principles of Bloodstain Pattern Analysis: Theory and Practice (1st ed.)*, pages 1–10. CRC Press, 2012.

[27] W. Burger and M. J. Burger. *Principles of Digital Image Processing: Fundamental Techniques*. Springer London, 2009.

[28] S. Cadd, B. Li, P. Beveridge, W. T. O'Hare, A. Campbell, and M. Islam. The non-contact detection and identification of blood stained fingerprints using visible wavelength hyperspectral imaging: Part II effectiveness on a range of substrates. *Science & Justice*, 56(3):191–200, 2016.

[29] T. A. Caswell, M. Droettboom, A. Lee, J. Hunter, E. Firing, D. Stansby, J. Klymak, E. S. de Andrade, T. Hoffmann, J. H. Nielsen, and et al. *matplotlib/matplotlib v3.1.2*, 2019.

[30] K.-C. Chang, P.-H. Hsieh, M.-Y. Wu, Y.-C. Wang, J.-Y. Chen, F.-J. Tsai, E. S. Shih, M.-J. Hwang, and T.-C. Huang. Usefulness of machine learning-based detection and classification of cardiac arrhythmias with 12-lead electrocardiograms. *Canadian Journal of Cardiology*, 37(1):94–104, 2021.

[31] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014.

[32] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, 2014. arXiv:1412.3555.

[33] Z. Chunhui, G. Bing, Z. Lejun, and W. Xiaoqing. Classification of hyperspectral imagery based on spectral gradient, SVM and spatial random forest. *Infrared Physics & Technology*, 95:61–69, 2018.

[34] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[35] W. J. Conover and R. L. Iman. *On Multiple-Comparisons Procedures. Informal report*, 1979.

[36] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[37] R. Delgado and X.-A. Tibau. Why Cohen's Kappa should be avoided as performance measure in classification. *PLOS ONE*, 14(9):1–26, 09 2019.

[38] A.-V. Diaconu and A. C. Dascalescu. Correlation distribution of adjacent pixels randomness test for image encryption. In *Proceedings of the Romanian Academy, Series A*, volume 18, pages 351–360, 2017.

[39] J. Dissing, A. Søndervang, and S. Lund. Exploring the limits for the survival of dna in blood stains. *Journal of Forensic and Legal Medicine*, 17(7):392–396, 2010.

[40] K. C. Doty, G. McLaughlin, and I. K. Lednev. A raman „spectroscopic clock" for bloodstain age determination: the first week after deposition. volume 408, pages 3993–4001, June 2016.

[41] Z. Dou, K. Gao, X. Zhang, H. Wang, and J. Wang. Blind hyperspectral unmixing using dual branch deep autoencoder with orthogonal sparse prior. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2428–2432, 2020.

[42] Z. Dou, K. Gao, X. Zhang, H. Wang, and J. Wang. Hyperspectral unmixing using orthogonal sparse prior-based autoencoder with hyper-laplacian loss and data-driven outlier detection. *IEEE Transactions on Geoscience and Remote Sensing*, 58(9):6550–6564, 2020.

[43] G. Edelman, E. Gaston, T. van Leeuwen, P. Cullen, and M. Aalders. Hyperspectral imaging for non-contact analysis of forensic traces. *Forensic Science International*, 223(1):28–39, 2012.

[44] G. Edelman, T. G. van Leeuwen, and M. C. Aalders. Hyperspectral imaging for the age estimation of blood stains at the crime scene. *Forensic Science International*, 223(1):72–77, 2012.

[45] G. Eraslan, Ž. Avsec, J. Gagneur, and F. J. Theis. Deep learning: new computational modelling techniques for genomics. *Nature Reviews Genetics*, 20(7):389 – 403, 2019.

[46] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[47] A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, page 148–155, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[48] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 2017.

[49] P. Ghamisi, J. Plaza, Y. Chen, J. Li, and A. J. Plaza. Advanced spectral classifiers for hyperspectral images: A review. *IEEE Geoscience and Remote Sensing Magazine*, 5(1):8–32, 2017.

[50] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[51] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[52] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[53] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[54] Y. Guan, Q. Li, H. Liu, Z. Zhu, and Y. Wang. Pathological leucocyte segmentation algorithm based on hyperspectral imaging technique. *Optical Engineering*, 51(5):053202, 2012.

[55] A. J. Guo and F. Zhu. Improving deep hyperspectral image classification performance with spectral unmixing. *Signal Processing*, 183:107949, 2021.

[56] J. A. Gutiérrez-Gutiérrez, A. Pardo, E. Real, J. M. López-Higuera, and O. M. Conde. Custom scanning hyperspectral imaging system for biomedical applications: Modeling, benchmarking, and specifications. *Sensors*, 19(7), 2019.

[57] P. Głomb, M. Romaszewski, M. Cholewa, and K. Domino. Application of hyperspectral imaging and machine learning methods for the detection of gunshot residue patterns. *Forensic Science International*, 290:227–237, 2018.

[58] S. Han, J. Pool, S. Narang, H. Mao, E. Gong, S. Tang, E. Elsen, P. Vajda, M. Paluri, J. Tran, B. Catanzaro, and W. J. Dally. DSD: dense-sparse-dense training for deep neural networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[59] C. Harris, K. Millman, S. Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. Smith, R. Kern, M. Picus, S. Hoyer, M. Kerkwijk, M. Brett, A. Haldane, J. Río, M. Wiebe, P. Peterson, and T. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 09 2020.

[60] K. He, X. Zhang, S. Ren, and J. Sun. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015. arXiv:1502.01852.

[61] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[62] A. Holzner. Variance calculation relu function (deep learning). `https://stats.stackexchange.com/questions/138035/variance-calculation-relu-function-deep-learning`, 2015. In: Cross Validated, Accessed: 2022-02-18.

[63] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[64] M.-P. Hosseini, T. X. Tran, D. Pompili, K. Elisevich, and H. Soltanian-Zadeh. Multimodal data analysis of epileptic EEG and rs-fMRI via deep learning and edge computing. *Artificial Intelligence in Medicine*, 104, 2020.

[65] W. Hu, Y. Huang, F. Zhang, and H. Li. Deep convolutional neural networks for hyperspectral image classification. *Journal of Sensors*, 2015.

[66] J. D. Hunter. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[67] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org, 2015.

[68] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[69] C. Kandaswamy, L. M. Silva, L. A. Alexandre, J. M. Santos, and J. M. de Sá. Improving deep neural network performance by reusing features trained with

transductive transference. In S. Wermter, C. Weber, W. Duch, T. Honkela, P. Koprinkova-Hristova, S. Magg, G. Palm, and A. E. P. Villa, editors, *Artificial Neural Networks and Machine Learning – ICANN 2014*, pages 265–272, Cham, 2014. Springer International Publishing.

[70] N. Keshava and J. Mustard. Spectral unmixing. *IEEE Signal Processing Magazine*, 19(1):44–57, 2002.

[71] S. Kind, D. Patterson, and G. Owen. Estimation of the age of dried blood stains by a spectrophotometric method. *Forensic Science*, 1(1):27–54, 1972.

[72] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[73] J. Koronacki and J. Mielniczuk. *Statystyka dla studentów kierunków technicznych i przyrodniczych, Wydanie trzecie [Statistics for students of technical and natural sciences, Third edition]*. Wydawnictwa Naukowo-Techniczne, Warsaw, 2006.

[74] S. Kotz and N. L. Johnson. *Breakthroughs in statistics: Methodology and Distribution*. Springer–Verlag New York, New York, NY, 1992.

[75] R. Kozma, R. Ilin, and H. T. Siegelmann. Evolution of abstraction across layers in deep learning neural networks. *Procedia Computer Science*, 144:203–213, 2018. INNS Conference on Big Data and Deep Learning.

[76] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*, April 2009. Technical Report TR-2009.

[77] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[78] W. H. Kruskal and W. A. Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952.

[79] K. Książek, P. Głomb, M. Romaszewski, M. Cholewa, B. Grabowski, and K. Búza. Improving autoencoder training performance for hyperspectral unmixing with network reinitialisation. In *Image Analysis and Processing – ICIAP 2022: 21st International Conference, Lecce, Italy, May 23–27, 2022, Proceedings, Part I*, page 391–403, Berlin, Heidelberg, 2022. Springer-Verlag.

[80] K. Książek, M. Romaszewski, P. Głomb, B. Grabowski, and M. Cholewa. Blood stain classification with hyperspectral imaging and deep neural networks. *Sensors*, 20(22), 2020.

[81] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 05 2015.

[82] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[83] H. Lee and H. Kwon. Going deeper with contextual cnn for hyperspectral image classification. *IEEE Transactions on Image Processing*, 26(10):4843–4855, 2017.

[84] H. Levene. Robust tests for equality of variance. In *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling*, pages 278–292. Stanford University Press, 1960.

[85] B. Li, P. Beveridge, W. T. O'Hare, and M. Islam. The application of visible wavelength reflectance hyperspectral imaging for the detection and identification of blood stains. *Science & Justice*, 54(6):432–438, 2014.

[86] J. Li, A. Agathos, D. Zaharie, J. M. Bioucas-Dias, A. Plaza, and X. Li. Minimum volume simplex analysis: A fast algorithm for linear hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 53(9):5067–5082, 2015.

[87] L. Li, C. Wang, W. Li, and J. Chen. Hyperspectral image classification by adaboost weighted composite kernel extreme learning machines. *Neurocomputing*, 275:1725–1733, 2018.

[88] 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*, 57(9):6690–6709, 2019.

[89] X. Li, H. Xiong, H. An, C.-Z. Xu, and D. Dou. RIFLE: Backpropagation in depth for deep transfer learning through re-initializing the fully-connected LayEr. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6010–6019. PMLR, 13–18 Jul 2020.

[90] Y. Li and F. Liu. Adaptive Gaussian noise injection regularization for neural networks. In M. Han, S. Qin, and N. Zhang, editors, *Advances in Neural Networks – ISNN 2020*, pages 176–189, Cham, 2020. Springer International Publishing.

[91] Y. Li, H. Zhang, and Q. Shen. Spectral–spatial classification of hyperspectral imagery with 3D convolutional neural network. *Remote Sensing*, 9(1), 2017.

[92] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. *Tune: A Research Platform for Distributed Model Selection and Training*, 2018.

[93] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[94] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan. A survey on evolutionary neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2021.

[95] B. Lu, P. D. Dao, J. Liu, Y. He, and J. Shang. Recent advances of hyperspectral imaging technology and applications in agriculture. *Remote Sensing*, 12(16), 2020.

[96] G. Lu and B. Fei. Medical hyperspectral imaging: a review. *Journal of Biomedical Optics*, 19(1):010901, 2014.

[97] L. Lu. Dying ReLU and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, Jun 2020.

[98] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu. Neural architecture optimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7827–7838, Red Hook, NY, USA, 2018. Curran Associates Inc.

[99] D. Manolakis and G. Shaw. Detection algorithms for hyperspectral imaging applications. *IEEE Signal Processing Magazine*, 19(1):29–43, 2002.

[100] F. B. Marion, L. B. Larry, and A. L. David. 220 band AVIRIS hyperspectral image data set: June 12, 1992 Indian Pine Test Site 3, 2015.

[101] S. Matteoli, M. Diani, and G. Corsini. A tutorial overview of anomaly detection in hyperspectral images. *IEEE Aerospace and Electronic Systems Magazine*, 25(7):5–28, 2010.

[102] M. L. McHugh. Interrater reliability: the kappa statistic. *Biochemia Medica*, pages 276–282, 10 2012.

[103] M. C. Meinke, G. J. Müller, J. Helfmann, and M. Friebel. Optical properties of platelets and blood plasma and their influence on the optical behavior of whole blood in the visible to near infrared wavelength range. *Journal of Biomedical Optics*, 12(1):1 – 9, 2007.

[104] L. Miao and H. Qi. Endmember extraction from highly mixed data using minimum volume constrained nonnegative matrix factorization. *IEEE Transactions on Geoscience and Remote Sensing*, 45(3):765–777, 2007.

[105] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3523–3542, 2022.

[106] A. Mohan and M. Venkatesan. Hybridcnn based hyperspectral image classification using multiscale spatiospectral features. *Infrared Physics & Technology*, 108:103326, 2020.

[107] C. Molnar. *Interpretable Machine Learning.* 2022. Second Edition.

[108] D. C. Montgomery. *Design and Analysis of Experiments.* John Wiley & Sons, 2013.

[109] L. Mou, P. Ghamisi, and X. X. Zhu. Deep recurrent neural networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 55(7):3639–3655, 2017.

[110] J. Nascimento and J. Dias. Vertex component analysis: a fast algorithm to unmix hyperspectral data. *IEEE Transactions on Geoscience and Remote Sensing*, 43(4):898–910, 2005.

[111] O. Okwuashi and C. E. Ndehedehe. Deep support vector machine for hyperspectral image classification. *Pattern Recognition*, 103:107298, 2020.

[112] B. T. Ong, K. Sugiura, and K. Zettsu. Dynamically pre-trained deep recurrent neural networks using environmental monitoring data for predicting pm2.5. *Neural Computing and Applications*, 27(6):1553–1566, 2016.

[113] S. Ozkan, B. Kaya, and G. B. Akar. Endnet: Sparse autoencoder network for endmember extraction and hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 57(1):482–496, 2019.

[114] B. Palsson, J. Sigurdsson, J. R. Sveinsson, and M. O. Ulfarsson. Hyperspectral unmixing using a neural network autoencoder. *IEEE Access*, 6:25646–25656, 2018.

[115] B. Palsson, J. R. Sveinsson, and M. O. Ulfarsson. Blind hyperspectral unmixing using autoencoders: A critical comparison. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 15:1340–1372, 2022.

[116] B. Palsson, M. O. Ulfarsson, and J. R. Sveinsson. Convolutional autoencoder for spatial-spectral hyperspectral unmixing. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 357–360, 2019.

[117] B. Pan, Z. Shi, and X. Xu. Mugnet: Deep learning for hyperspectral image classification using limited samples. *ISPRS Journal of Photogrammetry and Remote Sensing*, 145:108–119, 2018. Deep Learning RS Data.

[118] M. Paoletti, J. Haut, J. Plaza, and A. Plaza. Deep learning classifiers for hyperspectral imaging: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 158:279–317, 2019.

[119] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[120] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[121] Y. Qian, S. Jia, J. Zhou, and A. Robles-Kelly. Hyperspectral unmixing via $l_{1/2}$ sparsity-constrained nonnegative matrix factorization. *IEEE Transactions on Geoscience and Remote Sensing*, 49(11):4282–4297, 2011.

[122] Y. Qian, F. Xiong, Q. Qian, and J. Zhou. Spectral mixture model inspired network architectures for hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 58(10):7418–7434, 2020.

[123] J. Qin, K. Chao, M. S. Kim, R. Lu, and T. F. Burks. Hyperspectral and multispectral imaging for evaluating food safety and quality. *Journal of Food Engineering*, 118(2):157–171, 2013.

[124] Y. Qu and H. Qi. udas: An untied denoising autoencoder with sparsity for spectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 57(3):1698–1712, 2019.

[125] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions, 2017. arXiv:1710.05941.

[126] S. Z. Ramadan. Methods used in computer-aided diagnosis for breast cancer detection using mammograms: A review. *Journal of Healthcare Engineering*, pages 1–21, March 2020.

[127] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[128] J. Reback, jbrockmendel, W. McKinney, J. Van den Bossche, T. Augspurger, M. Roeschke, S. Hawkins, P. Cloud, gfyoung, Sinhrks, P. Hoefler, A. Klein,

T. Petersen, J. Tratner, C. She, W. Ayd, S. Naveh, J. Darbyshire, M. Garcia, R. Shadrach, J. Schendel, A. Hayden, D. Saxton, M. E. Gorelli, F. Li, M. Zeitlin, V. Jancauskas, A. McMaster, T. Wörtwein, and P. Battiston. *pandas-dev/pandas: Pandas 1.4.2*, 2022.

[129] J. Redmon and A. Farhadi. *YOLOv3: An Incremental Improvement*, 2018. arXiv:1804.02767.

[130] P. Ren, Y. Xiao, X. Chang, P. Huang, Z. Li, X. Chen, and X. Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Computing Surveys*, 54(4), 2021.

[131] B. Rister and D. L. Rubin. *Probabilistic bounds on neuron death in deep rectifier networks*, 2021. arXiv:2007.06192.

[132] M. Romaszewski, P. Głomb, A. Sochan, and M. Cholewa. A dataset for evaluating blood detection in hyperspectral images. `https://zenodo.org/record/3984905`, 2020.

[133] M. Romaszewski, P. Głomb, A. Sochan, and M. Cholewa. A dataset for evaluating blood detection in hyperspectral images. *Forensic Science International*, 320:110701, 2021.

[134] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, New Jersey, 2010.

[135] S. Santra, J.-W. Hsieh, and C.-F. Lin. Gradient descent effects on differential neural architecture search: A survey. *IEEE Access*, 9:89602–89618, 2021.

[136] J. R. Schott. *Remote Sensing. The Image Chain Approach. Second Edition.* Oxford University Press, 2007.

[137] C. Shi, D. Liao, T. Zhang, and L. Wang. Hyperspectral image classification based on 3d coordination attention mechanism network. *Remote Sensing*, 14(3), 2022.

[138] P. Simard, D. Steinkraus, and J. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, 2003.

[139] R. B. Smith. Introduction to hyperspectral imaging. `https://www.microimages.com/documentation/Tutorials/hyprspec.pdf`, 2012. By: MicroImages Inc., Accessed: 2022-06-27.

[140] D. Stein, S. Beaven, L. Hoff, E. Winter, A. Schaum, and A. Stocker. Anomaly detection from hyperspectral imagery. *IEEE Signal Processing Magazine*, 19(1):58–69, 2002.

[141] M. B. Stuart, A. J. S. McGonigle, and J. R. Willmott. Hyperspectral imaging in environmental monitoring: A review of recent developments and technological advances in compact field deployable systems. *Sensors*, 19(14), 2019.

[142] Y. Su, J. Li, A. Plaza, A. Marinoni, P. Gamba, and S. Chakravortty. DAEN: deep autoencoder networks for hyperspectral unmixing. *IEEE Transactions on Geoscience and Remote Sensing*, 57(7):4309–4321, 2019.

[143] M. Sugiyama, M. Krauledat, and K.-R. Müller. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(35):985–1005, 2007.

[144] Y. Sun, B. Xue, M. Zhang, and G. G. Yen. A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE Transactions on Neural Networks and Learning Systems*, 30(8):2295–2309, 2019.

[145] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv. Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Transactions on Cybernetics*, 50(9):3840–3854, 2020.

[146] I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 1017–1024, Madison, WI, USA, 2011. Omnipress.

[147] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[148] M. A. Terpilowski. scikit-posthocs: Pairwise multiple comparison tests in python. *Journal of Open Source Software*, 4(36):1169, 2019.

[149] S. Trajanovski, C. Shan, P. J. C. Weijtmans, S. G. B. de Koning, and T. J. M. Ruers. Tongue tumor detection in hyperspectral images using deep learning semantic segmentation. *IEEE Transactions on Biomedical Engineering*, 68(4):1330–1340, 2021.

[150] Y. Tsuboi, H. Kashima, S. Hido, S. Bickel, and M. Sugiyama. Direct density ratio estimation for large-scale covariate shift adaptation. volume 17, 01 2008.

[151] J. Umbreit. Methemoglobin—it's not just blue: A concise review. *American Journal of Hematology*, 82(2):134–144, 2007.

[152] J. van der Burgt. Neuron decay and reinitialization: a Bayesian regularizer for neural nets, 11 2021.

[153] A. Villa, J. Chanussot, J. A. Benediktsson, and C. Jutten. Spectral unmixing for the classification of hyperspectral images at a finer spatial resolution. *IEEE Journal of Selected Topics in Signal Processing*, 5(3):521–533, 2011.

[154] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[155] X. Wang, H. Qian, E. J. Ciaccio, S. K. Lewis, G. Bhagat, P. H. Green, S. Xu, L. Huang, R. Gao, and Y. Liu. Celiac disease diagnosis from videocapsule endoscopy images with residual learning and deep feature extraction. *Computer Methods and Programs in Biomedicine*, 187, 2020.

[156] S. Washington, J. Leonard, D. G. Manning, C. Roberts, B. Williams, A. R. Bacchus, A. Devanhalli, J. Ogle, and D. Melcher. *Scientific Approaches to Transportation Research: Volume 2*. NCHRP Project. National Cooperative Highway Research Program Transportation Research Board of the National Academies, Washington, 2002.

[157] M. L. Waskom. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021.

[158] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[159] M. E. Winter. N-FINDR: an algorithm for fast autonomous spectral endmember determination in hyperspectral data. In M. R. Descour and S. S. Shen, editors, *Imaging Spectrometry V*, volume 3753, pages 266 – 275. International Society for Optics and Photonics, SPIE, 1999.

[160] Y. Xia. Chapter eleven - correlation and association analyses in microbiome study integrating multiomics in health and disease. In J. Sun, editor, *The Microbiome in Health and Disease*, volume 171 of *Progress in Molecular Biology and Translational Science*, pages 309–491. Academic Press, 2020.

[161] H. Xiao, K. Rasul, and R. Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*, 2017. arXiv:1708.07747.

[162] L. Xie, S. He, X. Song, X. Bo, and Z. Zhang. Deep learning-based transcriptome data classification for drug-target interaction prediction. *BMC Genomics*, 19, 2018.

[163] Y. Xu, L. Zhang, B. Du, and L. Zhang. Hyperspectral anomaly detection based on machine learning: An overview. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 15:3351–3364, 2022.

[164] Z. Yang and Z. Liu. The risk prediction of alzheimer's disease based on the deep learning model of brain 18f-fdg positron emission tomography. *Saudi Journal of Biological Sciences*, 27(2):659 – 665, 2020.

[165] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[166] G. Zadora and A. Menżyk. In the pursuit of the holy grail of forensic science – spectroscopic studies on the estimation of time since deposition of bloodstains. *TrAC Trends in Analytical Chemistry*, 105:137–165, 04 2018.

[167] M. D. Zeiler. *ADADELTA: An Adaptive Learning Rate Method*, 2012. arXiv:1212.5701.

[168] H. Zhang, Y. Li, Y. Zhang, and Q. Shen. Spectral-spatial classification of hyperspectral imagery using a dual-channel convolutional neural network. *Remote Sensing Letters*, 8:438 – 447, 2017.

[169] L. Zhang, E. Y. Lam, and J. Ke. Temporal compressive imaging reconstruction based on a 3d-cnn network. *Opt. Express*, 30(3):3577–3591, January 2022.

[170] L. Zhang, S. Wang, and B. Liu. Deep learning for sentiment analysis: A survey. *WIREs Data Mining and Knowledge Discovery*, 8(4):e1253, 2018.

[171] K. Zhao, T. Matsukawa, and E. Suzuki. Retraining: A simple way to improve the ensemble accuracy of deep neural networks for image classification. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 860–867, Aug 2018.

[172] Z. Zhao and B. sun. Hyperspectral anomaly detection via memory-augmented autoencoders. *CAAI Transactions on Intelligence Technology*, pages 1–14.

[173] Y. T. Zhou and R. Chellappa. Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, pages 71–78, Vol.2, 1988.

[174] F. Zhu. *Hyperspectral Unmixing: Ground Truth Labeling, Datasets, Benchmark Performances and Survey*, 2017. arXiv:1708.05125.

[175] F. Zhu, Y. Wang, S. Xiang, B. Fan, and C. Pan. Structured sparse method for hyperspectral unmixing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 88:101–118, 2014.

[176] W. G. Zijlstra, A. Buursma, and W. P. Meeuwsen-van der Roest. Absorption spectra of human fetal and adult oxyhemoglobin, de-oxyhemoglobin, carboxyhemoglobin, and methemoglobin. *Clinical Chemistry*, 37(9):1633–1638, 09 1991.

[177] B. Zoph and Q. V. Le. *Neural Architecture Search with Reinforcement Learning*, 2017. arXiv:1611.01578.