Master's Programme in Computer, Communication and Information Sciences – Machine Learning, Data Science and Artificial Intelligence

Variational Auto-Encoders for Satellite Images of Fields

Summarising Sentinel-2 Images of Fields with Variational Auto-Encoders for the Prediction of Crop Loss and Plant Species

---

Maximilian Proll

Aalto University

MASTER'S THESIS

Aalto University
MASTER'S THESIS 2019

Variational Auto-Encoders for Satellite Images of Fields

Summarising Sentinel-2 Images of Fields with Variational Auto-Encoders for the Prediction of Crop Loss and Plant Species

Maximilian Proll

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.
Otaniemi, 20 May 2019

Supervisor: Pekka Marttinen, Assistant Professor
Advisor: Santosh Hiremath, Postdoctoral Researcher

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences – Machine Learning, Data Science and Artificial Intelligence

Author
Maximilian Proll

Title
Variational Auto-Encoders for Satellite Images of Fields

School School of Science

Master's programme Computer, Communication and Information Sciences

Major Machine Learning, Data Science and Artificial Intelligence **Code** SCI3044

Supervisor Pekka Marttinen, Assistant Professor

Advisor Santosh Hiremath, Postdoctoral Researcher

Level Master's thesis **Date** 20 May 2019 **Pages** 56 **Language** English

Abstract

This thesis is situated at the overlap of probabilistic machine learning and remote sensing as it analyses the application of variational auto-encoders to satellite images of fields with the final objective of image classification. The rising availability of high-resolution satellite images of fields increases the need for compressing the images in order to keep maintenance and inference of machine learning models on a feasible and cost-efficient scale. Machine learning, in general, has proven to offer auspicious methods for summarising high-dimensional data into a lower-dimensional representation. Variational auto-encoders are a modern and advanced representation learning algorithm and are the topic of research in this thesis. An extensive hyperparameter search for the implemented networks is performed. The best architecture is selected and compared against conventional computer vision methods. The research shows that summarising high-resolution satellite images with variational auto-encoders is possible. It will, however, still take a performance hit on the classification tasks in comparison to the conventional computer vision techniques. The findings show the potential that variational auto-encoders offer for image compression but also that the used method needs further refinement in order to beat conventional approaches.

Keywords Probabilistic Machine Learning, Bayesian Deep Generative Models, Variational Inference, Variational Auto-Encoder, Remote Sensing, Agriculture, Satellite Images

Preface

I want to thank Pekka Marttinen, Assistant Professor and Santosh Hiremath, Postdoctoral Researcher for their good guidance. Their door was always open whenever I ran into a trouble spot or had a question about my research or writing. They consistently allowed this paper to be my own work but steered me in the right direction whenever they thought I needed it.

Additionally, I would also like to acknowledge the computational resources provided by the Aalto Science-IT project. (Aalto University Science-IT, 2019)

Finally, I want to express my very profound gratitude to my girlfriend, to my parents and to my sister for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Otaniemi, 20 May 2019

Maximilian Proll

Table of Contents

Abstract ii

Preface iii

Table of Contents iv

List of Figures vii

List of Tables ix

1. Introduction 1

2. Background 3

2.1 Background Related to Applications in the Domain of Agriculture 3

2.2 Background Related to Methods 4

2.2.1 Variational Inference 4

2.2.2 Variational Auto-Encoder 7

2.2.3 Variational Auto-Encoder for Images 9

3. Data 12

3.1 Source 12

3.2 Preparation 13

3.3 Satellite Image Download 13

3.4 Masking the Satellite Images 14

4. Methods **15**
4.1 Classification Task 15
4.2 Feature Extraction with Conventional Computer Vision Methods 15
4.2.1 Histogram of Oriented Gradients 16
4.2.2 Local Binary Pattern 17
4.2.3 Grey-Level Co-Occurrence Matrix 17
4.3 Classification Algorithms 18
4.3.1 Random Forest 18
4.3.2 Support-Vector Machine 19
4.3.3 AdaBoost 19
4.4 Variational Auto-Encoder for Satellite Images 20
4.5 Implementation 21
4.6 Hyperparameter Search 24
4.6.1 Batch Normalisation 24
4.6.2 Number of Convolutional Layers 25
4.6.3 Latent Dimensionality 25
4.6.4 Loss Function 25
4.6.5 In-Field-Loss 27

5. Results **28**
5.1 Hyperparameter Search 28
5.1.1 Findings for the Low-dimensional Variational Auto-Encoders 28
5.1.2 Findings for the High-dimensional Variational Auto-Encoders 35
5.2 Comparison with Conventional Computer Vision Methods 35
5.3 Comparison of Low- and High-dimensional Variational Auto-Encoders 37

6. Discussion **39**

7. Conclusion **42**

References **43**

Appendices

A. Findings for High-dimensional Variational Auto-Encoders 46
A.1 Batch Normalisation 46
A.2 Number of Convolutional Layers 46
A.3 Latent Dimensionality 49
A.4 Loss Function 50
A.5 In-Field-Loss 51
A.6 Summary 51

B. Confusion Matrices for the Comparison with Conventional Computer Vision Methods 53

C. Confusion Matrices for the Comparison of Low- and High-Dimensional Variational Auto-Encoders 55

List of Figures

2.1 The reparameterisation trick by Kingma and Welling (2013). 8

3.1 Examples of masked satellite images. 14

4.1 General architecture of the VAE implemented. 21
4.2 Improvement in training time when leveraging TensorFlows pipelining API `tf.data`. 23

5.1 Results for alternating the inclusion of batch normalisation layers for the low-dimensional VAEs. 29
5.2 Results for increasing the number of convolutional layers for the low-dimensional VAEs. 31
5.3 Results for increasing the dimensionality of the latent representation $\dim(z)$ for the low-dimensional VAEs. . . . 32
5.4 Results for alternating the loss function used in the reconstruction loss for the low-dimensional VAEs. 33
5.5 Results for alternating if the reconstruction is computed only inside the field for the low-dimensional VAEs. 34
5.6 Confusion matrices for the Loss-2D classification for the best conventional algorithm as well as the best low-dimensional VAEs without and with IFL. 37
5.7 Confusion matrices for the Loss-2D classification for the best low- and high-dimensional VAEs without and with IFL. 38

A.1 Results for alternating the inclusion of batch normalisation layers for the high-dimensional VAEs. 47
A.2 Results for increasing the number of convolutional layers for the high-dimensional VAEs. 48
A.3 Results for increasing the dimensionality of the latent representation $\dim(z)$ for the high-dimensional VAEs. . . 49
A.4 Results for alternating the loss function used in the reconstruction loss for the high-dimensional VAEs. 50
A.5 Results for alternating if the reconstruction is computed only inside the field for the high-dimensional VAEs. . . . 51

B.1 Confusion matrices for the Loss-4D classification for the best conventional algorithm as well as the best low-dimensional VAEs without and with IFL. 53
B.2 Confusion matrices for the Plant-5D classification for the best conventional algorithm as well as the best low-dimensional VAEs without and with IFL. 54

C.1 Confusion matrices for the Loss-4D classification for the best low- and high-dimensional VAEs without and with IFL. 55
C.2 Confusion matrices for the Plant-5D classification for the best low- and high-dimensional VAEs without and with IFL. 56

List of Tables

3.1 Definition of the four loss categories. 13
3.2 Definition of the binary loss category. 13
3.3 Names of the five most frequent plant species. 14

4.1 Classification tasks and their characteristics. 15
4.2 Parameter alternations for the VAEs. 21
4.3 Default values for the hyperparameter search. 24

5.1 Optimal hyperparameters for the low-dimensional VAEs. 35
5.2 Performance comparison for the best conventional algorithm and the best low-dimensional VAEs without and with In-Field-Loss. 36
5.3 Performance comparison for the best low- and high-dimensional VAEs without and with IFL. 38

A.1 Optimal hyperparameters for the high-dimensional VAEs. 52

1. Introduction

While the global population is rapidly growing the agricultural area is constant, at best. Thus the productivity and efficiency of those fields need to increase in order to provide enough resources for the growing population.

Machine learning techniques have been applied to a wide variety of unrelated disciplines ranging from forecasting the location and intensity of a flood (Nevo et al., 2019) to improving the state-of-the-art language models (Devlin et al., 2018) or the ability to restore corrupted images without access to clean data (Lehtinen et al., 2018).

In order to develop methods for intelligent plant breeding a large scale machine learning algorithm which predicts the crop for a given field by learning a robust model from the climate, the weather, the soil information, the pheno- and genotypes of the plant species and finally satellite and drone images of the field is proposed.

This intelligent crop model incorporates time-series data both on the weather / climate side as well as the satellite / drone images of the fields. As a first step satellite images for the most prominent five plant species[1] are downloaded for one moment in time[2]. Downloading approximately $180{,}000$ satellite images took over 12 days and occupies over $660\,\mathrm{GB}$. The images were taken by the Sentinel-2 satellite which revisits the same point every 5 days. In the best case, this gives approximately 73 data points in time for one year. It becomes clear that this method gets quickly out of hand if all 58 plant species were included and every available point in time would be downloaded and also field images taken from drones will be included. Not only will maintaining the model and continuously running inference on this amount of data will be costly and quickly run into technical limits of existing hardware but it will make it also nearly impossible to execute this algorithm on anything else than a high-performance computing cluster

[1] Out of 58 different plant species in total.
[2] August 2015

with multiple CPUs and GPUs.

Recent developments in the area of satellites are drastically lowering the prices for heretofore expensive imagery from satellites. For example the Finnish companies RSL, Reaktor and VTT launched together a proof-of-concept satellite in November 2018 that orbits the Earth every 95 minutes (Reaktor Space Lab, 2019). This satellite is capable of frequently taking images in the full spectrum of light, which means, that it can analyse soil nutrients, moisture content and chlorophyll levels in plants. The estimated costs for this satellite are between $ 1 million and $ 2 million, significantly cheaper than similar previous builds. Examples like this prove that in the future satellite images will become even more accessible, which will trigger an increased demand to quickly process them and extract business-critical information – not only in the industry of agriculture.

In order to be able to scale the planned intelligent crop model from analysing one point in time for a selection of plant species to analyse time-series data for all available plant species, a reduction in data file size is needed. This master's thesis proposes and analyses the use of variational auto-encoders (**VAE**s) to project the high-dimensional satellite images into a lower-dimensional latent space which then is used for any kind of prediction task in the intelligent crop model. Performing predictions tasks on the lower-dimensional representation of the satellite images comes short in comparison to performing the same task on features gained by conventional computer vision methods.

2. Background

This chapter gives a brief introduction into related work of applying machine learning methods in the domain of agriculture and additionally sets the theoretical foundation for the mathematical concepts used in this thesis.

2.1 Background Related to Applications in the Domain of Agriculture

Pioneering work regarding the classification of satellite images into different plant species was done by Kussul et al. (2016) and Rebetez et al. (2016).

Kussul et al. improve image classification algorithms by including the information about parcel boundaries that take into account the spatial context. Additionally, they allow one field parcel to contain several different crop types. In their paper, they compare pixel-based and parcel-based approaches to crop classification from multitemporal optical (Landsat-8) and synthetic-aperture radar (SAR) Sentinel-1 imagery and they show that the overall classification accuracy can be increased by including parcel boundaries. Kussul et al. analyse the influence that cloud cover and restored pixels values have on the parcel-based classification and propose to assign each pixel a weight that depends on the number of clouded pixels in a time series of optical images.

Rebetez et al. proposed a hybrid neural network architecture which combines histograms and convolutional layers. The presented network classifies the pixels of aerial high-resolution imagery taken from drones into 23 different classes. The hybrid model performs better than separate histogram-based and a simple convolutional model.

Strictly speaking Rebetez et al. did not use satellite images but rather

aerial imagery taken from drones capable of taking high-resolution images. But the argument remains the same: the authors of both papers use the full resolution of images at a high resolution. In general, the availability of high-resolution images will grow because more modern satellites or drones will take more images with even higher resolution. Also, those models can be extended to cover not only parts of Ukraine (like done by Kussul et al.) but cover the whole Ukraine, the entire continent of Europe or even the entire globe. Scaling up the models as presented in both papers to thousands or even millions of images will significantly slow down the training time and will bring the feasibility to a limit where they are not useable any more.

Machine learning algorithms are not only applied to aerial images of the fields but can also be applied to various other aspects. Yalcin (2017) proposes a model to classify phenological stages of several types of plants based on the visual data captured by cameras mounted on the ground agro-stations. Mohanty et al. (2016) developed a deep convolutional neural network to identify 14 crop species and 26 diseases (or absence thereof) based on a public dataset of about $54{,}000$ close up images of diseased and healthy plant leaves. Dyrmann et al. (2017) and McCool et al. (2017) trained a fully convolutional neural network for automating weed detection in colour images despite heavy leaf occlusion. Bargoti and Underwood (2016) apply the object detection framework, Faster R-CNN, in the context of fruit detection in orchards, including mangoes, almonds and apples.

2.2 Background Related to Methods

This section introduces first the concept of variational inference and then establishes the mathematical concepts behind the implemented VAEs and its specifications when they are applied to images.

2.2.1 Variational Inference

For the following analysis let us consider the set of all observed variables by $\mathbf{X}$. In a fully Bayesian model one normally considers that the model may also have latent variables as well as latent parameters. The set of all latent variables and parameters is denoted by $\mathbf{Z}$. Generally, those models contain some prior belief about the distribution of the latent variables and latent parameters $p(\mathbf{Z})$. Let us also consider that there is a set of

N independent, identically distributed data, for which $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and $\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_N\}$. Bayesian inference on such models defines the joint probability distribution $p(\mathbf{X}, \mathbf{Z})$ and it consists of updating the prior distribution of the latent variables and latent parameters $p(\mathbf{Z})$ after having observed data $\mathbf{X}$ into an updated information over the latent parameters in the form of the posterior distribution $p(\mathbf{Z}|\mathbf{X})$. The central task of probabilistic modelling is to evaluate the posterior distribution $p(\mathbf{Z}|\mathbf{X})$ and find those latent variables $\mathbf{Z}$ that maximise the posterior distribution. (Bishop, 2006)

$$p(\mathbf{Z}|\mathbf{X}) = \frac{p(\mathbf{X}, \mathbf{Z})}{p(\mathbf{X})} = \frac{p(\mathbf{X}|\mathbf{Z})p(\mathbf{Z})}{\int_{\mathbf{Z}} p(\mathbf{X}, \mathbf{Z}) d\mathbf{Z}} \tag{2.1}$$

Equation 2.1 shows how the posterior distribution $p(\mathbf{Z}|\mathbf{X})$ can, in principle, be calculated analytically. But for many applications it is not feasible to evaluate the posterior distribution or to compute expectations with respect to it, due to the required integration in the denominator. This can be caused by a number of different factors: either the dimensionality of the latent space is too high to work with directly or the posterior distribution has a highly complex form for which expectations are not analytically tractable. For the case of continuous variables the integrations may not have closed-form analytical solutions, while at the same time the dimensionality of the space and the complexity of the integrand may prohibit numerical integration. When the latent variables are discrete the integration turns into summing over all possible combinations of hidden variables, and though this summation is theoretically always possible, there are often exponentially many hidden states so that exact calculation becomes extremely expensive. (Bishop, 2006)

One possible solution to overcome the issue of intractable integration is to approximate the posterior distribution $p(\mathbf{Z}|\mathbf{X})$. Approximation schemes can be separated into two classes: stochastic and deterministic approximations.

Stochastic approximations such as Markov chain Monte Carlo can generate exact results given infinite computational resource. This means that in practice stochastic sampling methods are often computationally demanding which limits their application to small-scale problems.

Deterministic approximations on the other hand scale well to large applications. They are based on an analytical approximation to the posterior distribution, hence they can never generate exact results. The strengths and weaknesses of stochastic and deterministic approximations are there-

fore complementary.

In *variational inference* the posterior distribution $p(\mathbf{Z}|\mathbf{X})$ is approximated by a variational distribution $q_\phi(\mathbf{Z})$ described by the parameters ϕ:

$$q_\phi(\mathbf{Z}) \approx p(\mathbf{Z}|\mathbf{X})$$

The motivation is that the variational distribution $q_\phi(\mathbf{Z})$ belongs to a family of distributions of simpler form than the posterior distribution $p(\mathbf{Z}|\mathbf{X})$ but it shall be selected with the intention that variational distribution is as similar as possible to the true posterior distribution. This similarity is measured by the Kullback-Leibler divergence (**KL** divergence) $D_{KL}\left(q_\phi(\mathbf{Z})\|p(\mathbf{Z}|\mathbf{X})\right)$ which quantifies the difference of the variational distribution $q_\phi(\mathbf{Z})$ to the posterior distribution $p(\mathbf{Z}|\mathbf{X})$.

The decomposition of the log marginal probability $\ln p(\mathbf{X})$ into the KL divergence $D_{KL}\left(q_\phi(\mathbf{Z})\|p(\mathbf{Z}|\mathbf{X})\right)$ and the evidence lower bound (**ELBO**)[1] $\mathcal{L}(q)$ as displayed in Equation 2.2 holds for any choice of $q_\phi(\mathbf{Z})$.

$$\ln p(\mathbf{X}) = \mathcal{L}(q) + D_{KL}(q\|p), \tag{2.2}$$

where:

$$\mathcal{L}(q) = \int q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{X},\mathbf{Z})}{q(\mathbf{Z})} \right\} \mathrm{d}\mathbf{Z} \quad \text{and} \tag{2.3}$$

$$D_{KL}(q\|p) = -\int q(\mathbf{Z}) \ln \left\{ \frac{p(\mathbf{Z}|\mathbf{X})}{q(\mathbf{Z})} \right\} \mathrm{d}\mathbf{Z}. \tag{2.4}$$

Since the log marginal probability $\ln p(\mathbf{X})$ is constant with respect to q minimising the KL divergence $D_{KL}\left(q_\phi(\mathbf{Z})\|p(\mathbf{Z}|\mathbf{X})\right)$ with respect to q is equivalent to maximising the variational lower bound $\mathcal{L}(q)$. This characteristic is crucial to variational inference because we cannot simply minimise the KL divergence $D_{KL}\left(q_\phi(\mathbf{Z})\|p(\mathbf{Z}|\mathbf{X})\right)$ since it requires evaluating the intractable posterior $p(\mathbf{Z}|\mathbf{X})$. Choosing the variational distribution $q_\phi(\mathbf{Z})$ appropriately makes both the computation and the maximisation of the ELBO $\mathcal{L}(q)$ tractable.

If the variational distribution $q_\phi(\mathbf{Z})$ is allowed to have any form then the ELBO $\mathcal{L}(q)$ is at its maximum when the KL divergence vanishes, in other words when $D_{KL}\left(q_\phi(\mathbf{Z})\|p(\mathbf{Z}|\mathbf{X})\right) = 0$, which happens only when $q(\mathbf{Z}) = p(\mathbf{Z}|\mathbf{X})$. But since working with the true posterior distribution $p(\mathbf{Z}|\mathbf{X})$ is intractable, it is assumed that the variational distribution $q_\phi(\mathbf{Z})$ belongs

[1] also called variational lower bound

to a restricted family of distributions. Minimising the KL divergence seeks this particular member which best matches true posterior distribution.

2.2.2 Variational Auto-Encoder

In a plain vanilla auto-encoder (**AE**) the encoder converts an input into a lower dimensional representation (*aka.* the code), which the decoder converts back to the original input.

AEs face one drawback in terms of it not being a generative model. The latent space, where the encoded representations of the input lie, is typically not continuous and thus does not allow an easy interpolation.

In terms of generation, an AE is limited to replicate the original input, but cannot generate variations of an input image from a continuous latent space. A sample from a region of discontinuities in the latent space (e.g. gaps between clusters) will be decoded to an unrealistic output because the decoder has never learned how to treat samples from this region of the latent space.

VAEs were proposed by Kingma and Welling (2013) and Jimenez Rezende et al. (2014) and overcome this drawback. VAEs in contrast to normal AEs have a latent space, that is by design continuous and thus allows easy random sampling and interpolation. This property is especially useful for generative modelling, as the decoder of a VAE will know how to interpret regions of discontinuities in the latent space.

The key difference behind any VAE in comparison to normal AEs is that they are trained by maximizing the variational lower bound $\mathcal{L}(q)$ given the data $\mathbf{X}$. The variational lower bound $\mathcal{L}(q)$ defined in Equation 2.3 can be expanded as follows:

$$\mathcal{L}(q) = \mathbb{E}_{\mathbf{Z}\sim q(\mathbf{Z})}\left[\ln p(\mathbf{Z}, \mathbf{X})\right] + \mathcal{H}(q(\mathbf{Z}|\mathbf{X})) \tag{2.5}$$

$$= \mathbb{E}_{\mathbf{Z}\sim q(\mathbf{Z})}\left[\ln p(\mathbf{X}|\mathbf{Z})\right] - D_{\mathrm{KL}}(q(\mathbf{Z})\|p(\mathbf{Z})) \tag{2.6}$$

In equation 2.5 the first term is the joint log-likelihood of the visible and hidden variables under the approximate posterior over the latent variables. The second term describes the entropy of the approximate posterior. Generally, this entropy term encourages the variational posterior to place high probability mass on many $\mathbf{Z}$ values that could have generated $\mathbf{X}$, rather than collapsing to a single point estimate of the most likely value.

In equation 2.6 the first term formulates the reconstruction log-likelihood.

The second term tries to make the approximate posterior distribution $q(\mathbf{Z})$ and the model prior $p(\mathbf{Z})$ approach each other. (Goodfellow et al., 2016)

A continuous latent space is achieved by altering the nature of an AE in a subtle but not insignificant way. Instead of outputting one deterministic latent vector $\mathbf{Z}$ the encoder of a VAE will output **two** deterministic latent vectors: a vector of means μ and another vector of standard deviations σ. Both deterministic vectors form the parameters of a distribution $\phi(\cdot)$ from which a random vector $\mathbf{Z}$ is being sampled: $\mathbf{Z} \sim \phi(\mu, \sigma)$. This sampled random vector $\mathbf{Z}$ is then passed on to the decoder like in a normal AE.

This stochastic generation means that the same input will result in the same mean and standard deviations but due to the random sampling, the actual encoding will vary on every single draw. As the encodings are generated at random from anywhere inside the probability distribution parametrised by the means μ and the standard deviations σ, the decoder learns that not only is a single point in latent space referring to a sample of that class, but all nearby points refer to the same as well. This allows the decoder to not just decode single, specific encoding in the latent space – hence leaving the decodable latent space discontinuous – but one that slightly varies too, as the decoder is exposed to a range of variations of the encoding of the same input during training.

A reparameterisation trick is performed in order to allow the backpropagation to work all the way from the final output through the code to the initial input. It was proposed by Kingma and Welling (2013) and is visualised in Figure 2.1. According to the reparameterisation trick one randomly samples ϵ from a unit Gaussian and scales it by the latent distribution's standard deviation σ and then shifts the product by the latent distribution's mean μ. With this reparameterisation, we can now optimize the parameters of the distribution while still maintaining the ability to randomly sample from that distribution.

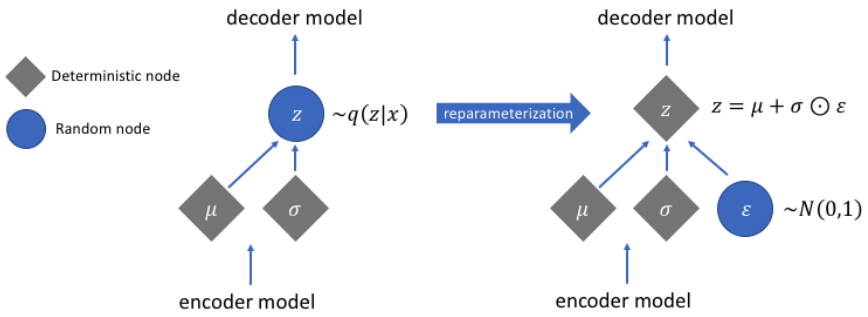


Figure 2.1. The reparameterisation trick by Kingma and Welling (2013). Image: Jordan (2018)

One convenient property of the VAE is that simultaneously training a parametric encoder in combination with the generator network forces the model to learn a predictable coordinate system that the encoder can capture. This makes it an excellent manifold learning algorithm (Goodfellow et al., 2016).

2.2.3 Variational Auto-Encoder for Images

hen implementing a VAE for images it comes naturally to apply convolutional and pooling layers in the encoder, because both the convolution as an operation and the pooling respect and maintain the spatial information of the input (2D images or higher dimensional tensors), while simultaneously achieving an effective and desired reduction in the dimensionality.

But which type of operation / function / kernel to use for the decoder is not as straightforward. The decoder receives typically a low-dimensional input and transforms it with the goal of restoring the original input of the (V)AE, in this case, an image. Therefore the operation is supposed to work similar to a convolution but performing a transformation going in the opposite direction: from something that has the shape of the output of a convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution. This way of thinking probably motivated Zeiler et al. (2010) to use the name **de**convolution.

Dumoulin and Visin (2016) motivate that the naming of such function should rather be *transposed convolution* and their reasoning is based on the mathematics behind a convolution and the function, which works in the opposite direction.

They propose to unroll the input and the output of the convolution into vectors (from left to right, top to bottom) because now the case looks like a simple fully-connected layer. The convolution is parametrised by a kernel $\mathbf{w}$, which in analogy to the a fully-connected layer can now be used to fill the sparse matrix $\mathbf{C}$ where the non-zero elements are the elements $w_{i,j}$ of the kernel $\mathbf{w}$. The matrix $\mathbf{C}$ is filled in such a way, that the spatial characteristics of the convolutional operation are reflected. The example below demonstrates the case, where the convolution of the kernel $\mathbf{w}_{2\times2}$ shall be applied to the input matrix $\mathbf{X}_{3\times3}$ to form the output matrix $\mathbf{Y}_{2\times2}$. In order to keep the example simple, the dimensions are chosen as stated

in the indices of the matrices.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \mathbf{w} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \mathbf{Y} = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix}$$

Unrolling both the input matrix $\mathbf{X}_{3\times3}$ as well as the output matrix $\mathbf{Y}_{2\times2}$ gives:

$$\mathbf{X}_{flat} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{21} & x_{22} & x_{23} & x_{31} & x_{32} & x_{33} \end{bmatrix}^\intercal$$

$$\mathbf{Y}_{flat} = \begin{bmatrix} y_{11} & y_{12} & y_{21} & y_{22} \end{bmatrix}^\intercal$$

Now depending on how the convolution is specified in terms of padding, stride and number of filers the specifics may vary, but the general idea remains the same. Assuming no padding, only *valid* application and a stride of 1 the first entry of the output matrix is defined by:

$$y_{11} = w_{11} \cdot x_{11} + w_{12} \cdot x_{12} + w_{21} \cdot x_{21} + w_{22} \cdot x_{22}$$

Thus those values of $w_{i,j}$ will be used at the corresponding place of $\mathbf{C}$ in order to reflect the same multiplication. The first row of $\mathbf{C}$ which corresponds to the first entry of $\mathbf{Y}$ will look like this:

$$\mathbf{C}_{1:} = \begin{bmatrix} w_{11} & w_{12} & 0 & w_{21} & w_{22} & 0 & 0 & 0 & 0 \end{bmatrix}$$

This procedure demonstrates how the kernel $\mathbf{w}$ and the specifications of the convolution (padding, stride and application type) define the entries of the matrix $\mathbf{C}$. This matrix $\mathbf{C}$ defines then the following matrix multiplication, which essentially formalises the convolution in a manner that looks like a fully connected layer:

$$\mathbf{Y}_{flat} = \mathbf{C} \cdot \mathbf{X}_{flat}$$

where the output $\mathbf{Y}_{flat}$ just needs to be reshaped accordingly.

Now the convolution takes the input matrix flattened as a high-dimensional vector and produces a low-dimensional vector that is later reshaped into the desired output matrix shape.

In this flattened representation, the backward pass of a neural network is formalised by using the transposed matrix $\mathbf{C}^\intercal$. Any error in the output is backpropagated by multiplying its loss with $\mathbf{C}^\intercal$. The backpropaga-

tion transforms a low-dimensional vector as input and returns a high-dimensional vector as output, and its connectivity pattern is aligned with $\mathbf{C}$ by construction.

Since the kernel $\mathbf{w}$ and the specifications of the convolution define the entries of the matrix $\mathbf{C}$ used for the forward pass, they also defined the matrix $\mathbf{C}^\intercal$ used for the backpropagation.

Dumoulin and Visin also make a strong case against using the term *deconvolution*, because a *deconvolution* is mathematically defined as the inverse of a convolution, which is different from a transposed convolution.

Pu et al. have implemented a VAE for not only analysing images but also to generate either a caption with a recurrent neural network or a label with a Bayesian support vector machine.

3. Data

This chapter gives an overview of the dataset used and details the whole data processing pipeline from the source and any required pre-processing to the download of the satellite images. Lastly, this chapter explains how the images were masked to focus fully on the actual area of the field and not its surroundings.

3.1 Source

Maaseutuviraston (**MAVI**), the Agency for Rural Affairs, is the data owner of the location and the crop loss (**CL**) information of the fields in Finland. MAVI is responsible for the use of agricultural aid and rural development funds of the European Union in Finland. MAVI provided a data set which was split into three zip files each containing a shapefile for the years 1997-2004, 2005-2014 and 2015-2017[1]. The following information is contained in the attribute table:

- year
- field parcel
- identifier
- plant & plant code
- variety & variety code
- property area
- full crop loss
- partial crop loss

The given data set has the above-mentioned information on $465{,}595$ individual fields belonging in total to 58 different plant species.

[1] file names: *raportti_1997_2006.zip, raportti_2007_2014.zip, raportti_2015_2017.zip*

3.2 Preparation

The bounding box for each field is required in order to download the satellite images for the required location. Due to the missing information of the longitude and latitude of the bottom left and the top right corner surrounding the field it was added manually in QGIS and then finally exported as a CSV.

3.3 Satellite Image Download

The bounding box created in section 3.2 is used as an input source for the SentinelHub API.

The download of the satellite images was initially performed for all $158{,}304$ fields of the most frequent plant species *Rehuohra* (feed barley). During this download, all available colour channels (13 in total) were downloaded for a specified resolution of 512 by 512 pixels. This download took approximately 8 days. From those $158{,}304$ fields a balanced subset of in total 7500 fields was created. This subset was balanced in terms of the presence of the four loss categories defined in Table 3.1. Additional to those four loss categories a binary loss category is defined by merging all types of CL greater than zero, which is similarly defined in Table 3.2.

		Partial Crop Loss	
		$= 0$	> 0
Full Crop Loss	$= 0$	no loss	only partial
	> 0	only full	full and partial

Table 3.1. Definition of the four loss categories.

		Partial Crop Loss	
		$= 0$	> 0
Full Crop Loss	$= 0$	no loss	some loss
	> 0	some loss	some loss

Table 3.2. Definition of the binary loss category.

In a secondary step $29{,}768$ more images for the four next most frequent plant species *Kaura* (Oats), *Mallasohra* (Malting Barley) *Kevätvehnä* (Spring Wheat) and *Kevätrypsi* (Spring Rapeseed) were downloaded in the same resolution and with all 13 colour channels. For those four plants a data set was created in which the number of fields with *some loss* was equal to the number of fields with *no loss*, which totals to $11{,}538$ images of

individual fields. The five most frequent plant species are summarised in Table 3.3.

	Plant Species (in Finnish)	Plant Species (in English)
1^{st}	Rehuohra	Feed Barley
2^{nd}	Kaura	Oats
3^{rd}	Mallasohra	Malting Barley
4^{th}	Kevätvehnä	Spring Wheat
5^{th}	Kevätrypsi	Spring Rapeseed

Table 3.3. Names of the five most frequent plant species in Finnish and English.

3.4 Masking the Satellite Images

Since the bounding box is a rectangle but the shape of the field can be, and most often is, different from a rectangle, the downloaded image of the field captures some area which does not belong to the field. Since it is not intended that a machine learning algorithm picks up on any kind of information outside of the field, the surroundings of the field will be simply set to zero. This process is also known as *image masking*.

Two masked example images of fields are displayed in Figure 3.1.

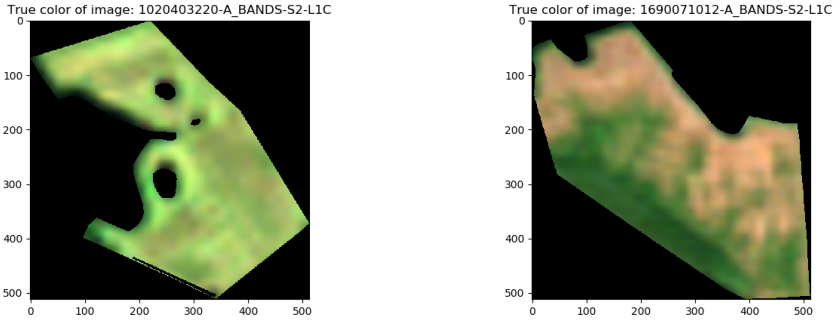


Figure 3.1. Examples of masked satellite images.

4. Methods

First, this chapter defines the classification task that both the VAE and conventional computer visions methods are performing. It is followed by a summary of the conventional computer visions methods used for extracting the features of the satellite images and the algorithms used for classifying them according to the given classification task. Then it provides specific details on the architecture of the VAEs. The chapter continues with a description of the workflow how the implementation was adapted from running the machine learning algorithms locally to running them on a high-performance computing cluster and finishes with an explanation of the hyperparameter search performed.

4.1 Classification Task

The three classification tasks to be performed by the VAE and the conventional computer visions methods are summarised in Table 4.1.

Classification of the		Abbreviation	Number of Classes	Definition in
(i)	four loss categories	Loss-4D	4	Table 3.1
(ii)	binary loss categories	Loss-2D	2	Table 3.2
(iii)	plant species	Plant-5D	5	Table 3.3

Table 4.1. Classification tasks and their characteristics.

4.2 Feature Extraction with Conventional Computer Vision Methods

This section is dedicated to giving a brief summary of the more conventional computer vision (**CV**) techniques used as comparison methods in this thesis. *Conventional* computer vision approaches are those that do not rely on

more modern ideas like deep learning. The referenced techniques are: Histogram of Oriented Gradients, Local Binary Pattern and Grey-Level Co-Occurrence Matrix. All of them follow the main goal of traditional computer vision methods, that is to reduce the complexity of an image by extracting important features. How the importance of certain features is evaluated and computed is the key distinction property of traditional computer vision techniques.

The above-mentioned feature extractors were designed to be applied to rather low-dimensional single-channel images. They do not perform as good and lose some of its advantages – namely speed and simplicity – when they are applied to high-dimensional multi-channel images. This is why the conventional computer vision methods are applied exclusively to single-channel images with a lower resolution of 64×64. The channel used in those images is the normalized difference vegetation index[1] (**NDVI**), which was computed based on the 13 original channels of the image.

4.2.1 Histogram of Oriented Gradients

The histogram of oriented gradients (**HOG**) is a feature descriptor proposed by Dalal and Triggs (2005). This method suggests the idea to summarise an image by the distribution (histogram) of the directions of its gradients (oriented gradients). Derivatives in x and y direction are an informative measurement because regions with abrupt changes in the intensity of gradients are generally occurring at corners and edges which help to find, describe, and differentiate objects.

The horizontal (g_x) and vertical (g_y) gradients are computed by applying the following kernels:

$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \qquad \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}.$$

With the horizontal and vertical component the intensity g and (unsigned) orientation θ of the gradient can be computed with the following equations:

$$g = \sqrt{g_x^2 + g_y^2}, \qquad \theta = \arctan \frac{g_y}{g_x}.$$

For a smaller sub-cell of the image the magnitudes g and orientations

[1] The definition of NDVI channel is: $\mathrm{NDVI} = \frac{\mathrm{NIR}-\mathrm{Red}}{\mathrm{NIR}+\mathrm{Red}}$, where Red and NIR stand for the red respectively near-infrared reflectance measurements.

θ are summarised in a histogram with some discrete number of bins n[2] corresponding to angles from $0°$ to $180°$. The intensity g of the gradient is associated with one or two bins according to its orientation θ.

Such histograms are sensitive to changes in light intensity which is why the histograms are normalised by dividing each element by the magnitude of this vector. This normalisation creates an invariance to changes in light intensities and is computed over larger blocks of 2x2 cells.

Finally, those blocks are moved over the entire image with an overlap of 50%, which results in the final feature representation of the image that is fed into a classification algorithm.

4.2.2 Local Binary Pattern

Local binary pattern (**LBP**) is a visual descriptor proposed by Ojala et al. (1996). It compares each pixel to its neighbouring pixels and saves the result as a binary number, which is then summarised in a histogram and represents the features of the input image. LBP has become a popular method for various applications where images are analysed and classified in real-time due to its computational simplicity. One of the most important characteristics of LBP is its insensitivity to changes in light intensities. LBPs are able to recognise certain primitive forms of an object such as spots, flats, end of lines, edges and corners. All of those forms are crucial pieces for localising objects.

To transform the input image to a LBP feature vector, the input image is divided into cells. Each pixel in a cell is subsequently compared to its 8 neighbouring pixels always following the same circular orientation – clockwise or counter-clockwise. For each comparison where the centre pixel's value is greater than the neighbour's value, a 0 is saved. Otherwise, a 1 is saved. In total, this results in an 8-digit binary number which is then converted to a decimal number. One histogram is computed for all decimal LBP numbers in a cell, which additionally can be normalised. Finally, the LBP feature vector is a result of concatenating the histograms for all cells.

4.2.3 Grey-Level Co-Occurrence Matrix

A grey-level co-occurrence matrix (**GLCM**) is a matrix that describes the texture of an input image by calculating how often pairs of pixels with a certain value in a defined spatial order occur.

[2]Most implementations of HOG compute a histogram for a 8×8 cell with $n = 9$ bins.

The transformation from an input image to a GLCM can be done as follows: the spatial offset $\Delta x, \Delta y$ has to be defined first, as it defines the desired relationship between the pixel pairs. The number of unique grey levels p defines a $p \times p$ dimensional GLCM. This could be anything between a binary value and a 32-bit colour, which would create a GLCM as big as $2^{32} \times 2^{32}$. In order to avoid such big matrices, the grey level is normally scaled or binned into 8 intensity levels. Each element (i, j) in the GLCM simply counts how often the pixel with value i occurred in the specified spatial relationship to a pixel with value j in the input image.

Normally co-occurrence matrices are typically large and sparse, which is why in the context of texture analysis statistical measure such as contrast, correlation, energy and homogeneity are extracted from the GLCM. Either the GLCM is used without further processing as a feature representation of the input image or the aforementioned statistics are used.

4.3 Classification Algorithms

This section is providing a short introduction to the supervised machine learning algorithms used: Random Forest, Support-Vector Machine and AdaBoost. They use the extracted features from the conventional computer vision methods explained in section 4.2 and ultimately perform the threefold classification task as specified in section 4.1.

4.3.1 Random Forest

Random forests are an ensemble method proposed by Ho (1995), which is used for both classification and regression problems in machine learning. It defines and trains multiple de-correlated decisions trees simultaneously and combines them by reporting that label, which is reported most often by the individual decision trees (when used for classification) or it averages the reported value (when used for regression).

Generally, random forests are easy and fast to train and its hyperparameter are optimised with little effort in comparison to other simple algorithms which makes them a popular go-to method for a straightforward model baseline.

The combination of multiple decisions trees corrects the inherent flaw of individual decision trees of overfitting to the training data. A random forest has the same bias as the individual trees but the variance is reduced (Hastie et al., 2009).

Breiman (2001) extended the initial algorithm by modifying the idea of bagging and how the input features are selected.

4.3.2 Support-Vector Machine

Support-vector machines (**SVM**) were proposed by Cortes and Vapnik (1995) and are the generalisation of support-vector classifiers (**SVC**) which intend to find a linear hyperplane that maximises the margin between the hyperplane and the nearest point of each group. SVCs can be made more powerful, e.g. perform non-linear classification, by applying the so-called *kernel trick*. The kernel trick projects the input data into a high-dimensional space, where then the SVCs try to find the above-mentioned maximum-margin hyperplane. SVM pick up on the same idea, where the dimension of the enlarged space is allowed to get very large, infinite in some cases. (Hastie et al., 2009)

SMVs are either used in a supervised setting where they solve classification or regression tasks, but they can also be used in an unsupervised manner where they categorise unlabelled data into clusters. (Ben-Hur et al., 2001)

4.3.3 AdaBoost

Boosting methods follow the general idea to combine the result of many so-called *weak learners* to a powerful group by merging the individual results of the weak learners to a weighted sum which defines the final output of the boosted classifier. Boosting methods were originally developed for classifications tasks but can be extended to regression problems as well.

Adaptive Boosting (**AdaBoost**) was proposed by Freund and Schapire (1999) and is one of the most prominent members of the class of boosting methods. AdaBoost is *adaptive* in the sense that at each boosting step the weight associated with a subsequent weak classifier is optimised to concentrate more on previously misclassified samples. Thus observations that are difficult to classify correctly receive step by step increasing importance.

The weak learners in AdaBoost are often decisions trees where subsequent classifiers focus on increasingly difficult to classify samples.

4.4 Variational Auto-Encoder for Satellite Images

For the sake of comparability to the conventional computer vision methods a VAE is implemented using the very same input: the single-channel NDVI images with a lower resolution of 64×64. During this document this implementation will be referenced as *low-dimensional VAE* and the VAE, which uses to full 13-channel image with its entire resolution of 512×512 will be referenced as *high-dimensional VAE*.

The implemented VAEs that analyse the satellite images with the goal of finding a low-dimensional representation have the following general set up, which is additionally visualised in Figure 4.1:

- The satellite images with dimension either 512 x 512 and 13 colour channels or 64 x 64 and the computed NDVI colour channel are passed to the encoder.
- The encoder consists of a varying number of alternating layers of convolution and batch normalisation with different specifics.
- The encoder leads to the two deterministic variables: the latent distribution's mean μ and its standard deviation σ[3] of the assumed distribution.
- Figure 2.1 displays the reparameterisation trick, which is achieved by randomly sampling a noise term ϵ from a unit Gaussian distribution and scaling it by the standard deviation σ and then shifting the product by the mean μ : $z = \mu + \sigma \cdot \epsilon$[4].
- The sampled z is then passed through the decoder.
- The decoder is formed by reversing the order of the encoder, hence there are again a varying number of alternating layers of transposed convolutions and batch normalisations with different specifics.
- The decoder outputs the reconstructed image with the same dimension as the input (either 512 x 512 and 13 colour channels or 64 x 64 and the NDVI colour channel).

The specific alternations of certain parameters of the encoder and thus also the decoder are summarised in Table 4.2. Those parameters are treated as hyperparameters which are analysed and optimised in section 4.5.

[3]Most implementations learn the logarithm of the variance $\ln \sigma^2$ instead, because it facilitates the calculation of the KL divergence term.
[4]When the logarithm of the variance $\ln \sigma^2$ is learned the reparameterisation trick becomes: $z = \mu + e^{\frac{1}{2} \cdot \ln \sigma^2} \cdot \epsilon$.

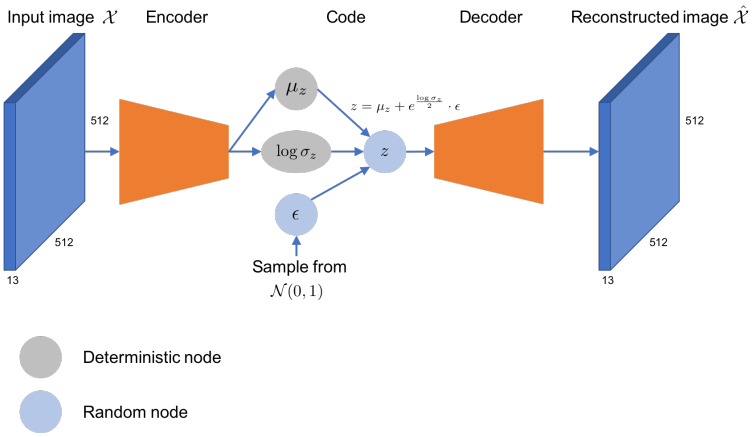


Figure 4.1. General architecture of the VAE implemented.

Network Parameter	Alternation Range
Usage of Batch Normalisation Layers	`boolean`: `True` or `False`
Number of Convolutional Layers	`int`: from $3 \dots 20$
Dimensionality of Latent Representation	`int`: $2^1, 2^2, 2^3, \dots 2^{10}$
Loss Function for the Reconstruction Loss	`MSE` or `Cross-entropy`
Usage of In-Field-Loss (explained later in section 4.6.5)	`boolean`: `True` or `False`

Table 4.2. Parameter alternations for the implemented VAEs.

The VAEs are implemented by using the Google's machine learning framework TensorFlow (by Abadi et al.).

The objective of the VAEs is to train them by finding a low-dimensional representation z that is well suited for reconstructing the original image. Once this low-dimensional representation z is found, a fully-connected network is used as a classifier for the classification tasks specified in Table 4.1.

4.5 Implementation

When implementing any kind of machine learning algorithm in general and the VAEs detailed in section 4.4 in particular an intuitive approach is to start it locally on a workstation or a personal laptop. This brings several advantages from the independence of network availability, easier control over software packages and its versions, full availability of computing resources without the need of previously requesting them to using integrated development environments with its own advantages on debugging.

Thus the first implementation of the network was done on the personal laptop where both data and resources were available locally. But since the

full data set occupies over 660 GB of disk space, the network was only run on a small subset of the data prioritising running speed and saving disk space over the completeness of the analysis. While using a small subset of the data it was no problem to load all the images including their labels into RAM. But it was clear that once more images shall be fed into the network, additional work has to be done since the RAM is not large enough to hold all images simultaneously. Also, neither the workstation nor the personal laptop have a high-performance CPU and GPU.

Aalto University has its own high-performance computing cluster, called Triton, which has a wide variety of different computing nodes offering multiple combinations of high-speed CPUs, large RAM and powerful GPUs.

The first naive approach is to simply run the very same model – written for both local data and resources availability – on Triton. The only difference is that the data is now located on servers that need to be accessed from the computing node. Running those algorithms with many powerful CPUs and GPUs did not result in the expected speed increase and monitoring the GPU utilisation showed the reason behind it: the GPU was idling nearly all the time with infrequent short peaks of high utilisation. This is a clear sign that the input/output of the programme is causing an issue. The speed of an algorithm is often directly linked to CPU and GPU speed, but this is missing a crucial factor: the network speed, because it essentially limits how fast the data can get to the CPU. Thus input/output (IO) is the true bottleneck and must be improved.

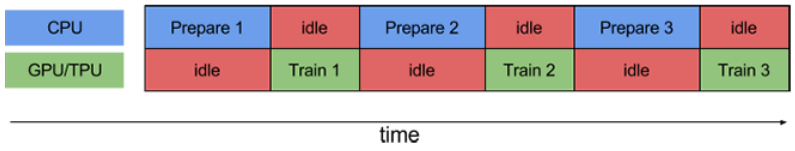
The team behind TensorFlow has summarised how a typical training input pipeline can be seen as an Extract-Transform-Load (**ETL**) process:

- **Extract**:
  Read data from storage – either local (e.g. HDD or SSD) or remote (e.g. GCS or HDFS).
- **Transform**:
  Use CPU cores to parse and perform preprocessing operations on the data such as data augmentation, shuffling, and batching.
- **Load**:
  Load the transformed data onto the accelerator device (e.g. GPUs or TPUs) that executes the machine learning model.

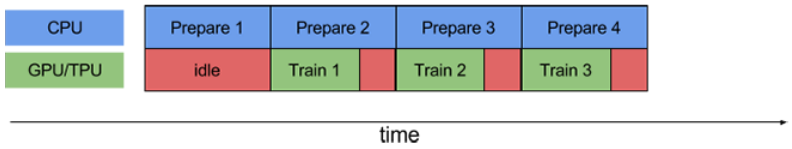
As of right now the model was implemented for local data storage and the ETL pipeline was designed in a rather naive synchronous way: The GPU has to wait until the CPUs have read and preprocessed one batch of images and passed it onto the GPU. When the GPU is doing all the heavy

lifting the CPU is sitting idle. Thus the time needed for a forward and backward pass for one batch is the sum of both CPU pre-processing time and the GPU training time.

TensorFlow offers a pipelining API that is called `tf.data`. It is designed to effectively utilize the CPU and thus optimise all steps of the ETL process. As a first step, the data has to be saved as TensorFlows native format – as `TFRecord` files. The principal change in the `tf.data` API is to overlap the preprocessing and model execution of a training step. When the GPU is performing training step N, the CPU can already start the data preparation for step $N + 1$. Now the time needed for a forward and backward pass for one batch is the maximum of the training and data preparation time. An intuitive visualisation of the suboptimal performance related to the IO issue and the improved setup is displayed in Figure 4.2.



(a) Without pipelining.



(b) With pipelining.

Figure 4.2. Improvement in training time when leveraging TensorFlows pipelining API `tf.data`. Fig. 4.2a shows how much time is needed when no pipelining is used. Consequently both CPU and the GPU sit idle much of the time. Fig. 4.2b shows the significant effect the usage of overlapped preprocessing and model execution have on the training time. The total time one of the devices is idle is drastically diminished.
Image: www.tensorflow.org/guide/performance/datasets

`tf.data` offers substantially more features than pipelining. The input data transformation is almost always independent between the samples. `tf.data` uses this characteristic and enables the parallelisation of the data preprocessing to multiple threads on a single CPU.

The `tf.data` API provides additionally dedicated functions for loading data from remote storage. Those functions tackle those issues that are related to locally and remotely stored data:

- **Time-to-first-byte**:
  The reading time of the first byte is significantly larger for a file from remote storage than one from local storage.

- **Read throughput**:
  Remote storage typically offers large aggregate bandwidth which is often not fully utilised when reading a single file.

Leveraging the vast possibilities that the `tf.data` API offers significantly improved training time and made it possible to achieve a high utilisation rate on both CPU and GPU.

4.6 Hyperparameter Search

The implemented VAEs are outlined in Section 4.4 where also the different hyperparameter options are summarised in Table 4.2. In order to find the best combination of hyperparameters, one hyperparameter was altered while the remaining ones were left equal (ceteris paribus). The default settings for the hyperparameters is shown in Table 4.3.

Network Parameter	Default Value
Usage of Batch Normalisation Layers	`False`
Number of Convolutional Layers	3
Dimensionality of Latent Representation	1024
Usage of Loss Function for the Reconstruction Loss	`MSE`

Table 4.3. Default values for the hyperparameter search.

4.6.1 Batch Normalisation

Normalising the distribution in each batch was proposed by Ioffe and Szegedy (2015) and showed promising results because the convergence speed could be increased.

Mathematically speaking the normalisation shifts each scalar feature independently to have the mean of zero and the variance of 1. For a layer with k-dimensional input $\mathrm{x} = \left(x^{(1)} \dots x^{(k)}\right)$ each dimension is normalised according to the following equation:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right]}}$$

where the expectation and variance are computed over the training data set. A batch normalisation layer is almost always displayed as $\mathrm{BN}(x)$, which encodes the normalisation across each dimension as described above.

The batch normalisation layer is applied immediately before any non-

linear activation function since the motivation behind batch normalisation is the prevention of saturation of the non-linear activation functions. A fully connected or convolutional layer in a network can be described by this equation:

$$\mathrm{z} = g(W\mathrm{x} + \mathrm{b})$$

where W and b are the network parameters of that particular layer and $g(\cdot)$ is any activation function. With batch normalisation this transformation changes as follows:

$$\mathrm{z} = g(\mathrm{BN}(W\mathrm{x}))$$

The bias term $+\mathrm{b}$ can be omitted since the subtraction of the mean during the batch normalisation cancels any effect of it.

During the hyperparameter search, a batch normalisation layer is added before each non-linear activation function.

4.6.2 Number of Convolutional Layers

The encoder part of the VAEs is responsible for gradually reducing the dimensionality of the analysed input images. Therefore the encoder is defined with a stride of two[5] and a minimum 3 subsequent convolutional layers. The number of convolutional layers is increased from 3 up to 15 respectively 19 layers for the low-dimensional respectively the high-dimensional VAEs.

4.6.3 Latent Dimensionality

At the bottleneck of the VAE, the dimensionality of the latent representation can be chosen arbitrarily without any constraint. In order to cover a wide range of different low-dimensional representations, the dimensionality was increased from $2^1, 2^2, \dots 2^{10}$.

4.6.4 Loss Function

In equation 2.3 of section 2.2.2 the ELBO $\mathcal{L}(q)$ was defined and then rewritten in equation 2.6 to the following form:

$$\mathcal{L}(q) = \mathbb{E}_{\mathbf{Z}\sim q(\mathbf{Z})}\left[\log p(\mathbf{X}|\mathbf{Z})\right] - D_{\mathrm{KL}}(q(\mathbf{Z})\|p(\mathbf{Z}))$$

[5]The 6^{th} convolutional layers and all subsequent ones are implemented with a stride of one since the output after the 5^{th} layer is already reduced by a factor of $2^5 = 32$.

The first term $\mathbb{E}_{\mathbf{Z}\sim q(\mathbf{Z})}\left[\log p(\mathbf{X}|\mathbf{Z})\right]$ is often called the *reconstruction loss* and boils down to the comparison of the original image $\mathbf{X}$ to its reconstructed counterpart $\hat{\mathbf{X}}$. The most prominently used loss functions for computing the difference between both images $\mathbf{X}$ and $\hat{\mathbf{X}}$ are the mean squared error (**MSE**) and the cross-entropy loss (**X-Ent**).

The MSE is computed by summing up the pixel-wise squared difference for all training samples $\mathbf{X}_i, \quad \forall i = 1 \dots N$:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(\mathbf{X}_i - \hat{\mathbf{X}}_i\right)^2$$

The cross-entropy loss is also called logistic loss and it is often used for classification problems. Instead of using both the true and predicted value it reports the probability of belonging to a certain class. It is calculated as follows:

$$H(p,q) = -\sum_{\forall x} p(x)\log(q(x))$$

where $p(x)$ is the true distribution $q(x)$ the estimated distribution. In machine learning it is often interpreted that the true distribution $p(x)$ is the ground truth – here the original input image $\mathbf{X}$ – and the estimated distribution $q(x)$ is the predicted output of the neural network – here the reconstructed image $\hat{\mathbf{X}}$. Thus the loss function changes to:

$$H = -\frac{1}{N}\sum_{i=1}^{N}\mathbf{X}_i \cdot \log\left(\hat{\mathbf{X}}_i\right).$$

For the binary case it becomes:

$$H = -\frac{1}{N}\sum_{i=1}^{N}\mathbf{X}_i \cdot \log\left(\hat{\mathbf{X}}_i\right) + (1-\mathbf{X}_i)\cdot\log\left(1-\hat{\mathbf{X}}_i\right).$$

When applied to binary classification cross-entropy loss can become zero when the correct class is predicted with absolute certainty for all samples – $p\left(\mathbf{X}_i\right) = 1.00, \quad \forall i = 1 \dots N$.

However, when the cross-entropy loss is applied to real-valued predictions it must be noted, that even for a perfect classification with absolute certainty, this is not the case. This is demonstrated by a simple example, where the cross-entropy loss is a computed for a toy dataset of one sample

($N = 1$) with $\mathbf{X}_1 = 0.9$ which is correctly predicted, hence $\hat{\mathbf{X}}_1 = 0.9$:

$$\begin{aligned} H &= -\mathbf{X}_1 \cdot \log\left(\hat{\mathbf{X}}_1\right) \\ &= -0.9 \cdot \log(0.9) \\ &\simeq 0.0948 \neq 0 \end{aligned}$$

But regardless of if the cross-entropy loss can actually become zero or not, it can be successfully used for training machine learning models with real-valued output. This fact has to be kept in mind during the implementation and for interpreting the final results.

4.6.5 In-Field-Loss

Figure 3.1 shows two example images. When visually comparing more field images it becomes quickly clear, that most of the variation of the images is caused by the difference in shape. Therefore any machine learning algorithm will try to pick up on this largest variance and encode the differences. Since the shape of the field is not a desired characteristic in this setup – where the focus lies rather on the classification of the plant type as well as the binary and the four loss category described in Table 3.2 and 3.1 – the loss was modified in such a way that the loss was only computed inside of the field, hence the name: In-Field-Loss (**IFL**).

During the masking of the satellite images as described in section 3.4 the area outside of the field was set to zero across all channels. Therefore the reconstruction loss was only computed where the true image had values greater than zero.

5. Results

At first, this chapter presents the results of the hyperparameter search for the low- and high-dimensional implementations of the VAEs. It then exhibits the outcomes of the comparison of the low-dimensional VAE with the conventional computer vision methods. After which the results of the comparison of the same low-dimensional VAEs to its high-dimensional counterpart are shown.

5.1 Hyperparameter Search

This section shows the results from the hyperparameter search outlined in section 4.6 and draws conclusions for each hyperparameter both for the implementation of VAEs for the low-dimensional and the high-dimensional input.

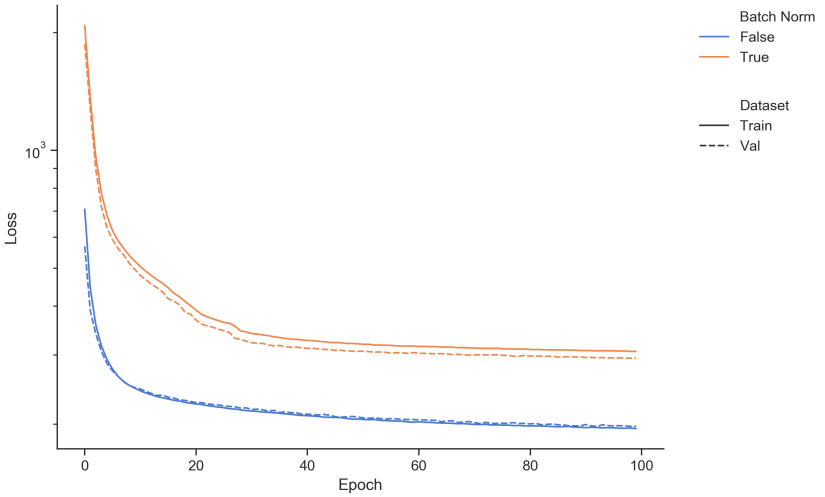
5.1.1 Findings for the Low-dimensional Variational Auto-Encoders

Batch Normalisation

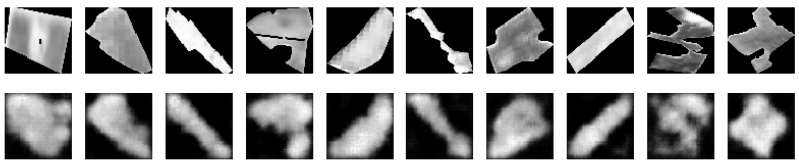
The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when alternating the inclusion of batch normalisation layers for the low-dimensional VAEs, are displayed in Figure 5.1. As stated in section 4.6.1, the normalisation of a single batch increases convergence speed and it often results in reduced loss value (Ioffe and Szegedy, 2015). None of those statements are covered by the results in fig. 5.1a as the convergence is certainly slower and the final loss is larger by a factor of $\times 1.5$. The larger loss can be verified when looking more closely to the comparison of the input images and its reconstructed counterpart in fig. 5.1b and 5.1c since the reconstruction

shows a pattern of black rectangles where the original field has non-zero entries.

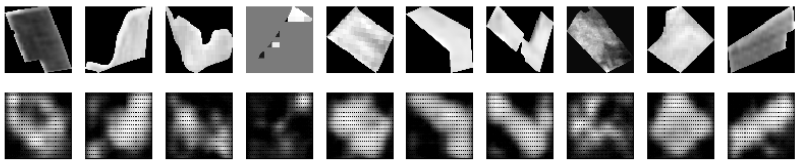
The conclusion for this parameter is that batch normalisation does not seem to have the desired effect and hence is not used for the final model.



(a) Training and validation loss



(b) Without batch normalisation



(c) With batch normalisation

Figure 5.1. Results for alternating the inclusion of batch normalisation layers for the low-dimensional VAEs. Fig. 5.1a shows the training and validation loss for the VAE with and without batch normalisation layers (best viewed in colour). A comparison of the input image to its reconstructed counterpart for the VAE trained for 100 epochs with and without batch normalisation layers is displayed in fig. 5.1b respectively 5.1c.

Number of Convolutional Layers

The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when increasing the number of convolutional layers for the low-dimensional VAEs, are displayed in Figure 5.2. The training and validation loss in fig. 5.2a show a clear tendency: the number of convolutional layers is directly proportional to the final loss, which means that the least amount of convolutional layers returns the lowest loss. This tendency can also be seen when comparing the

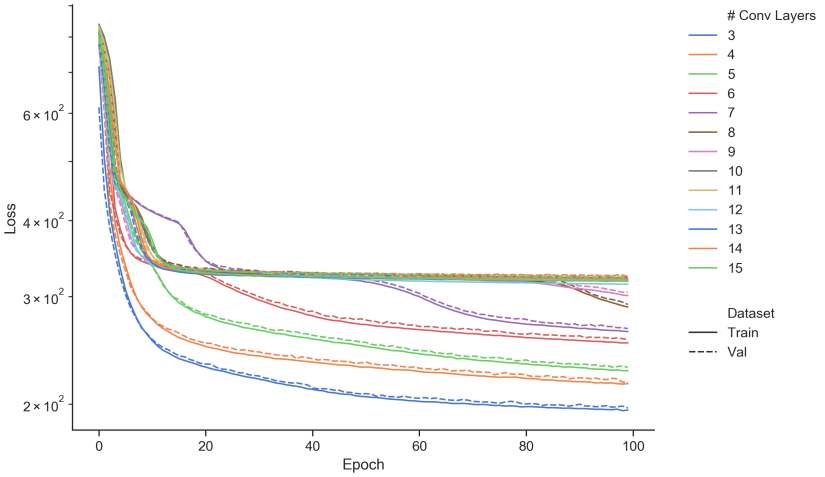
reconstructed images with either 3 or 15 convolutional layers in fig. 5.2b and 5.2c. The reconstruction of the images with 15 convolutional layers outputs basically the same fairly rectangular shaped area regardless of any differences in the intensities of the original field.

This allows drawing the conclusion that the best performance is achieved by three convolutional layers. This finding is somewhat unexpected since more convolutional layers are generally capable of finding more patterns in the images since any additional layer increases the number of trainable parameters. One would certainly assume that a network with more convolutional layers performs at least as good as one with less convolutional layers. This intuition creates the hypothesis that a deeper network with more convolutional layers needs more cautious investigation since the initialisation of layers or the learning rate might be improvable. The research done by He et al. (2015) and the well-known network ResNet is a promising candidate to look into for training increasingly deep architectures.

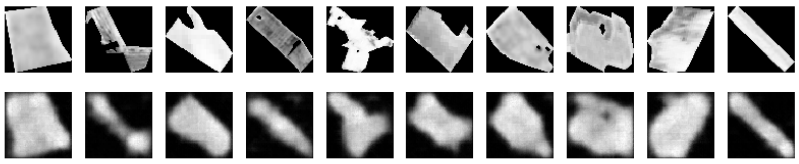
Latent Dimensionality

The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when increasing the dimensionality of the latent representation $\dim(z)$ for the low-dimensional VAEs, are displayed in Figure 5.3. As for the number of convolutional layers the training and validation loss in fig. 5.3a show a clear tendency: the dimensionality of the latent space is indirectly proportional to the final loss, which means that the higher-dimensional the latent space, the lower the final loss. The final loss is decreased by a factor of $\times 2.0$ when comparing the largest and smallest dimensions for the latent space. This tendency can also be verified when comparing the reconstructed images for either a 2 or 1024-dimensional latent space in fig. 5.3b and 5.3c. The reconstruction from a 2-dimensional latent representation lacks the differences in the intensities of the original field since it cannot encode enough information in 2 variables. The reconstruction from the high-dimensional latent space is significantly better at resembling variations in the intensities of the original field.

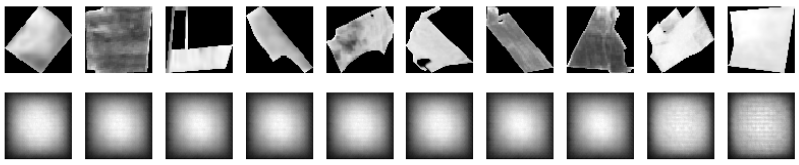
This analysis allows to draw the conclusion that the best performance is achieved for $\dim(z) = 1024$. It should be noted that the final loss is just marginally higher when increasing the dimensionality from 128 to larger values. This means that if a further compression of the latent representation is needed, smaller values can be chosen up to a lower bound



(a) Training and validation loss



(b) 3 convolutional layers



(c) 15 convolutional layers

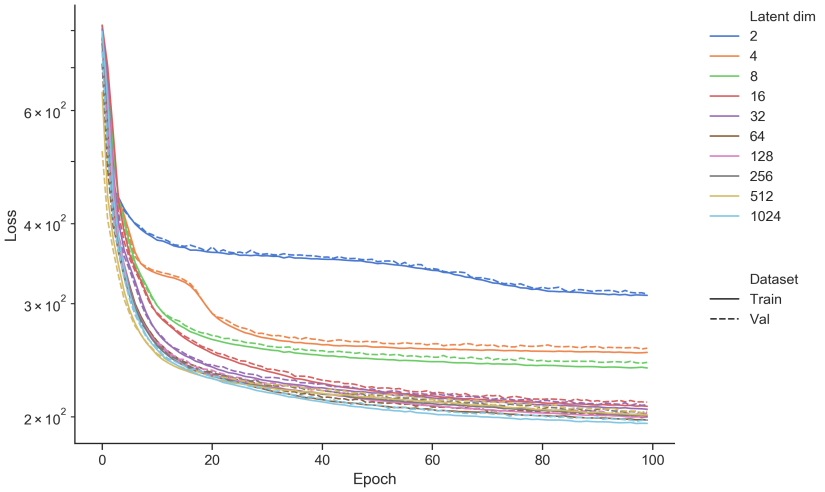
Figure 5.2. Results for increasing the number of convolutional layers for the low-dimensional VAEs. Fig. 5.2a shows the training and validation loss for the VAE with different numbers of convolutional layer (best viewed in colour). A comparison of the input image to its reconstructed counterpart for the VAE trained for 100 epochs with 3 and 15 and convolutional layers is displayed in fig. 5.2b respectively 5.2c.

of $\dim(z) = 128$.

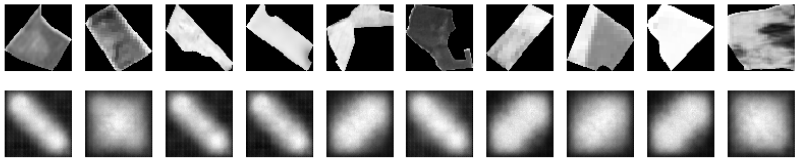
This finding is aligned with the expectation since a smaller latent space means effectively a higher compression, which itself comes along with an information loss. Hence it is intuitive that the largest latent space allows the best reconstruction and thus smallest error.

Loss Function

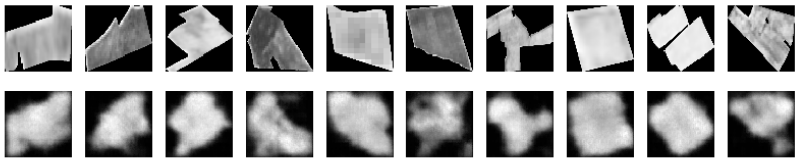
The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when alternating the loss function used in the reconstruction loss for the low-dimensional VAEs, are displayed in Figure 5.4. As explained previously in section 4.6.4 the scale of the loss for the very same prediction is different for the used loss functions MSE and X-Ent, which is immediately visible in fig. 5.4a.



(a) Training and validation loss



(b) $\dim(z) = 2$

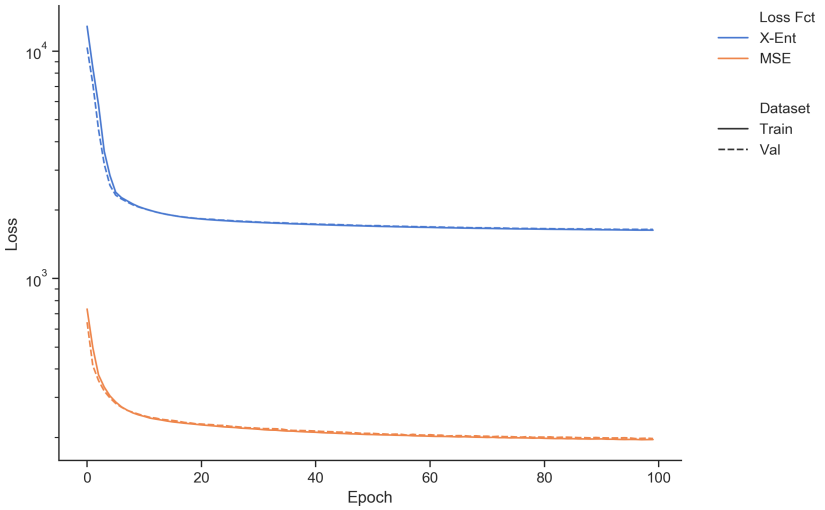


(c) $\dim(z) = 1024$

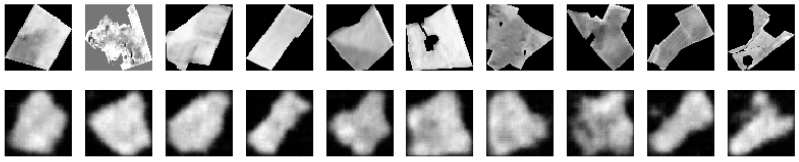
Figure 5.3. Results for increasing the dimensionality of the latent representation $\dim(z)$ for the low-dimensional VAEs. Fig. 5.3a shows the training and validation loss for the VAE with different dimensions of the latent representation $\dim(z)$ (best viewed in colour). A comparison of the input image to its reconstructed counterpart for the VAE trained for 100 epochs with $\dim(z) = 2$ and $\dim(z) = 1024$ is displayed in fig. 5.3b respectively 5.3c.

Keeping in mind that in this analysis a lower absolute loss value does not ultimately mean a better reconstruction draws more attention to the visual comparison of the reconstructed images using either MSE or X-Ent in fig. 5.4b and 5.4c. Those figures show that regardless of which loss function is used the reconstruction is capable of providing a very similar level of detail.

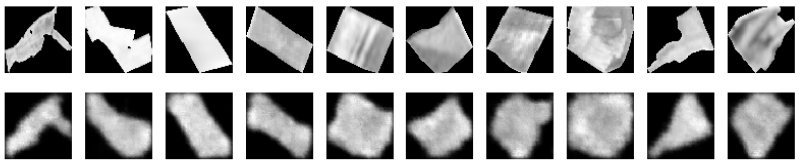
The apparent indifference in the choice of loss functions means that it could be chosen arbitrarily without any influence on the performance of the model. Finally, the MSE was chosen for the model though similar results would have been achieved if the X-Ent loss function would have been chosen.



(a) Training and validation loss



(b) Mean Squared Error



(c) Binary Cross-entropy

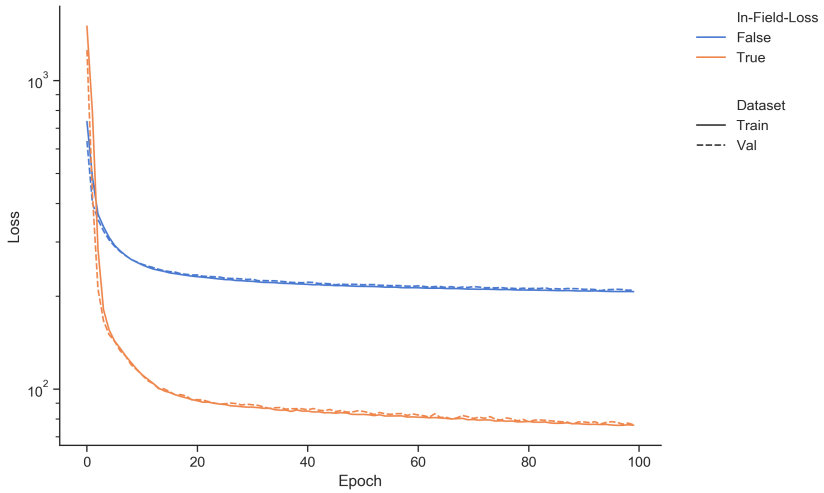
Figure 5.4. Results for alternating the loss function used in the reconstruction loss for the low-dimensional VAEs. Fig. 5.4a shows the training and validation loss for the VAE using different a loss functions in the reconstruction loss (best viewed in colour). A comparison of the input image to its reconstructed counterpart for the VAE trained for 100 epochs with MSE and X-Ent as a loss function in the reconstruction loss is displayed in fig. 5.4b respectively 5.4c.

In-Field-Loss

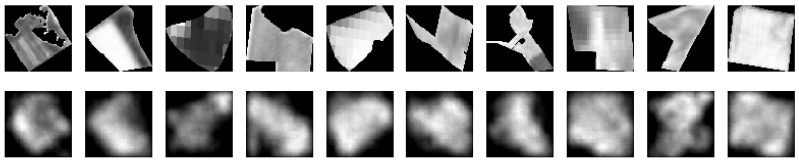
The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when alternating if the reconstruction is computed only inside the field for the low-dimensional VAEs, are displayed in Figure 5.5. For the case that the loss is only computed inside the field it is intuitive that the scale of the loss will be significantly smaller since less pixels contribute to the calculated loss value. This intuition can be verified when looking at fig. 5.5a. Restricting the loss function to pixels inside the actual field have the consequence that the shape the field is not learned and encoded by the VAE. This consequence was the principal motivation for introducing the In-Field-Loss in section 4.6.5. Comparing fig. 5.5b and 5.5c shows clearly that in the case

of only computing the loss in the area of the field the VAE does not encode the shape of the field and hence is not successful in its reconstruction.

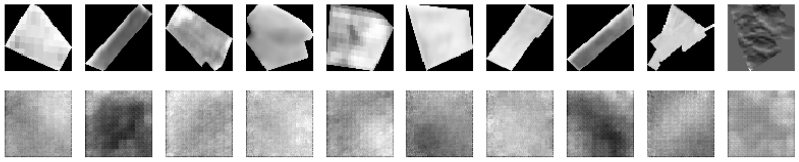
But since later on the primal objective of the VAE is not to reconstruct the original field but to learn a meaningful latent representation which itself is used for the three classification tasks, no obvious recommendation for restricting the computation of the loss to the inside of the field can be made. Hence both methods are selected, explored and evaluated for the final classification tasks.



(a) Training and validation loss



(b) Without In-Field-Loss



(c) With In-Field-Loss

Figure 5.5. Results for alternating if the reconstruction is computed only inside the field for the low-dimensional VAEs. Fig. 5.5a shows the training and validation loss for the VAE with and without In-Field-Loss (best viewed in colour). A comparison of the input image to its reconstructed counterpart for the VAE trained for 100 epochs without and and with In-Field-Loss is displayed in fig. 5.5b respectively 5.5c.

Summary

The findings for the hyperparameter search for the low-dimensional VAEs are summarised Table 5.1.

Network Parameter	Best Value
Usage of Batch Normalisation layers	`False`
Number of Convolutional Layers	3
Dimensionality of Latent Representation	1024
Usage of Loss Function for the Reconstruction Loss	`MSE`
Usage of In-Field-Loss	`True` and `False`

Table 5.1. Optimal hyperparameters for the low-dimensional VAEs.

5.1.2 Findings for the High-dimensional Variational Auto-Encoders

This section does not describe and discuss the findings for the high-dimensional case for better readability and because the arguments and the conclusions from the low-dimensional case can be transferred without any exception.

The best combination of hyperparameters for the high-dimensional case is thus the same as for the low-dimensional one, summarised in Table 5.1.

A more detailed view on the individual hyperparameters and its training and validation loss as well as the comparison of the input image to the reconstructed counterpart can be found in the appendix section A.

5.2 Comparison with Conventional Computer Vision Methods

Similar to the search for optimal hyperparameters for the VAEs for the low- and high-dimensional implementation a feature selection was carried out to select the optimal features. This means comparing the feature extraction methods proposed in section 4.2 computed at different parameter settings. A wrapper-based feature selection approach was adopted where the quality of features is evaluated, indirectly, based on the F_1-score and accuracy of the classification algorithm. The performance of the classification algorithm is evaluated using five-fold cross-validation.

Since each feature type has different parameters, the feature selection process was carried out in a sequential manner, by first finding good features for each feature type followed by comparing them to get the optimal features. The good features for each feature type are computed by parameter optimization. That is, good GLCM features are computed by optimizing the GLCM parameters via-grid search. Similarly, good LBP and HOG are obtained which are then compared each other to find the

optimal features.

The optimal features obtained from this procedure are LBP features[1] fed into the AdaBoost[2] classifier with decision trees as weak learners. This algorithm is referenced in Table 5.2 under the name *Best Conventional*, since it resembles the best algorithm based on conventional CV methods. It is compared against the best VAEs with those hyperparameters specified in section 5.1.1. All algorithms have in common that they were trained on the same low-dimensional images.

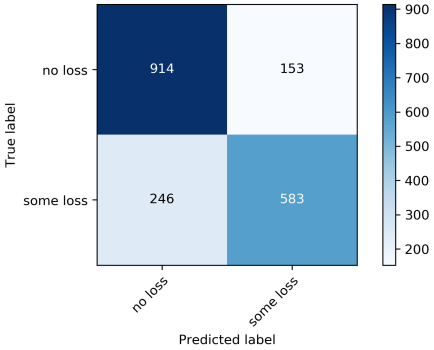
Table 5.2 summarises the performance of the best conventional algorithm and the best low-dimensional implementations of the VAEs on the three classification tasks defined in Table 4.1 by measuring the F_1-score and accuracy. This table is a condensed view of the underlying confusion matrices, for which for better readability only the confusion matrix for the classification of the Loss-2D is displayed in Figure 5.6. All remaining confusion matrices can be found in the appendix section B.

Algorithm Classification Tasks	Input Dimension	F_1	Accuracy
Best Conventional	$64 \times 64 \times 1$		
(i) Loss-4D		0.42	0.58
(ii) Loss-2D		0.75	0.79
(iii) Plant-5D		0.44	0.48
Best VAE – without In-Field-Loss	$64 \times 64 \times 1$		
(i) Loss-4D		0.22	0.43
(ii) Loss-2D		0.67	0.55
(iii) Plant-5D		0.24	0.41
Best VAE – with In-Field-Loss	$64 \times 64 \times 1$		
(i) Loss-4D		0.22	0.43
(ii) Loss-2D		0.67	0.55
(iii) Plant-5D		0.23	0.40

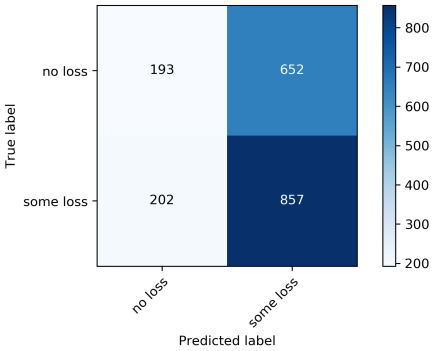
Table 5.2. Performance comparison for the best conventional algorithm and the best low-dimensional VAEs without and with In-Field-Loss. Higher values for the F_1-score and accuracy are better.

[1]The optimal LBP features are computed with the following parameter values $r = 1$, $P = 8$, $d = 25$ where r defines the radius of circle (spatial resolution of the operator), P defines the number of circularly symmetric neighbour set points (quantisation of the angular space) and d defines the number of bins in the histogram.

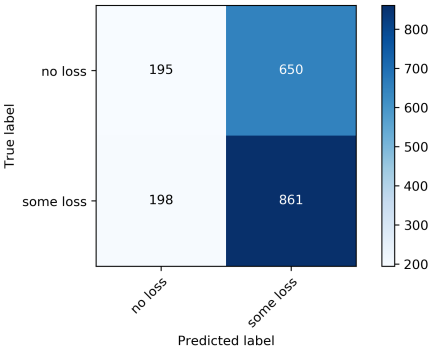
[2]The best hyperparameters for AdaBoost are a learning rate of 1 together with 600 weak learners.



(a) Best Conventional



(b) Best VAE without IFL



(c) Best VAE with IFL

Figure 5.6. Confusion matrices for the Loss-2D classification for the best conventional algorithm (fig. 5.6a) as well as the best low-dimensional VAEs without and with In-Field-Loss (fig. 5.6b resp. 5.6c).

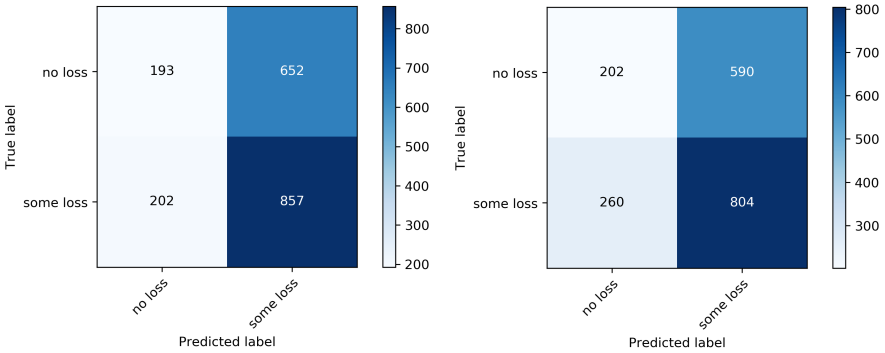
5.3 Comparison of Low- and High-dimensional Variational Auto-Encoders

This section shows the comparison of the best low- and high-dimensional implementations of the VAEs with those hyperparameters specified in section 5.1.1 respectively 5.1.2.

Table 5.3 summarises the performance of the algorithms on the three classification tasks defined in Table 4.1 by measuring the F_1-score and accuracy. Again this table is a condensed view of the underlying confusion matrices, for which for better readability only the confusion matrix for the classification of the Loss-2D is displayed in Figure 5.7. All remaining confusion matrices can be found in the appendix section C.

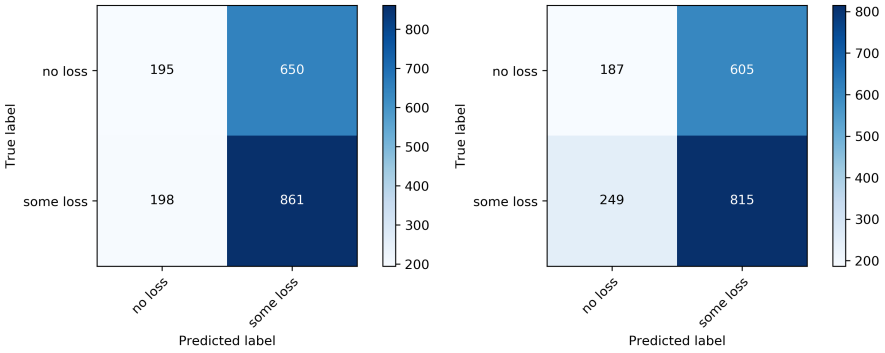
Algorithm Classification Tasks	Input Dimension	F_1	Accuracy
without In-Field-Loss			
Best VAE	$64 \times 64 \times 1$		
(i) Loss-4D		0.22	0.43
(ii) Loss-2D		0.67	0.55
(iii) Plant-5D		0.24	0.41
Best VAE	$512 \times 512 \times 13$		
(i) Loss-4D		0.22	0.41
(ii) Loss-2D		0.65	0.54
(iii) Plant-5D		0.17	0.35
with In-Field-Loss			
Best VAE	$64 \times 64 \times 1$		
(i) Loss-4D		0.22	0.43
(ii) Loss-2D		0.67	0.55
(iii) Plant-5D		0.23	0.40
Best VAE	$512 \times 512 \times 13$		
(i) Loss-4D		0.22	0.41
(ii) Loss-2D		0.66	0.54
(iii) Plant-5D		0.17	0.32

Table 5.3. Performance comparison for the best low- and high-dimensional VAEs without and with In-Field-Loss. Higher values for the F_1-score and accuracy are better.



(a) Best low-dimensional VAE without IFL

(b) Best high-dimensional VAE without IFL



(c) Best low-dimensional VAE with IFL

(d) Best high-dimensional VAE with IFL

Figure 5.7. Confusion matrices for the Loss-2D classification for the best low- and high-dimensional VAEs without (fig. 5.7a respectively 5.7b) and with (fig. 5.7c respectively 5.7d) In-Field-Loss.

6. Discussion

This chapter discusses first the results of the comparison of the low-dimensional VAEs with the conventional computer vision methods and then continues with a discussion of the comparison of the same low-dimensional VAEs with its high-dimensional counterpart.

Table 5.2 summarises the performance of the low-dimensional VAEs and the conventional method on the three classification tasks by stating their scores on the F_1 metric and accuracy. The table shows that the conventional computer vision methods outperform the low-dimensional VAEs in any classification problem regardless of if the loss is computed only within the actual field or not.

It is rather unexpected that the low-dimensional implementation of the VAEs with its $\sim 1.3 \times 10^6$ trainable parameters perform worse than a simple feature extractor such as LBP. But paying attention to the two main differences between both approaches helps to explain this shortcoming.

First, the conventional algorithm uses AdaBoost as a classifier whereas the low-dimensional implementation of the VAEs uses one fully connected layer to perform each of the classification tasks. Using a boosted classifier such as AdaBoost means essentially that there is a plurality in classifiers as there are many weak estimators applied to the extracted features to perform the classification task. In fact, AdaBoost is implemented with 600 classifiers here. As described in section 4.3.3 the advantage of boosted classifiers is the combination of many weak learners to pay more attention to increasingly difficult to classify samples. In the low-dimensional implementation of the VAEs on the other hand, only a single classifier is responsible for the decision between classes. Therefore this disadvantage is responsible at least for some part of the performance hit the low-dimensional implementation of the VAEs take.

The second main difference is the way how both methods find the latent

representation. The VAEs have a powerful mechanism that learns a low-dimensional representation of the input whereas the feature computation for the LBP is defined. This generally means that LBP is rather rigid and not flexible enough to select relevant features for more challenging tasks while a VAE is capable of adapting depending on the input and the task. Having said that, the low-dimensional VAEs with its $\sim 1.3 \times 10^6$ trainable parameters should at least be able to replicate a similar representation to the one defined by LBP. But the performance values in Table 5.2 strongly indicate that the features found by the low-dimensional VAEs are less relevant to the classification tasks than the features the LBP defines. Even though it is difficult to examine the sole effect of the learned latent representation due to the different classifiers used.

Table 5.3 summarises the performance of the low- and high dimensional VAEs on the three classification tasks by stating their scores on the F_1 metric and accuracy. The table shows that both approaches deliver matching classification results.

Since the amount of information that is available to distinguish between different classes increases drastically from the low-dimensional to the high-dimensional implementation one would assume that the classification task becomes easier and more accurate, but the metric values in the table clearly indicate that the additional information does not yield a higher prediction accuracy.

This conclusion undermines the belief that the provided data is sufficiently informative to train deep classifiers for a determined prediction on the classification tasks. But when judging the informative value of the provided data one should keep in mind two aspects of this analysis.

The first detail is that the analysis presented was not conducted on the entire data set but rather a small subset of it. The subset contains only the five most frequent plant species and drops 53 less frequent plants. Additionally, within those five most frequent plant species a subset of roughly $19{,}000$ fields was selected while there were over $380{,}000$ fields belonging to either of the five plants.

The second aspect is that during this research one point in time was chosen for the given time window of August 2015. This choice was mainly motivated by the fact that the typical harvesting month for most species in Finland is the month of August. This particular choice might be beneficial for classifying the type of crop loss but it might not contain enough information to build a robust prediction for the plant species. But the intelligent

crop model which is the large scale scope this analysis it contributing to will leverage time-series data. That means that more points in time will be taken into consideration and thus additional information will be included that is more explanatory for the given classification tasks.

Furthermore, Table 5.3 shows that restricting the area where the loss is computed does not have the expected effect on the stated metrics in a significant and consistent way.

The In-Field-Loss was introduced to force the VAEs to pay more attention to the underlying structure of the satellite images instead of its shape. But the values in the table do not support this motivation. Instead of setting the outside of the field to zero during the masking one improvement could be to set it to the average value of that particular channel. Thereby the variance per channel would be reduced significantly and the network would automatically not pick up on the shape of the field any more. There would not be the need to force the network to not take the shape of the field into consideration.

7. Conclusion

Summarising high-resolution satellite images by using Variational Auto-Encoders is possible. It will, however, still take a performance hit on the outlined classification tasks in comparison to conventional computer vision techniques. This statement is true for using both low- and high-dimensional input images as well as both alternations of how the reconstruction loss is restricted to the actual field area.

Recent improvements in image classification tasks on the famous ImageNet database (Deng et al., 2009) by the prominent networks AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan and Zisserman, 2014), GoogLeNet / Inception (Szegedy et al., 2014) and ResNet (He et al., 2015) have successfully demonstrated that deep learning is a powerful tool capable of beating human-level performance. Therefore with more enhanced testing on network structure as well as hyperparameter search the proposed VAEs are a promising implementation with the potential to deliver superior results to the conventional methods while significantly reducing the image file size. Especially the novelty of *skip-connections* introduced by He et al. as part of ResNet enables training of exceptionally deep architectures which together with increasing the amount of training data to the full scope can deliver encouraging improvements to the current status quo.

Another idea that could improve the results is a combination of the training of the VAEs with the classification task. Until now the quality of the VAEs is evaluated by selecting the least loss together with a visual inspection of the reconstructed image in case two candidates return similar loss. A direct combination of both tasks – training and classification – enables a more direct evaluation of the desired task. Even though balancing both loss contributions is nothing short of cumbersome.

References

Aalto University Science-IT (2019). Triton. https://scicomp.aalto.fi/triton/overview.html/.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from https://www.tensorflow.org.

Bargoti, S. and Underwood, J. P. (2016). Deep Fruit Detection in Orchards. *CoRR*, abs/1610.03677:3626–3633.

Ben-Hur, A., Horn, D., Siegelmann, H. T., and Vapnik, V. (2001). Support vector clustering. *Journal of Machine Learning Research*, 2:125–137.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

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

Dalal, N. and Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In Schmid, C., Soatto, S., and Tomasi, C., editors, *International Conference on Computer Vision & Pattern Recognition (CVPR '05)*, volume 1, pages 886–893. IEEE Computer Society.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*, abs/1810.04805.

Dumoulin, V. and Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Dyrmann, M., Jørgensen, R., and Midtiby, H. (2017). RoboWeedSupport - Detection of weed locations in leaf occluded cereal crops using a fully convolutional neural network. *Advances in Animal Biosciences*, 8:842–847.

Freund, Y. and Schapire, R. E. (1999). A Short Introduction to Boosting. In *In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 1401–1406. Morgan Kaufmann.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. *CoRR*, abs/1512.03385.

Ho, T. K. (1995). Random decision forests. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1 of *ICDAR*, pages 278–282. IEEE Computer Society.

Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*, abs/1502.03167.

Jimenez Rezende, D., Mohamed, S., and Wierstra, D. (2014). Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv preprint arXiv:1401.4082*.

Jordan, J. (2018). Variational Autoencoders. https://www.jeremyjordan.me/variational-autoencoders/. [Online; accessed 05-Sep-2018].

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc.

Kussul, N., Lemoine, G., Gallego, F. J., Skakun, S. V., Lavreniuk, M., and Shelestov, A. Y. (2016). Parcel-Based Crop Classification in Ukraine Using Landsat-8 Data and Sentinel-1A Data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(6):2500–2508.

Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., and Aila, T. (2018). Noise2Noise: Learning Image Restoration without Clean Data. *CoRR*, abs/1803.04189.

McCool, C., Perez, T., and Upcroft, B. (2017). Mixtures of Lightweight Deep Convolutional Neural Networks: Applied to Agricultural Robotics. *IEEE Robotics and Automation Letters*, PP:1–1.

Mohanty, S. P., Hughes, D. P., and Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. *CoRR*, abs/1604.03169.

Nevo, S., Anisimov, V., Elidan, G., El-Yaniv, R., Giencke, P., Gigi, Y., Hassidim, A., Moshe, Z., Schlesinger, M., Shalev, G., Tirumali, A., Wiesel, A., Zlydenko, O., and Matias, Y. (2019). ML for Flood Forecasting at Scale. *CoRR*, abs/1901.09583.

Ojala, T., Pietikäinen, M., and Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51–59.

Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., and Carin, L. (2016). Variational Autoencoder for Deep Learning of Images, Labels and Captions. *arXiv preprint arXiv:1609.08976*.

Reaktor Space Lab (2019). Reaktor Hello World. `https://reaktorspace.com/reaktor-hello-world/`. [Online; accessed 17-May-2019].

Rebetez, J., Satizábal, H. F., Mota, M., Noll, D., Büchi, L., Wendling, M., Cannelle, B., Pérez-Uribe, A., and Burgos, S. (2016). Augmenting a convolutional neural network with local histograms - A case study in crop classification from high-resolution UAV imagery. In *24th European Symposium on Artificial Neural Networks, ESANN*.

Simonyan, K. and Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2014). Going deeper with convolutions. *CoRR*, abs/1409.4842.

Yalcin, H. (2017). Plant phenology recognition using deep learning: Deep-Pheno. In *2017 6th International Conference on Agro-Geoinformatics*, pages 1–5.

Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional Networks. In *CVPR*, volume 10, page 7.

A. Findings for High-dimensional Variational Auto-Encoders

This chapter summarises the findings for the high-dimensional case in fewer details as for the low-dimensional one, where the findings for each hyperparameter were analysed and a justified conclusion was made because the arguments and the conclusions are still valid without any exception.

A.1 Batch Normalisation

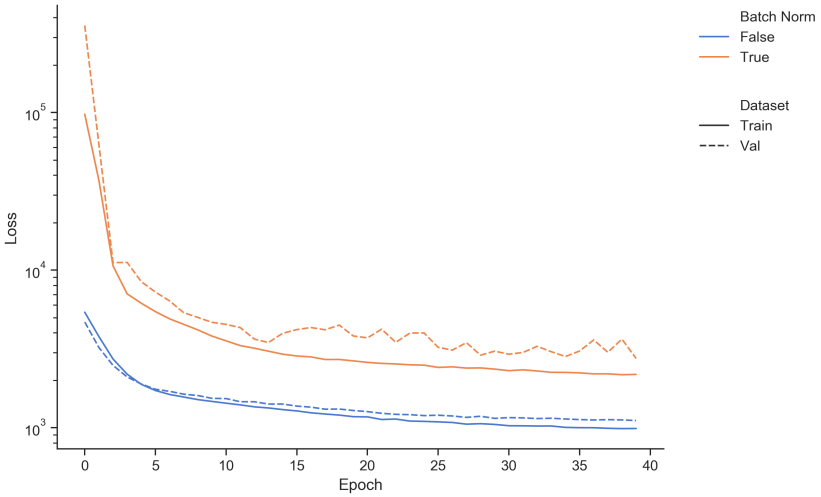
The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when alternating the inclusion of batch normalisation layers for the high-dimensional VAEs, are displayed in Figure A.1.

The conclusion for this hyperparameter is that no batch normalisation is used in the final model. The argument is the same as in the low-dimensional case.

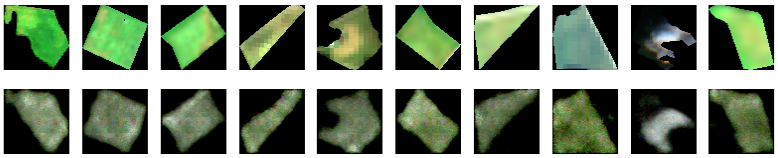
A.2 Number of Convolutional Layers

The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when increasing the number of convolutional layers for the high-dimensional VAEs, are displayed in Figure A.2.

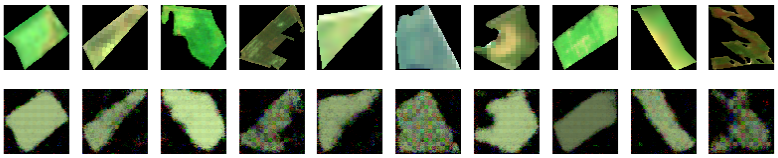
The best performance is achieved by three convolutional layers, which is aligned with the result for the low-dimensional case.



(a) Training and validation loss

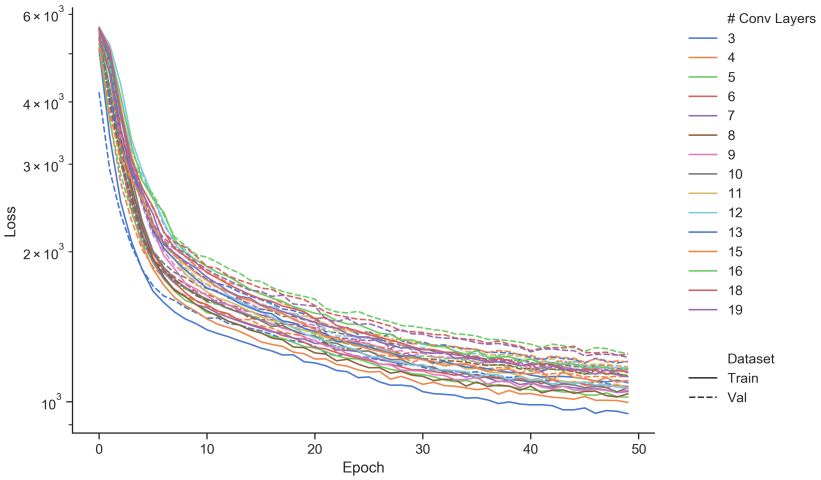


(b) Without batch normalisation

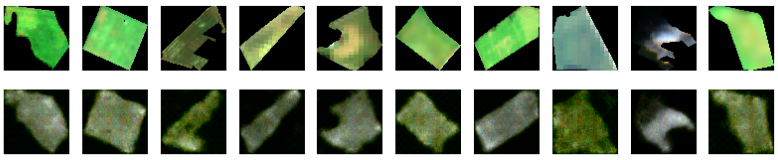


(c) With batch normalisation

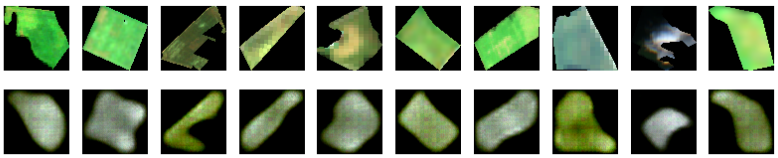
Figure A.1. Results for alternating the inclusion of batch normalisation layers for the high-dimensional VAEs. Fig. A.1a shows the training and validation loss for the VAE with and without batch normalisation layers. A comparison of the input image to its reconstructed counterpart for the VAE trained for 40 epochs with and without batch normalisation layers is displayed in fig. A.1b respectively A.1c. (best viewed in colour)



(a) Training and validation loss

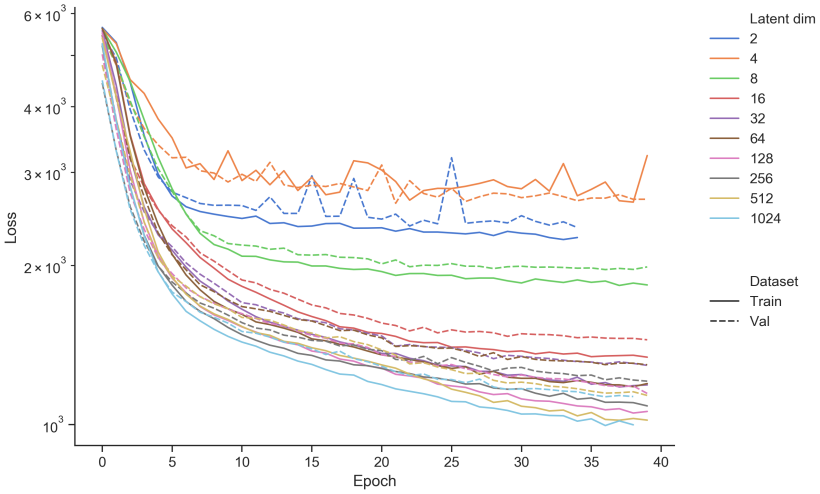


(b) 3 convolutional layers

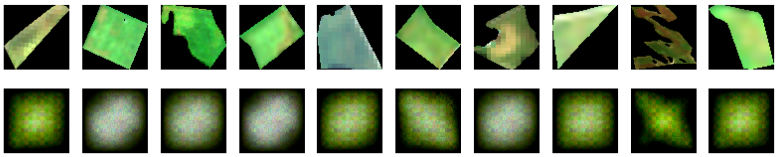


(c) 19 convolutional layers

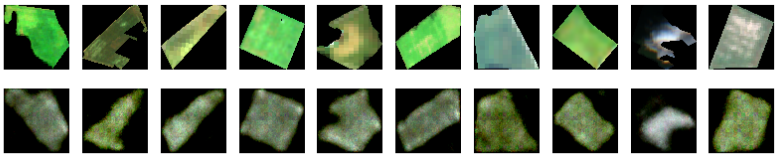
Figure A.2. Results for increasing the number of convolutional layers for the high-dimensional VAEs. Fig. A.2a shows the training and validation loss for the VAE with different numbers of convolutional layer. A comparison of the input image to its reconstructed counterpart for the VAE trained for 50 epochs with 3 and 19 and convolutional layers is displayed in fig. A.2b respectively A.2c. (best viewed in colour)



(a) Training and validation loss



(b) $\dim(z) = 2$



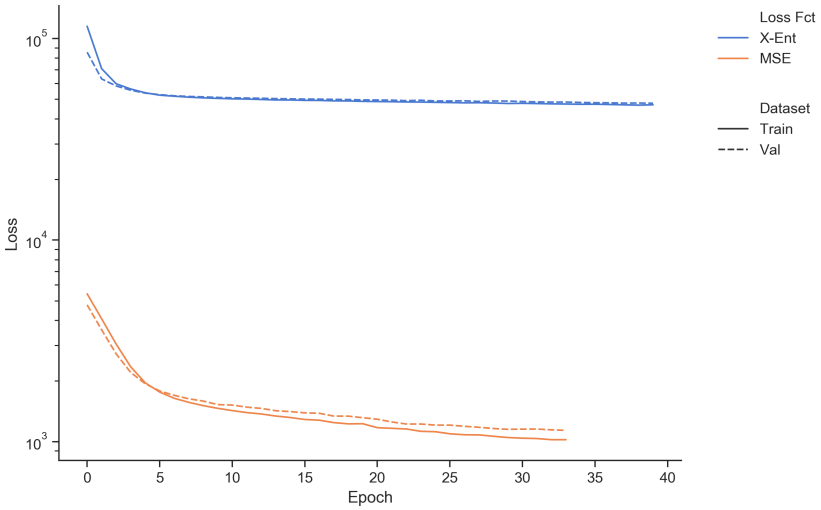
(c) $\dim(z) = 1024$

Figure A.3. Results for increasing the dimensionality of the latent representation $\dim(z)$ for the high-dimensional VAEs. Fig. A.3a shows the training and validation loss for the VAE with different dimensions of the latent representation $\dim(z)$. A comparison of the input image to its reconstructed counterpart for the VAE trained for 40 epochs with $\dim(z) = 2$ and $\dim(z) = 1024$ is displayed in fig. A.3b respectively A.3c. (best viewed in colour)

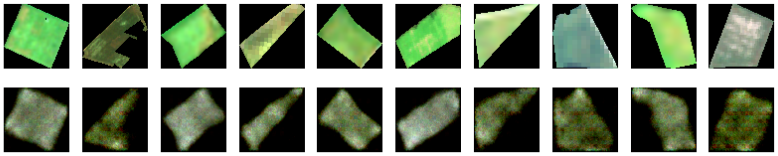
A.3 Latent Dimensionality

The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when increasing the dimensionality of the latent representation $\dim(z)$ for the high-dimensional VAEs, are displayed in Figure A.3.

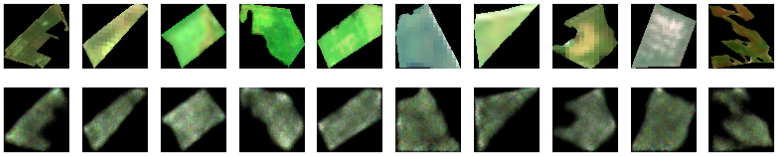
Similar to the low-dimensional case $\dim(z) = 1024$ achieves the best performance but dimensionality from 128 to larger values result in just marginally higher final loss values. In case a further compression of the latent representation is needed, smaller values can be chosen up to a lower bound of $\dim(z) = 128$.



(a) Training and validation loss



(b) Mean Squared Error



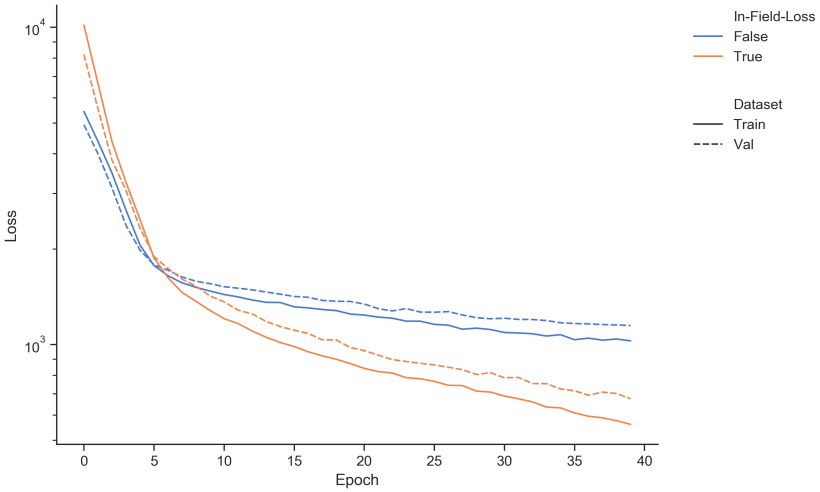
(c) Binary Cross-entropy

Figure A.4. Results for alternating the loss function used in the reconstruction loss for the high-dimensional VAEs. Fig. A.4a shows the training and validation loss for the VAE using different a loss functions in the reconstruction loss. A comparison of the input image to its reconstructed counterpart for the VAE trained for 35 respectively 40 epochs with MSE and X-Ent as a loss function in the reconstruction loss is displayed in fig. A.4b respectively A.4c. (best viewed in colour)

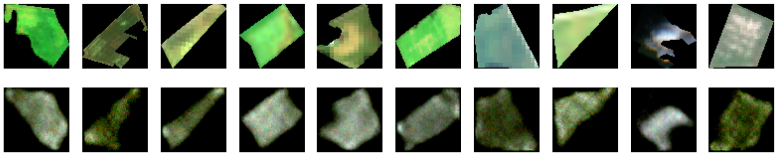
A.4 Loss Function

The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when alternating the loss function used in the reconstruction loss for the high-dimensional VAEs, are displayed in Figure A.4.

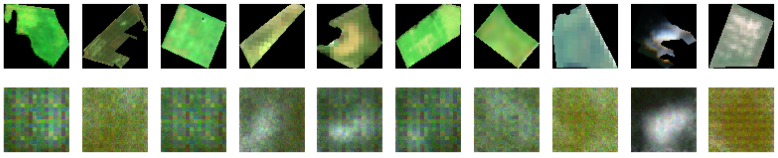
Aligned with the low-dimensional case the choice of loss function does not have a significant effect on the reconstruction capabilities of the model. The MSE was chosen for the model, but similar results would have been achieved if the X-Ent loss function would have been chosen.



(a) Training and validation loss



(b) Without In-Field-Loss



(c) With In-Field-Loss

Figure A.5. Results for alternating if the reconstruction is computed only inside the field for the high-dimensional VAEs. Fig. A.5a shows the training and validation loss for the VAE with and without In-Field-Loss. A comparison of the input image to its reconstructed counterpart for the VAE trained for 40 epochs without and and with In-Field-Loss is displayed in fig. A.5b respectively A.5c. (best viewed in colour)

A.5 In-Field-Loss

The progress of the training and validation loss, as well as the comparison of the input image to the reconstructed counterpart when alternating if the reconstruction is computed only inside the field for the high-dimensional VAEs, are displayed in Figure A.5.

Again the results do not show one conclusive tendency, hence both methods are selected, explored and evaluated for the final classification tasks.

A.6 Summary

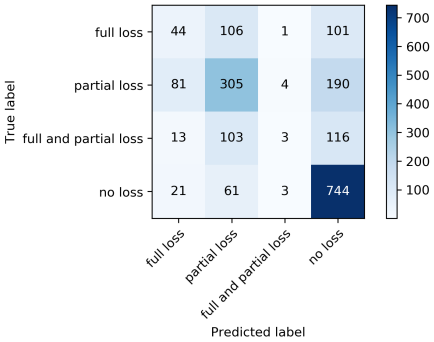
The findings for the hyperparameter search for the high-dimensional VAEs are summarised Table A.1.

Network Parameter	Best Value
Usage of Batch Normalisation layers	`False`
Number of Convolutional Layers	3
Dimensionality of Latent Representation	1024
Usage of Loss Function for the Reconstruction Loss	`MSE`
Usage of In-Field-Loss	`True` and `False`

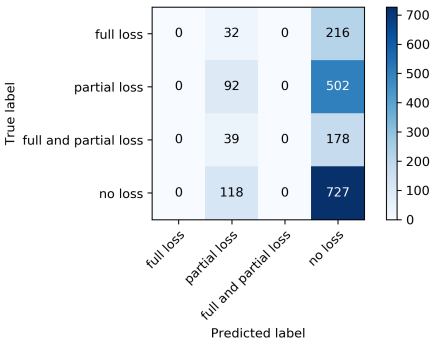
Table A.1. Optimal hyperparameters for the high-dimensional VAEs.

B. Confusion Matrices for the Comparison with Conventional Computer Vision Methods

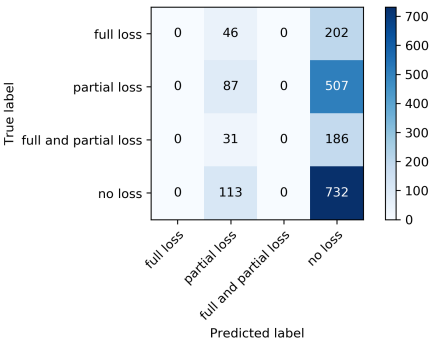
This chapter displays the remaining confusion matrices for the Loss-4D and Plant-5D classification for the best conventional algorithm as well as the best low-dimensional VAEs without and with In-Field-Loss in Figure B.1 and B.2.



(a) Best Conventional

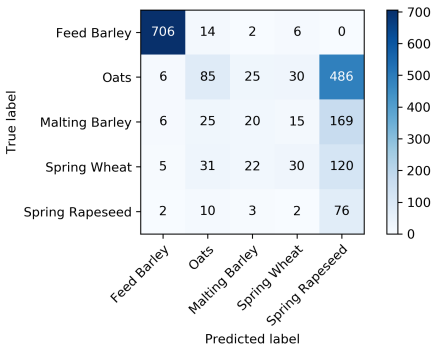


(b) Best VAE without IFL

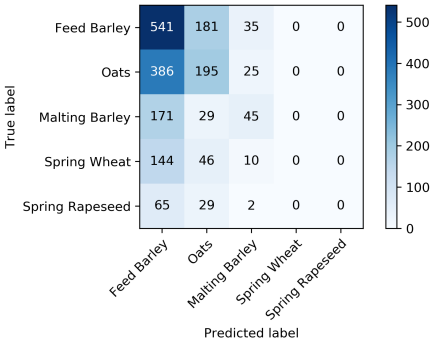


(c) Best VAE with IFL

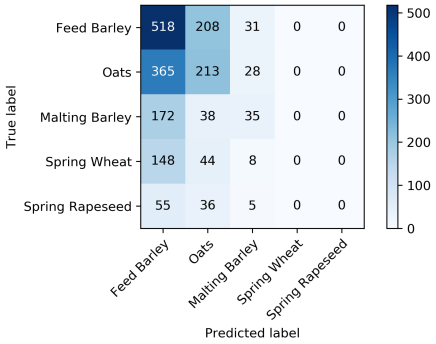
Figure B.1. Confusion matrices for the Loss-4D classification for the best conventional algorithm (fig. B.1a) as well as the best low-dimensional VAEs without and with In-Field-Loss (fig. B.1b respectively B.1c).



(a) Best Conventional



(b) Best VAE without IFL

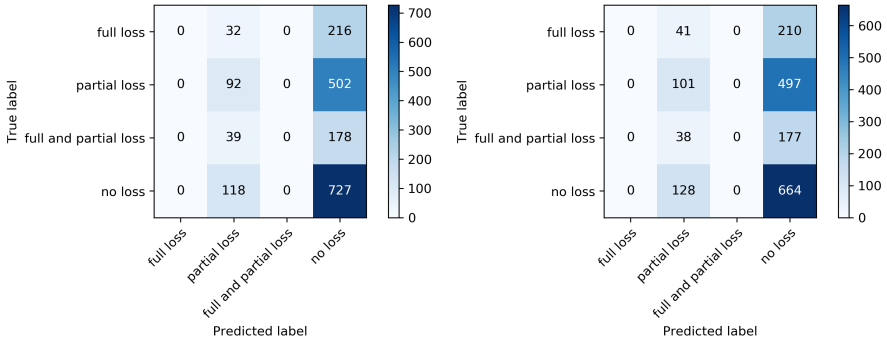


(c) Best VAE with IFL

Figure B.2. Confusion matrices for the Plant-5D classification for the best conventional algorithm (fig. B.2a) as well as the best low-dimensional VAEs without and with In-Field-Loss (fig. B.2b respectively B.2c).

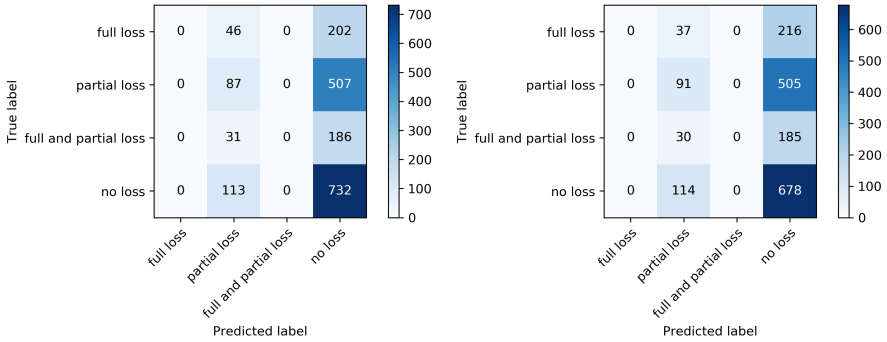
C. Confusion Matrices for the Comparison of Low- and High-Dimensional Variational Auto-Encoders

This chapter displays the remaining confusion matrices for the Loss-4D and Plant-5D classification for the best low- and high-dimensional VAEs without and with In-Field-Loss in Figure C.1 and C.2.



(a) Best low-dimensional VAE without IFL

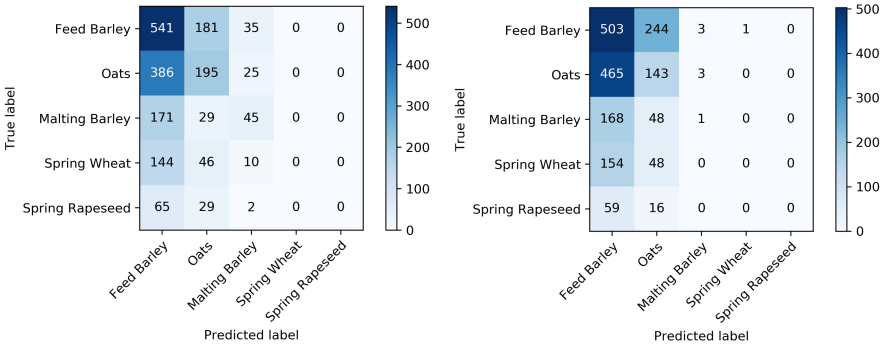
(b) Best high-dimensional VAE without IFL



(c) Best low-dimensional VAE with IFL

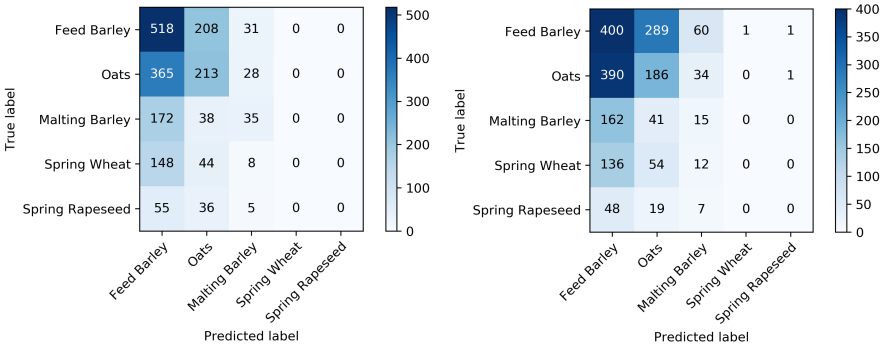
(d) Best high-dimensional VAE with IFL

Figure C.1. Confusion matrices for the Loss-4D classification for the best low- and high-dimensional VAEs without (fig. C.1a respectively C.1b) and with (fig. C.1c respectively C.1d) In-Field-Loss.



(a) Best low-dimensional VAE without IFL

(b) Best high-dimensional VAE without IFL



(c) Best low-dimensional VAE with IFL

(d) Best high-dimensional VAE with IFL

Figure C.2. Confusion matrices for the Plant-5D classification for the best low- and high-dimensional VAEs without (fig. C.2a respectively C.2b) and with (fig. C.2c respectively C.2d) In-Field-Loss.