A LATENT VARIABLE MODEL FOR PLANT STRESS
PHENOTYPING USING DEEP LEARNING

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Jordan Ubbens

©Jordan Ubbens, March 2020. All rights reserved.


PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Or

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9
Canada
ABSTRACT

With a growing population and a changing climate, increasing crop yields in a diversity of environmental conditions is becoming increasingly important. Studying genome-by-environment (GxE) effects is a critical path for such improvements, and high-throughput plant phenotyping is necessary for carrying out such experiments at scale. Image-based phenotyping techniques offer a scalable, non-destructive way of quantifying plants’ responses to their environment – however, these techniques can be cumbersome and subjective. Each image dataset is unique, and requires either a hand-crafted image processing pipeline or a large annotated training set, which can be expensive and time-consuming. Additionally, researchers must select what feature is to be used to quantify changes due to the treatment, such as biomass, colour, the number of organs, or some other visual indication of the individual’s response to its environment.

This dissertation explores image-based plant phenotyping, beginning with a discussion of image processing tools. Deep learning is introduced, with a survey of popular deep learning tasks applicable to plant phenotyping. Deep Plant Phenomics, a novel software platform for deep learning research in plant phenotyping, is introduced. A model of the *Arabidopsis thaliana* rosette is introduced and it is demonstrated that the use of synthetic data has the potential to mediate some of the issues common to plant image datasets in deep learning. Finally, Latent Space Phenotyping (LSP) is introduced. LSP is a novel paradigm for quantifying response to treatment in plants which requires no hand-engineered pipelines or annotation of training data. The ability of LSP to detect arbitrary visual responses to treatment is demonstrated through five case studies involving both real as well as synthetic data. These case studies show that the method replicates two previously identified candidate loci for drought tolerance in an interspecific cross of *Setaria*, as well as demonstrating the flowering-time dependent drought response of *Brassica napus* L. The flexibility of the previously described synthetic *A. thaliana* model facilitates follow-up discussion where the behaviour of LSP is studied in additional experiments.

The techniques described in this dissertation lay the groundwork for future developments in image-based plant phenotyping, particularly in the use of deep learning, simulation, and latent variable models.
I would like to gratefully acknowledge my supervisor, Ian Stavness, for his guidance during the course of this work. Throughout my PhD he has been a mentor when I needed wisdom, a support when I needed stability, and a friend when I needed to laugh.

Thank you to my committee members – Kevin Stanley, Isobel Parkin, Matthew Links, and my external examiner, Graham Taylor. From the proposal to the final revisions, this work has been made better at every step thanks to their expert direction. Writing an interdisciplinary dissertation required grounding in a range of topics from machine learning to plant physiology and genomics, and it could not have been done without them.

I have had the pleasure to do this work as part of a terrific research group at the Plant Phenotyping and Imaging Research Center (P2IRC) at the University of Saskatchewan. Fellow students, faculty, postdocs, and research staff members at P2IRC and the Department of Computer Science have been instrumental in shaping my research and helping me keep my sanity. In no particular order I would like to thank Jana Ebersbach, Mark Eramian, Ian McQuillan, Tony Kusalik, Steve Shirtliffe, Travis Gray, William van der Kamp, Josh Kocur, Nico Higgs, Theron Cory, Masi Aslaha, Sara Mardanisamani, Kyle Seidenthal, Tewodros Ayalew, Blanche Leyeza, Danny Huang, Donovan Lavoie, Ellen Redlick, Ilya Ovsyannikov, Peggy Anderson, Javier Garcia Gonzalez, Anupama Das, Najeeb Khan, the department staff, and the countless others around this project and department who made my experience what it was.

Much of this work arose from the time I spent as a visiting PhD student at the Biological Modeling and Visualization group at the University of Calgary during the summer of 2017. I would like to thank Przemyslaw Prusinkiewicz (Dr. P) for hosting me at his lab, as well as Mikolaj Cieslak for a long and very successful collaboration. I must acknowledge Mik’s endless patience, having spent most of that time knocking on his office door, asking question after question and forcing him to help me debug my code. Plant modeling is a fascinating topic, and I am delighted to have it featured in this dissertation.

Thank you to my wife, Samantha, and to both of our extended families. They were all with me every step of the way, celebrating my successes and offering comfort and support when there were none to celebrate. It is no exaggeration to say that this dissertation would not have been possible without them.

This work was made possible by financial support from the Canada First Research Excellence Fund and from the Geddes Graduate Scholarship in Computer Science.
To my wife Samantha, for fifteen years of love and support.
CONTENTS

Permission to Use i
Abstract ii
Acknowledgements iii
Contents v
List of Tables vii
List of Figures viii
List of Abbreviations xi
Preface xii

1 Introduction 1
   1.1 Motivation ................................................................. 1
   1.2 Outline and Contributions ........................................... 2

2 Background 4
   2.1 Plant Stress Phenotyping ............................................... 4
   2.2 Image Processing Tools in Plant Phenotyping .......................... 5
   2.3 Field and Aerial Imagery .............................................. 8
   2.4 Deep Learning ............................................................ 9
      2.4.1 Neural Networks .................................................. 9
      2.4.2 Convolutional Neural Networks ................................ 16
      2.4.3 Object Detection ................................................. 18
      2.4.4 Recurrent Neural Networks .................................... 19
      2.4.5 Latent Variable Models ......................................... 21

3 Deep Learning in Plant Phenotyping 24
   3.1 Introduction ............................................................... 24
      3.1.1 Organ and Plant Segmentation and Counting .................. 24
      3.1.2 Stress Detection .................................................. 26
   3.2 Deep Plant Phenomics Software Platform .............................. 27
      3.2.1 Dataset and Tests ................................................ 27
      3.2.2 Approach .......................................................... 29
      3.2.3 Results ........................................................... 30
      3.2.4 Discussion ......................................................... 30

4 Synthetic Data for Deep Learning Using Plant Modeling 36
   4.1 Introduction ............................................................... 36
      4.1.1 Deep Learning and Dataset Shift ................................ 37
      4.1.2 Computational Plant Models .................................... 38
      4.1.3 Related Work ..................................................... 38
   4.2 Methods ................................................................. 39
      4.2.1 CNN Architectures ................................................ 40
      4.2.2 An L-System Model of the Arabidopsis thaliana Rosette ........ 40
   4.3 Results ................................................................. 42
# List of Tables

3.1 Mean (std) absolute difference for the three leaf counting benchmarks. ........................................ 30
4.1 Real and synthetic training datasets. .......................................................... 40
4.2 Augmentation results, Ara2013-Canon dataset. ............................................. 42
4.3 Performance when training and testing on different datasets. Training on a single dataset of synthetic rosettes performs significantly better than training on a dataset of real rosettes with a different distribution of phenotypes. ............................................. 44
4.4 Interoperability between real and synthetic rosettes. ........................................ 45
A.1 Segmentation methods provided by popular image-based phenotyping libraries and platforms. 98
A.2 Object descriptions provided by popular image-based phenotyping libraries and platforms. 98
A.3 Advanced segmentation/localization functionality provided by popular image-based phenotyping libraries and platforms. .................................................. 99
A.4 Imaging types supported by popular image-based phenotyping libraries and platforms. .... 99
A.5 The network architectures used for each of the phenotyping datasets and tasks. All pooling operations are max pooling with a stride of 2. ................................. 100
A.6 Architecture details for the convolutional neural network used in the embedding. All pooling layers are followed by batch normalization. ................................. 100
A.7 Architecture details for the decoder network. All pooling layers are followed by batch normalization. .......................................................... 101
### List of Figures

2.1 Three examples of *Arabidopsis thaliana* rosettes captured in an indoor imaging system [81]. 5
2.2 A graphical representation of a two-layer neural network. .......................... 10
2.3 Some non-linear activation functions used in neural networks: hyperbolic tangent, sigmoid ($c = 1$) and rectified linear unit (ReLU). .......................... 10
2.4 Data points sampled iid from a multivariate gaussian distribution (top), and the corresponding loss surface under linear regression (bottom). .......................... 12
2.5 The convolution operation for one of $K$ filters in a convolutional layer. 16
2.6 A convolutional neural network (CNN). Convolutional and pooling layers act as a feature extractor, and fully-connected layers estimate the output. .......................... 17
2.7 An example of the original R-CNN architecture with a small feature extractor. The region proposals are generated via selective search. .......................... 19
2.8 The structure of an autoencoder model. In this particular example, a volume representing an image is encoded to a latent vector, and then decoded back to an output with the original dimensionality. .......................... 22
3.1 Example images from the A2 dataset showing a range of diversity in image resolution, leaf size and shape, and leaf counts. .......................... 28
3.2 Distribution of count errors for the three leaf counting datasets. The output of the regressor is not rounded in order to produce more granular bins. .......................... 31
3.3 Example training (red) and testing (blue) curves for several benchmark tasks. .......................... 34
3.4 Visualization of filter weights in the first convolutional layer of a leaf counting network. .......................... 35
4.1 Leaf growth and shape functions used in the L-system model. .......................... 41
4.2 Synthetic rosettes (left) generated by the L-system and real rosettes (right) from the public dataset [81]. .......................... 42
4.3 Top view of the synthetic *A. thaliana* model at maximum maturity. .......................... 43
4.4 Distributions of relative count difference in the generalization experiment. Training on one dataset and testing on another exhibits severe dataset shift (top), while training on synthetic data significantly reduces this error by encompassing a comprehensive range of leaf counts (bottom). .......................... 44
4.5 Scatter plots of actual and predicted leaf counts in the interoperability experiments. Training on synthetic and testing on real (left), and training on real and testing on synthetic (right). .......................... 45
4.6 Comparison of training and testing loss on real (red) and synthetic (blue) rosettes. Real plants show significantly higher generalization error, while the synthetic dataset is relatively easy to fit. .......................... 46
4.7 Test performance on purely synthetic data when using increasing sizes for the training set. Like with datasets of natural images, we see that generalization performance improves with larger training sets. .......................... 46
5.1 Overview of the proposed technique. The process consists of three phases, which take place in sequence. First, an embedding process projects images into latent space. Second, a decoder is trained to convert these embeddings back into the input space. Lastly, the decoder is used to calculate a geodesic path between the embeddings for the initial and final timepoints. .......................... 51
5.2 The deep network used in learning an embedding. A CNN takes a sequence of images at various timepoints and feeds outputs to an LSTM, which in turn is used to predict the treatment. The LSTM is removed and the CNN is retained to embed new samples. .......................... 54
5.3 Real images from the *Setaria*, synthetic Arabidopsis, and sorghum datasets (left), and the same images predicted from their latent space encodings by a decoder network (right). .......................... 57
5.4 Distance between embeddings of images of lines at different angles in a hypothetical latent space. Here, the semantic distance is the difference in the interior angle. The Euclidean distance between images in the image space is constant, and the Euclidean distance (dotted arrow) between their encodings in the latent space is evidently not representative of the semantic distance. However, the geodesic path (solid arrow) between images represents the semantic distance well. ................................................................. 58

5.5 Well-watered (left) and water-limited (right) examples of a particular line from the Setaria RIL population [30] .................................................................................................................. 60

5.6 Results for the Setaria RIL experiment. Top left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later timepoints. Top right: histogram of output trait values. Bottom: LOD plot showing QTL for comparison between water-limited and well-watered conditions. ............................................ 65

5.7 Embedding plot (left) and histogram of generated latent distance values (right) for the sorghum nitrogen treatment dataset for the treated (10/10) and control (100/100) conditions. . . . 66

5.8 Well-watered (left) and water-limited (right) examples of a particular line from the B. napus L. NAM population ........................................................................................................... 66

5.9 Result for the B. napus experiment. Left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later timepoints. Right: analysis of the output for the three experimental phases, categorized by flowering time. . . . 67

5.10 Untreated (left) and treated non-resistant (right) synthetic Arabidopsis plants at the final timepoint, showing differences in leaf elevation angle. .......................................................... 67

5.11 Results for LSP on the synthetic A. thaliana model. Top left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later timepoints. Top right: histogram of output trait values. Bottom: manhattan plot showing the simulated causal locus on chromosome 1. The Bonferroni-corrected p < 0.01 significance threshold is shown as a dashed line. ......................................................... 68

5.12 Ablation experiment using Euclidean image distance between each pair of images in the sequence for the synthetic Arabidopsis dataset. The naive solution fails to recover the simulated resistance QTL on chromosome 1. ................................................................. 69

5.13 Result of LSP on the synthetic circles dataset. Top left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later timepoints. Top right: histogram of output trait values. Bottom left: manhattan plot. Bottom right: ground truth growth rate data used to generate the image dataset. ......................... 69

6.1 The effect of encoder architecture on the encoder and decoder loss for the synthetic datasets. Only the moving average is shown on the encoder loss figures for readability. .................. 72

6.2 Histograms of trait values for the synthetic Arabidopsis dataset separated into the five periods between each of the six timepoints. ....................................................................... 73

6.3 Embedding plot for the synthetic A. thaliana dataset (left), and the embedding plot for the same dataset after reducing the temporal resolution by half (right) ........................................ 76

6.4 Trait values for the synthetic A. thaliana experiment without interpolation (left) and with interpolation (right). .................................................................................................... 77

6.5 Trait values for the synthetic circles dataset corresponding to a pathological encoding, without interpolation (left) and with interpolation (right). ......................................................... 78

6.6 Top, left to right: embedding plots for the synthetic A. thaliana experiment with $L_v$ coefficients of 1 (default), 0.1, and 0 (no variance loss). Bottom: corresponding trait values for the same experiments. ................................................................................. 78

A.1 Additional examples of synthetic Arabidopsis rosettes (left) decoded from their latent space vectors (right). ........................................................................................................... 96
A.2 Example images from the Setaria, synthetic Arabidopsis, and synthetic circles datasets (left) and corresponding saliency maps generated using guided backpropagation (right). Intensity is higher for the pixels which have high saliency with respect to the $L_2$ norm of the latent space embedding of the image. The Setaria image is from an experiment carried out without cropping to include the background in the saliency demonstration.
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFI</td>
<td>Chlorophyl Fluorescence Imaging</td>
</tr>
<tr>
<td>CNN</td>
<td>Convolutional Neural Network</td>
</tr>
<tr>
<td>DPP</td>
<td>Deep Plant Phenomics</td>
</tr>
<tr>
<td>ELBO</td>
<td>Evidence Lower Bound</td>
</tr>
<tr>
<td>FCN</td>
<td>Fully Convolutional Network</td>
</tr>
<tr>
<td>GAN</td>
<td>Generative Adversarial Network</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>GWAS</td>
<td>Genome-Wide Association Study</td>
</tr>
<tr>
<td>GxE</td>
<td>Genotype by Environment</td>
</tr>
<tr>
<td>GxM</td>
<td>Genotype by Management</td>
</tr>
<tr>
<td>HTP</td>
<td>High-Throughput Phenotyping</td>
</tr>
<tr>
<td>ICA</td>
<td>Independent Components Analysis</td>
</tr>
<tr>
<td>IPPN</td>
<td>International Plant Phenotyping Network</td>
</tr>
<tr>
<td>L-system</td>
<td>Lindenmayer-system</td>
</tr>
<tr>
<td>LSP</td>
<td>Latent Space Phenotyping</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short Term Memory</td>
</tr>
<tr>
<td>mAP</td>
<td>Mean Average Precision</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>MLP</td>
<td>Multi-Layer Perceptron</td>
</tr>
<tr>
<td>NAM</td>
<td>Nested Association Mapping</td>
</tr>
<tr>
<td>NDVI</td>
<td>Normalized Difference Vegetation Index</td>
</tr>
<tr>
<td>NIR</td>
<td>Near Infra-Red</td>
</tr>
<tr>
<td>OLS</td>
<td>Ordinary Least Squares</td>
</tr>
<tr>
<td>PCA</td>
<td>Principal Component Analysis</td>
</tr>
<tr>
<td>PH</td>
<td>Persistent Homology</td>
</tr>
<tr>
<td>QTL</td>
<td>Quantitative Trait Locus</td>
</tr>
<tr>
<td>R-CNN</td>
<td>Regions with CNN features</td>
</tr>
<tr>
<td>ReLU</td>
<td>Rectified Linear Unit</td>
</tr>
<tr>
<td>RIL</td>
<td>Recombinant Inbred Line</td>
</tr>
<tr>
<td>RGB</td>
<td>Red, Green, Blue</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>RPN</td>
<td>Region Proposal Network</td>
</tr>
<tr>
<td>SGD</td>
<td>Stochastic Gradient Descent</td>
</tr>
<tr>
<td>SNP</td>
<td>Single Nucleotide Polymorphism</td>
</tr>
<tr>
<td>SSD</td>
<td>Single-Shot Detector</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>UAV</td>
<td>Unmanned Aerial Vehicle</td>
</tr>
<tr>
<td>VAE</td>
<td>Variational Autoencoder</td>
</tr>
</tbody>
</table>
Preface

Some of the content of this dissertation is derived from previous publications. Section 3.2 contains material adapted from Ubbens and Stavness, 2017 [136]. I developed the software, performed the experiments, and wrote the manuscript with editorial support from Dr. Stavness.

The contents of Chapter 4 are derived from the previously published work Ubbens, Cieslak, Prusinkiewicz, and Stavness, 2018 [135]. I devised and performed the experiments, while the A. thaliana model was created by modifying an existing model in collaboration with Dr. Cieslak. The contents of Section 4.2.2 are due to Dr. Cieslak. The remainder of the manuscript was written by myself with editorial support from Drs. Cieslak, Prusinkiewicz, and Stavness.

The contents of Chapter 5 are derived from the previously published work Ubbens, Cieslak, Prusinkiewicz, Parkin, Ebersbach, and Stavness, 2020 [134]. I created the proposed method, devised and performed the experiments, and wrote the manuscript with editorial support from all authors. The B. napus dataset was developed by Dr. Parkin. The statistical analyses of the B. napus and Setaria results were performed by Dr. Ebersbach. The modifications to the previously described A. thaliana model are due to Dr. Cieslak.
Over the past 15 years, many authors have proposed that phenomics – large-scale phenotyping – is the natural complement to genome sequencing as a route to rapid advances in biology. The response to these propositions has mostly been silence, implying that ‘phenotyping as usual’ – measuring a limited set of phenotypes that seem the most relevant – is adequate. We disagree and argue that the case for phenomics is as compelling now as the case for genomics was 25 years ago and indeed shares many similarities with that case.

Houle et al., 2010
1 Introduction

1.1 Motivation

Challenged by a growing global population, it is projected that the global food supply must meet the demands of over 9 billion people by 2050 [62]. Meeting this demand will necessitate an accelerated rate of improvement in global crop yields, as by 2050, the rate of demand for cereal crops is projected to surpass the rate of improvement in crop yields. Therefore, the search for genetic gain in crop plants is more important than ever before. One relevant way to improve global crop yields is by enabling tolerance to a wider range of environmental conditions. Not only does this strategy have the ability to recover yields lost to environmental stresses such as excessive heat and limited water in land already used for crop production, but it also has the potential to open up new locations which were not previously used due to problems such as salinity, pH, and nitrogen deficiency.

Discovering the genetic basis underlying tolerance to such environmental stresses is done largely via genotype by environment (GxE) studies. These studies typically involve subjecting a diverse set of genotypes to the environmental treatment in question, and then performing association mapping or a genome-wide association study (GWAS) to find genetic polymorphisms corresponding to a favourable response. The question of quantifying the susceptibility or resistance of a plant to stress is a task for plant phenotyping methods. Phenotyping in this context involves quantifying the physical appearance of the individuals within the population, with the goal of extracting measurements which accurately reflect the degree to which the treatment affects them. For example, this is frequently done using measurements of biomass [64, 30, 29] or colour [138]. By introducing an abiotic stress to a population, heritable differences in tolerance characteristics between genotypes appear. These heritable tolerance responses include such traits as the production of photoprotective pigments, water efficiency through higher leaf water potential, and lifecycle changes such as the earlier production of flowering bodies [14]. Phenotypic indicators are also known to be the best predictors of important biological outcomes, for example, crop yields [49].

Since GWAS studies require relatively large populations in order to achieve the power necessary for finding significance, many have turned to automated plant imaging solutions in conjunction with image-based phenotyping techniques in order to efficiently cultivate and phenotype large populations [28]. Automated indoor and outdoor plant phenotyping facilities such as robotically controlled greenhouses and gantries have enabled the collection of image data at scale, and image processing tools have enabled the automated analysis of the large image datasets produced by these facilities [28]. However, although the size of the datasets is
larger, the scope of the analysis has remained largely unchanged [49]. Although some deep learning methods have been demonstrated in the literature, classical analyses for determining plant stress remain the most common approaches. Both the traditional image-processing techniques as well as the newer deep learning methods are typically designed for applications on a per-dataset basis. Image processing approaches often require the design of a processing pipeline with explicitly tuned parameters for the dataset being analyzed. Similarly, deep learning approaches typically require the tedious and time-consuming manual annotation of training data, for example, specifying bounding boxes around each organ in an image. Critically, these trained models also often suffer from domain shift problems and fail to generalize to other datasets, limiting their practical usefulness [135].

In addition, both image processing and deep learning techniques for plant phenotyping are based around the measurement of some rigidly defined visual concept, such as vegetation area, or organ count. Although these measurements can be based on concrete traits which are known to be correlated with stress responses, they introduce an *a priori* specification of what the plant’s response to the stress is expected to be.

In order to maintain the pace of progress in the discovery of loci contributing to environmental tolerance in crop plants, new phenotyping techniques will be necessary. These techniques may require a new way of thinking about environmental tolerance phenotypes. They will certainly be required to operate in high-throughput contexts and ideally without the need for human intervention such as manual annotation. An even stronger bias away from human intervention would obviate any *a priori* specification of how the relevant phenotype should be measured, fully abstracting the visual concept of the response phenotype.

### 1.2 Outline and Contributions

This dissertation proposes the concept of abstract plant phenotypes derived directly from image datasets. To this end, I demonstrate an abstract concept of stress response in plants through a technique called Latent Space Phenotyping, a latent variable model for phenotyping complex visual responses in plants. This main contribution is enabled by further technical contributions in deep learning and plant modelling in plant phenotyping.

In *Chapter 2*, I review the relevant background on plant stress phenotyping, starting with classical image processing and building up to state-of-the-art deep learning methods.

In *Chapter 3*, I discuss the role of deep learning in plant phenotyping, and introduce a software package I created for this application. Key contributions include:

- The development of Deep Plant Phenomics (DPP), an open-source software platform for facilitating and accelerating deep learning research in plant phenotyping.

- The demonstration of DPP to implement a convolutional neural network for several benchmark phenotyping tasks via global regression.
In Chapter 4, I demonstrate how plant models can be used to generate training data for deep learning
techniques in the absence of large datasets representing a sufficient range of phenotypic variation. A model
of the rosette stage of *Arabidopsis thaliana* is introduced. Key contributions include:

- Synthetic data yielded by the model is shown to improve the results of a leaf counting task when used
to augment training images of real plants.

- The ability of the synthetic model to generate an arbitrary distribution of phenotypes is shown to
mitigate the problem of dataset shift.

- Synthetic data yielded by the model is shown to be significantly interoperable with the real images, as
a regression model is able to generalize between the two types of data.

Finally, in Chapters 5 and 6, I introduce a new approach to phenotyping for genotype by environment
studies, termed Latent Space Phenotyping (LSP), using a learned latent variable model. The proposed
technique is completely automated, and does not require manual design, tuning, or data annotation. The
response to stress is detected automatically using the discriminative power of deep neural networks, whether
this response is a simple difference in color or growth rate or a complex morphological process. LSP has the
potential to enable plant scientists and breeders to discover environmental resistance loci in their treatment
studies without the challenging technical work involved in designing image processing pipelines. It also
enables the study of an abstract concept of the response to stress, instead of extracting various geometric
and reflectance features without knowing which of these measurements, if any, are relevant proxies for stress.
This shift in analysis has the potential to aid in the discovery of new sources of genetic gain, improving crop
yields in diverse environments. Key contributions include:

- The development of the first latent variable model for plant stress phenotyping using images.

- The description of a novel three-stage training process for parameterizing deep neural networks for this
task and quantifying the treatment response in the resulting continuous space. This training process
includes a novel loss term, the *variance loss*.

- Results validating LSP using three natural datasets (*Setaria*, sorghum, *Brassica napus L.* ) and two syn-
thetic datasets, including the previously described *A. thaliana* model from Chapter 4. Two previously
reported QTL for water use efficiency in *Setaria* are re-identified.

Appendix A contains supplementary tables and figures.
2 Background

In this chapter, I provide background information and related work on the main topics relevant to this dissertation, including common plant stress response phenotypes, image processing tools for measuring them, as well as an introduction to deep learning.

2.1 Plant Stress Phenotyping

Plant responses to biotic and abiotic stress are diverse, and many image-based assays have been proposed in the literature with the goal of quantifying these responses. Biotic stresses are defined as sources of stress stemming from a biological source, such as insect damage or disease, whereas abiotic stresses are from other environmental factors such as heat, drought, and nutrient deficiency. Primarily these abiotic stress responses have been measured with image processing methods which quantify the pixel-level differences in intensity or geometric features. One of the most common metrics of abiotic plant stress is biomass, as it is known that many types of stress have an adverse effect on growth rate. In the image processing context, this typically means measuring the vegetation area (the number of vegetation pixels) by segmenting the plant from the background [27, 30]. This is motivated by the finding that vegetation area has been shown to be a viable non-destructive proxy to measuring biomass via destructive sampling [64]. The correlation between abiotic stress and vegetation area has been shown for several abiotic stress sources including nitrogen deficiency [138], drought stress [27, 30, 29], cold treatment [50], and salinity [15]. However, although it is easily measurable and biologically meaningful, stress can have other effects, especially if the stress occurs later in the plant's lifecycle.

In addition to geometric features such as area, there are also image-based techniques which measure pixel intensity values to quantify stress. This can be as simple as measuring the normalized green area (the green quantity divided by the number of vegetation pixels). It can also involve hyperspectral imaging - that is, imaging in wavelengths outside of the standard visible light used to construct RGB images. For plants in particular, chlorophyll fluorescence imaging (CFI) is a common method of measuring the activity of photosystem two (PSII). Images measured using CFI are in a ratio $F_v/F_m$, representing the ratio of variation in fluorescence to the maximum fluorescence. CFI has been used to elucidate the effect of temperature stress on PSII and validated using a proximal chlorophyll fluorometer [50]. CFI has also been shown to be a fast and objective way to quantify biotic stress. By binning vegetation pixels by $F_v/F_m$, Rousseau et al. were able to categorize regions of tissue in leaf images as unaffected, symptomatic, impacted, wilted, or necrotic.
Figure 2.1: Three examples of *Arabidopsis thaliana* rosettes captured in an indoor imaging system [81]

[114]. Although hyperspectral imaging can be especially useful for plants, there is also stress quantification work which has been done using RGB images such as those collected with inexpensive consumer-grade camera systems. In one experiment aiming to detect the presence of iron deficiency stress in soybean, plant segmentation was followed by categorization of pixels into the categories of green, yellow, and brown color [87]. Based on expert domain knowledge, these three categories correspond to healthy tissue, the presence of chlorosis, and necrotic tissue.

While some stress detection applications rely on raw intensity values or geometric features as above, others combine image features with machine learning techniques for analysis. Singh et al. categorize these applications into four categories – identification, classification, quantification, and prediction [122]. In this categorization, classification refers to the categorization of stressed and unstressed plants, or a multi-class classification with multiple classes for multiple levels of stress. Identification is defined as not only identifying the presence of stress, but also the type of stressor (such as a specific disease). Quantification extends classification by predicting not a discrete stress level, but rather a continuous stress value (such as a percentage). Finally, Singh et al. describe prediction as the modelling of factors such as environmental factors and their influence on the presence or level of stress.

### 2.2 Image Processing Tools in Plant Phenotyping

Prior to the introduction of deep learning or other complex computer vision methods to plant phenotyping, there existed an extensive body of research related to image-based phenotyping of plants. This unique intersection between the fields of image processing and plant science has resulted in many collaborative efforts to relieve researchers of the labour-intensive job of manual phenotyping, which can be time-consuming and therefore expensive. In the modern research landscape, image-based methods have largely overtaken manual methods, enabling larger sample sizes and less time between experiment and analysis [57].

Often regarded as the seminal publication in the field of image-based plant phenotyping, Leister et al. [64] introduced the concept of using shoot coverage (the aerial part of the plant) estimated from images as a proxy for measuring plant weight. Foreground pixels were counted on top-down video imagery of *Arabidopsis thaliana* rosettes cultivated in-vitro. This non-destructive technique replaced the conventional procedure of destructively unearthing the plant to measure its weight. The automated imaging system allowed the
researchers to quantify mutations related to growth in different ecotypes. This was an instrumental result in
phenotyping, as it demonstrated that plant mass can be measured in a non-destructive fashion, paving the
way for other image-based techniques. The authors note that occlusion of leaves – where a leaf partially or
completely overlaps another in the image – is a challenging problem when using top-down imagery of rosette
plants.

Since Leister et al. and the inception of image-based phenotyping, many software tools have been de-
veloped by both the academic and commercial communities for the purpose of performing image-based plant
phenotyping, including HTPheno [43], Rosette Tracker (RT) [19], Integrated Analysis Platform (IAP) [57],
PlantCV [27], Image Harvest (IH) [58], and LemmaGrid [39]. These tools have been selected for further review
here due to their widespread use or relevance, although there are yet more tools described in the literature
which we omit in the details below [140, 131]. For the purpose of this survey, we include features present in
both major releases of PlantCV [27, 34]. Although this review focuses on phenotyping shoot traits, there is
an existing review available for root analysis tools [31].

The image analysis tools discussed here were created in support of research activities which require high-
throughput phenotyping, to avoid researchers having to author such tools in-house for each individual study.
Most of these tools have been created to be independent of the imaging system used (with the exception of
LemmaGrid, which is designed for use with proprietary imaging systems), and typically target biologically-
oriented users, providing simple GUI- or scripting-based interfaces for common image analyses. These systems
generally target indoor imaging contexts (as shown in Figure 2.1), and field contexts are discussed separately
in Subsection 2.3. All of the existing image processing libraries and platforms discussed here implement an
image processing pipeline with two distinct stages: a) segmentation, and b) description.

The segmentation phase isolates the plant material in the image from the background. Different libraries
take different approaches to the segmentation problem (Table A.1). All libraries offer some variety of
automatic thresholding function to isolate foreground and background pixels, but the specific algorithm used
varies between them. The simplest algorithm is employed by Image Harvest, which defines plant pixels as
any pixel which has a higher value in the green channel than in the blue channel. Although this method
works appreciably well for specific and simple backgrounds such as soil, it is not a robust solution in a more
general sense. HTPheno and Rosette Tracker both employ more powerful segmentation algorithms based on
multidimensional histogram thresholding (MHT), which is more robust to different backgrounds. This
technique is discussed and evaluated in the context of plant segmentation by Agapito et al. [93]. PlantCV
offers multiple automatic thresholding methods to choose from, including the popular Otsu method [92],
and Triangle thresholding, while Integrated Analysis Platform uses the YEN algorithm [146]. For some
segmentation algorithms, it is advantageous to perform pre-processing as an additional step before automatic
thresholding, in order to improve robustness to different lighting conditions or other imaging factors. Different
libraries offer pre-processing in the form of white balancing or color balancing [28, 57], or decorrelation
stretching [128].
In addition to automatic thresholding, IAP, PlantCV, and Lemnagrid all offer the ability to subtract a pre-defined background image. This one-step technique is simple and effective for highly controlled and static environments, for example, on a stage in an imaging booth. PlantCV and Lemnagrid also offer more advanced segmentation methods based on supervised learning, for more advanced segmentation use-cases such as identifying spots of disease and counting wheat heads in aerial images of fields. These methods are potentially more powerful, although they require hand-annotated ground truth segmentation masks. Supervised learning methods are typically used in instances such as those mentioned previously where the object being segmented is not an entire plant from its background, but rather a semantic segmentation of a particular object where pixel intensity varies slightly between different parts of the same object. These techniques are also easy to extend to multi-class applications where a pixel can belong to more than two classes (i.e. plant or background). PlantCV utilizes a naive Bayes classifier to perform this task \cite{2}, while Lemnagrid uses a boosted trees classifier with (unspecified) texture features\footnote{http://www.lemmatec.com/applications/wheat-ears-segmentation/}.

Following the segmentation step, the object defined as the plant material is analyzed, providing a variety of geometric and pixel-intensity statistics. Again, different image-based phenotyping libraries vary in their capabilities (Table A.2). Among all the examples, every one is capable of measuring the width/height/diameter of the object in question, whether the tool is focused on top-down images of rosettes or side-view non-rosette plants, as well as the shoot area/coverage based on the quantity of foreground pixels.

All tools, with the exception of HTPheno, also calculate the convex hull of the plant. The convex hull is defined by the smallest polygon with the minimum number of vertices which is able to fully enclose the object using only convex angles. From the convex hull, potentially biologically important features can be calculated, such as compactness or solidity, and stockiness. All of these metrics relate the area of the plant to its spatial extent based on the convex hull. Some tools also extract other useful geometric statistics, such as the angle between the leaves and the stem \cite{39}, or statistics of the minimum bounding ellipse \cite{27}.

Some image-based phenotyping tools also provide more advanced functionality for describing the architecture of plants (Table A.3). IAP, PlantCV, and Lemnagrid all provide the location of leaf tips, inferred by skeletonizing the plant region and finding the terminal vertices. This method works well for some species, such as maize, rice, and grasses, but is not designed for rosette plants. IAP and LemmaGrid are also capable of reporting the number of leaves using a similar skeletonizing algorithm, while PlantCV has support for per-leaf segmentation and counting in rosette plants using watershed segmentation.

Finally, many image analysis tools are capable of calculating information about reflectance of the plant in wavelengths outside of visible light (Table A.4). RT, IAP, PlantCV, and Lemnagrid are all capable of reporting the intensity of reflectance of near-infrared (NIR) wavelengths. Additionally, these same packages are all capable of measuring $Fm/Fv$ of Fluorescence, a measurement of photosynthetic efficiency generated by comparing the spectrum of light reflected before and after a high-intensity excitation by fluorescent light.
2.3 Field and Aerial Imagery

Although image analysis tools such as those described in Subsection 2.2 are effective in quantifying characteristics of plants in controlled, indoor conditions, their applicability to field trials is limited. These conditions are often more challenging in an image processing sense than images taken indoors, due to changes in lighting, shadows, multiple plants in close proximity to one another, as well as other factors such as increased movement in the plants due to wind. In outdoor contexts, traits can be measured either with proximal sensing, or via remote sensing such as with Unmanned Aerial Vehicles (UAVs). There is a comparatively small body of literature concerning the application of image-based techniques to specifically outdoor environments, often concentrating on crop performance metrics using remote sensing.

As with indoor imagery, measuring vegetation coverage is an important phenotype for outdoor imagery as well, as it relates to yield [119]. Many different thresholding methods have been proposed to separate vegetation from background, generally termed vegetation indices. Vegetation indices in the RGB colour space include Excess Green Index (ExG), Excess Green Red Index (ExGR), Color Index of Vegetation Extraction (CIVE), Normalized Difference Index (NDI), among many others [42]. Normalized Difference Vegetation Index (NDVI) additionally makes use of reflectance in the near-infrared wavelengths, making it very accurate when hyperspectral information is available [130]. Decorrelation stretching followed by thresholding in HSV colour space was shown to yield accurate results [40], and machine learning methods which operate on multiple colour spaces have also been used [116].

Besides vegetation coverage, other spectral indices derived from remote sensing data have also been used to measure performance of crops in field settings, which is particularly useful for breeding trials [130]. Canopy temperature can accurately be detected in the infrared wavelengths and is useful for measuring transpiration rate and drought stress [10, 41]. Similar to NDVI, Green-Normalized Difference Vegetation Index (GNDVI) was shown to correlate with nitrogen uptake in wheat [66].

One potentially important task for field imagery is estimating the number of a specific organ, such as seed pods, wheat heads, maize tassels, or the number of flowers present in the plot. In Subsection 2.2, it was noted that the Lemnagrid software provides a segmentation feature which is based on supervised classification and demonstrates it on the wheat head counting task [39]. Elsewhere in the literature, a Bag-of-Visual-Words descriptor is used with a support vector machine (SVM) classifier to detect heading and flowering in aerial images of outdoor plots of wheat [118]. Biotic and abiotic stress are also highly relevant in outdoor imaging contexts. In the literature on disease detection, characteristics of the histogram have also been used for quantifying disease damage from aerial photos of potato plots [128]. A review of stress detection, especially that of outdoor aerial imagery, is available in [122].
2.4 Deep Learning

2.4.1 Neural Networks

Parametric Regression Models

As a general goal in machine learning, we wish to learn a function \( f(x) = y \) which maps samples from inputs \( x \in X \) to outputs \( y \in Y \). In machine learning, a parametric model is a family of functions which incorporate a series of function parameters \( \theta \), making the problem \( f(x, \theta) = y \). These parameters are typically learned during an optimization process known as training on samples \((x, y)\) drawn from the underlying distribution, called the training set. The advantage of a parametric model over a non-parametric model is due to the fact that these samples from the training set can be discarded after the model parameters have been determined.

One of the most basic cases of a parametric model is simple linear regression with a vector input. In an arbitrary number of dimensions, a linear regression model can be expressed as a function

\[
f(x, W, b) = x^T W + b
\]

(2.1)

where \( f \in \mathcal{F} \) is a linear function in the hypothesis class of linear functions, \( W \) is a weight vector \((x^T W \) calculates the inner product), and \( b \) is a bias term. A linear relationship between \( x \) and the dependent variable \( y \) can be modelled by determining the model parameters \( W \) and \( b \). In this formulation, \( x \) is assumed to be a \( N \times 1 \) vector (thus the transpose), \( W \) to be an \( N \times M \) matrix, and \( b \) to be a \( M \times 1 \) matrix. Therefore the output space of the regressor is \( M \)-dimensional and a linear regressor can be thought of as mapping an \( N \)-dimensional input vector to an \( M \)-dimensional output.

Two-Layer Neural Network

Although truly linear functions can be modelled well by linear regression as in Equation 2.1, the model fails to capture any type of non-linear relationship. In contrast, neural networks represent a family of parametric models which are able to approximate any function, making them universal function approximators [20]. This capability holds not only for “deep” neural networks, but even a two-layer network containing only one hidden layer and one output layer.

The transition from linear regression to a neural network is mathematically simple. A two-layer network takes the form

\[
f(x, \theta) = W_1(\sigma(x^T W_0 + b_0)) + b_1.
\]

(2.2)

The key differences being the inclusion of a second set of weight and bias parameters \( W_1 \) and \( b_1 \), and \( \sigma \), some non-linear activation function such as a sigmoid function or a Rectified Linear Unit (ReLU), which is applied after the familiar affine transformation (Figure 2.3). A neural network without such an activation
function would be no different than a linear regressor or a linear classifier, due to the fact that a linear transformation of linear transformations of a given space still only represents a linear transformation. Intuitively, the introduction of a non-linear activation function gives the model expressive power by allowing for non-linear transformations of the input space.

![Graphical Representation of a Two-Layer Neural Network](image)

**Figure 2.2:** A graphical representation of a two-layer neural network.

Without relying on formal representations, a graphical representation of the two-layer network is shown in Figure 2.2. The *input layer* represents the dimensions of the input data, and each unit in each layer is connected to each unit in the subsequent layer, giving us the output-by-input matrix size. Not pictured is the activation function applied between the hidden layer and the output layer.

![Activation Functions](image)

**Figure 2.3:** Some non-linear activation functions used in neural networks: hyperbolic tangent, sigmoid ($c = 1$) and rectified linear unit (ReLU).

The ReLU activation function is simply $g(x) = \max(0, x)$. In practice, ReLU is a preferred activation function for deep networks. This is due to two main advantages over sigmoidal activations. First, ReLU
activations are less susceptible to vanishing gradients. The vanishing gradient problem is a numerical problem due to the fact that the first derivative of a sigmoid function decreases asymptotically as it gets further from zero (see Subsection 2.4.1). This means that local error gradients at these parameters will become exceedingly small, making them virtually unaffected by weight updates. This problem is especially pronounced in deep networks with many layers, where there is repeated multiplication of small weight updates. ReLU avoids this problem by having a constant first derivative above zero. However, ReLUs introduce the related problem of dead ReLUs, where a unit is never activated because the pre-activations never put it above zero (usually due to the bias term), and therefore its gradient is always zero. The second advantage of the ReLU unit is that it is generally more expressive when multiple activations are stacked together. For example, a sequence of two ReLUs are able to approximate a sigmoid function, but no combination of sigmoids are able to approximate a ReLU. Other variants have been proposed, such as leaky ReLU which multiplies values below zero with a small constant instead of gating at zero, aiming to improve on the performance of ReLU [76] – however, with a principled initialization and modern training conventions, ReLU is typically used instead.

Optimization

Learning the parameters of any parametric model, be it a linear regression model or a multi-layer neural network, amounts to a problem of finding the optimal parameterization. For neural networks, this involves solving a (typically) non-convex optimization problem which requires searching over the parameter space to find the optimal set of parameters \( \hat{\theta} \). Formally, the optimization objective can be expressed as

\[
\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x_i, \theta), y_i) \tag{2.3}
\]

for all training samples \( x_i \in \{x_0...x_n\} \) and corresponding labels \( y_i \in \{y_0...y_n\} \), where \( \mathcal{L} \) is some scalar-valued loss function which is a metric indicating the performance of the model on the training data. It is assumed that, the pairs \( (x_i, y_i) \) being drawn \( iid \) from the underlying distribution of the data, optimizing the expected loss over these samples is a viable proxy for optimizing over the unseen distribution, a concept known as generalization. The optimization problem posed here can be highly non-convex for many families of functions, and is worth studying briefly here.

For the sake of simplicity, let’s return to the linear regression described in Equation 2.1 with a loss function called ordinary least squares (OLS). Unlike a neural network, linear regression in a single dimension has only two scalar parameters and so can be visualized as a two-dimensional loss surface (sometimes also called an error surface or a loss manifold), allowing us to get an intuitive geometric understanding of the optimization problem. Substituting \( f \) for linear regression and \( \mathcal{L} \) for OLS in Equation 2.3, we get

\[
\hat{W}, \hat{b} = \arg\min_{W,b} \frac{1}{n} \sum_{i=1}^{n} (y_i - (x_i^T W + b))^2. \tag{2.4}
\]

We can visualize this optimization problem by sampling the loss at multiple values of \( W \) and \( b \) and
plotting the loss surface which results from this equation, as shown in Figure 2.4. Examining the loss surface shows that this is a convex optimization problem, gradually decreasing everywhere to a global minimum.

![Figure 2.4](image)

**Figure 2.4:** Data points sampled *iid* from a multivariate gaussian distribution (top), and the corresponding loss surface under linear regression (bottom).

In the case of linear regression, the global minimum can be found in closed form by solving for a set of linear equations. However, as previously mentioned, neural networks operate in a much higher-dimensional space and result in more complicated, non-convex optimization problems. These problems seldom have any analytical solutions, and so are typically approached with a set of numerical techniques called *gradient-based methods*, the preeminent example being *gradient descent*. In gradient descent, the parameter space is traversed from areas of high loss to areas of low loss by iteratively adjusting parameters in the negative direction of the loss gradient, by an amount proportional to its magnitude. Formally, if we consider the loss at the current parameterization $\theta_n$, then the update rule is
\[ \theta_{n+1} = -\alpha \nabla \mathcal{L}(f(x, \theta_n), y) \]  

where the constant \( \alpha \) is a tuneable hyperparameter known as the learning rate. Extending this from a single parameter to multiple parameters is simple, as we just have to calculate partial derivatives for each of the parameters with respect to the loss. For example, returning again briefly to our linear regression optimization problem from Equation 2.4, we calculate

\[ \frac{\delta \mathcal{L}}{\delta W} = \frac{2}{n} \sum_{i=1}^{n} -x_i (y_i - (x_i^T W + b)) , \]

\[ \frac{\delta \mathcal{L}}{\delta b} = \frac{2}{n} \sum_{i=1}^{n} -(y_i - (x_i^T W + b)) \]

adjusting each one separately according to their own gradients. Although vanilla gradient descent is useful, it is also subject to multiple drawbacks. For example, as the current state grows closer to the minimum, the rate of convergence decreases asymptotically. Additionally, there is a very high probability that it will get stuck in local minima, as the loss gradient in every direction will direct it back to this point. Non-convexity also means that the optimization will be hung on saddle points in the loss surface, where the gradient in all directions is zero, but the gradient would reappear in one dimension with a slight movement one way or another.

A common extension to gradient descent to solve some of these issues is stochastic gradient descent (SGD). In SGD, instead of considering gradients averaged over every training example, we calculate the gradients for just a small set of points, called batches. This effectively “jitters” the loss surface from one batch to another, which offers a protective effect against local minima, as local minima will appear and disappear from one batch to another as the surface changes. There also exist many extensions of SGD which introduce second-order moments in order to modulate the process of convergence, such as Momentum [107], Adagrad [25], Adadelta [147], and Adam [55]. Momentum introduces a concept similar to physical momentum, where the size of the update is a function not only of the learning rate and the current gradient magnitude, but also of the size of previous updates. This involves keeping track of a new “velocity” variable \( v \):

\[ v_n = \gamma v_{n-1}, \]

\[ \theta_{n+1} = -(v_n + \alpha \nabla \mathcal{L}(f(x, \theta_n), y)) \]

where the hyperparameter \( \gamma \leq 1 \) is a tuneable constant called the momentum. This new expression is slightly modified from 2.5, adding a portion of the previous update’s magnitude to the current update. Although this does mean that Momentum has the possibility of over-shooting the minimum in extreme cases, it can also greatly increase the rate of convergence as it allows the learning rate to “accelerate” and “decelerate” based on previous updates.

So far we have discussed how gradient-based methods are able to traverse the loss surface for optimization problems where the parameters do not act on each other, as in Equations 2.4 and 2.6. However, if we try to
expand this idea to the two-layer neural network, we encounter an issue in computing the local gradients at the network parameters. This problem can be solved by an algorithm called backpropagation. Backpropagation handles the flow of gradients by introducing intermediate equations and simply applying the chain rule from calculus to calculate the local loss gradient at every parameter in reverse order. For example, one can consider a hypothetical loss function (with no parameters) \( \mathcal{L}(x, y) = l = ((x - y) + 5)^2 \). We can then introduce some intermediate variables \( h = (x - y), j = h + 5, \) and \( k = j^2 \). In this case, we can calculate the gradient of the loss with respect to the inputs as \( \frac{\delta l}{\delta x} = \frac{\delta l}{\delta k} \frac{\delta k}{\delta j} \frac{\delta j}{\delta h} \frac{\delta h}{\delta x} \). In this way, we can calculate the local gradients all the way from \( l \) up to the input \( x \), and at every intermediate variable in between.

In practice, calculating gradients at each variable is vectorized by calculating the Jacobian at each function in the function composition. The Jacobian matrix for a vector-valued function is a matrix of first-order partial differentials with respect to the input and output vectors. For example, considering each operation (e.g. activation functions, matrix multiplications, etc.) to be a function \( f_i, i = 0...n \), we can calculate the Jacobian \( \frac{\delta f_i}{\delta f_{i-1}} \) at each \( i \) and multiply these matrices together (backwards, for computational reasons, from the loss to the inputs) to calculate gradient values at each parameter.

**Regularization**

For a given optimization problem such as the ones discussed previously, including neural networks as well as non-linear regression models, there may be many candidate parameterizations \( \theta \), some resulting in small loss values on the training data. However, it is not necessarily true that all of these parameterizations which allow the model to closely fit the training data will also result in good generalization to new, unseen data. Some solutions may be too tailored to the training samples, resulting in high error due to variance in the inputs, also called overfitting. Conversely, other parameterizations may cause the model to ignore meaningful signal in the data due to the model’s bias, also known as underfitting. The balance between these two sources of error is referred to as the bias-variance tradeoff. Regularization methods allow us to indirectly control the tradeoff between bias and variance, by adding extra forces to the objective function or training process which shape the distribution of the parameters. Many such techniques have been proposed [59].

By way of example, we can add regularization to our regression problem by adding another term to the loss function, called a regularization term. For example, generalizing the linear regression problem in Equation 2.4 and adding an \( L_2 \) regularization term, it can be amended to

\[
\hat{W} = \arg\min_W \frac{1}{n} \sum_{i=1}^{n} \left[ (y_i - (x_i^T W))^2 \right] + \left[ \lambda \|W\|_2^2 \right]
\]  

(2.8)

where \( \|W\|_2^2 \) is the sum of the squares of the weight vector \( W \), and \( \lambda \) is a tunable hyperparameter which controls regularization strength (called a regularization coefficient), and therefore the trade-off between overfitting and underfitting. This effectively prevents disproportionately large weights from emerging, which could potentially make the model too sensitive to a given input dimension. It is clear from examining Equation 2.8 that the optimal \( \hat{W} \) under these constraints may not necessarily be the one which minimizes the
squared error term the best, so long as the overall loss value is offset by the regularization term. This type of regularization is often also called $L_2$ weight decay, due to the fact that the magnitude of the parameters tends to decrease under this type of regularization, or ridge regression in the statistics literature. Other regularization terms can also be used, among which the $L_1$ norm $\|W\|_1$ is one of the most popular (also called LASSO in statistics). Unlike $L_2$ regularization, $L_1$ regularization encourages sparsity in the weights (i.e. a smaller proportion of non-zero values). Both $L_1$ and $L_2$ terms can also be used at once, resulting in a less common regularization scheme known as elastic net.

The other type of regularization which is commonly used, especially in deep neural networks, is stochastic regularization. The term stochastic refers to a random process being applied during training, in contrast to a deterministic method such as $L_2$ regularization. Among stochastic regularization techniques, the most prominent example is Dropout [126]. Dropout layers are inserted between (typically fully-connected) layers in the network. For each batch during training, in each Dropout layer in the network, a random mask is generated where some units in the layer are set to zero with some probability $p$ (called the dropout rate). Therefore, the “dropped” units are effectively removed from the model during both the forward pass as well as the backward pass. Intuitively, by removing these units for each training batch, a randomly sampled sub-network is being trained independently of the whole model. During inference, we combine all of these sub-sampled networks into one by simply scaling each unit by the constant $p$. Dropout works by preventing complex co-adaptation of units between successive layers in the network, since each Dropout operation removes a random set of units. This means that the set of input variables to a layer change each time a new sub-network is trained.

Classification

Since multi-class classification is a common task in machine learning, we can think of the $M$ output dimensions of our model as the logits for each of the candidate classes. For example, a five-class classification problem ($M = 5$) maps inputs $x$ to a vector $y \in \mathbb{R}^5$, where the magnitude in each dimension represents an unnormalized probability of $x$ belonging to that class. Performing a classification task requires rethinking the loss function ($\mathcal{L}$). While the distance (or squared distance) between two vectors is an intuitive loss function for the regression case, the output of a classification model is less straight-forward. Using 0-1 loss (zero if the logit corresponding to the correct class is 1, and one otherwise) does not result in the smooth loss surface which is needed to calculate gradients. For this purpose, we introduce the softmax activation function and the cross-entropy loss function.

For classification purposes, the output layer (the final layer in the network) is modified by adding a softmax activation instead of the typical linear activation used in regression problems. Softmax can be expressed as

$$f_{\text{softmax}}(z_j) = z'_j = \frac{e^{x_j}}{\sum_{m=1}^{M} e^{x_m}}$$  \hspace{1cm} (2.9)

where $z_j$ is the $j^{th}$ value in $z$, for all $j$ in $z$ where $z$ is an $M$-dimensional vector of logits. Therefore, the sum
of the outputs after softmax are normalized to one and can be thought of as class probabilities instead of unnormalized class scores. We can then use the cross-entropy loss function to assign a loss value to this set of probabilities by taking the negative log of the probability of the correct class

\[ \mathcal{L}(\mathbf{z}', y') = -\log(z'_{y'}) \]  

(2.10)

where \( y' \) is the index of the correct class. Therefore we gain a smooth loss function which approaches zero as \( p(y|x) = y' \) approaches one, and blows up asymptotically as it approaches zero.

### 2.4.2 Convolutional Neural Networks

Although regular fully-connected neural networks such as those described in Section 2.4.1 are good function approximators, their usefulness is often limited in practice. For visual problems such as the image-based phenotyping problems discussed in this dissertation, fully-connected networks do not take into account the high degree of local connectedness present in image data. Intuitively, pixels which are located close together spatially are very likely to be part of the same object or part in the image, semantically speaking. Convolutional layers and convolutional neural networks (CNNs) are an extension to feed-forward networks which helps capture these local patterns. Not only is this an important consideration for image data, but also in other application domains such as in audio, genomic, and time-series data [63]. All of these types of data can benefit from learning features of local patterns, either spatially, or temporally. Here temporally refers to local structure within a fixed period of time, not reasoning across multiple time windows, an issue discussed further in Section 2.4.4.

![Figure 2.5: The convolution operation for one of \( K \) filters in a convolutional layer.](image)

Unlike those in fully-connected layers, where the output of the unit is a non-linear function of every input, the parameters in convolutional layers act only on one local area in the input at a time, sometimes called a receptive field. This transforms the optimization process in these layers from a global problem to a local one, which serves to capture local patterns in the input. In any given convolutional layer, there exists an arbitrary number of filters \( K \). Each of these \( K \) filters operates over some spatial extent (for image data this corresponds to height and width), and the whole depth of the input (for the first convolutional layer, this corresponds to the colour channels of the image). Convolutional layers typically contain far fewer parameters...
than fully connected layers for the same input. For example, an initial convolutional layer following the 3-channel input image with a receptive size of $5 \times 5$ will contain $5 \times 5 \times 3 \times K$ parameters.

At each position in the input volume, the convolution operation calculates the dot product of the filter parameters with the values at that filter position in the image. This process is repeated for every position in the input volume (or every second position when used with a stride of two pixels, etc.), for every filter in the convolutional layer. This results in an output volume with depth $K$, where each depth slice corresponds to the activation map of one filter. Finally, an activation function (such as ReLU, see Section 2.4.1) is applied element-wise to every pre-activation in the volume. Figure 2.5 shows an example of the convolution operation working over an input volume to produce a depth slice of the output volume.

![Convolutional Neural Network](image)

**Figure 2.6:** A convolutional neural network (CNN). Convolutional and pooling layers act as a feature extractor, and fully-connected layers estimate the output.

Performing convolutions at the borders of a volume requires special consideration. A filter centred on a pixel at the edge of the input volume will overlap the edges, where the values are undefined. One common solution is to ignore positions in the image where this happens, which results in a slightly smaller output volume since some number of pixels around the edge of the image are ignored. The other solution is to add padding to the edges of the image so pixels outside the spatial extent of the volume are no longer undefined. This can be done with either replication padding or zero padding. This way, the spatial size of the output volume matches that of the input volume.

Another common operation in CNNs is the pooling operation which takes place in pooling layers. This is a simple spatial downsampling operation, using either a maximum filter (max pooling) or a mean filter (average pooling) with a stride size greater than one. Like the convolution operation, the pooling operation will have a particular spatial size in height and width, but unlike a convolutional operation it operates over each depth slice independently. Assuming a stride size of two, this means that the spatial resolution (height and width) of the input volume will each decrease by half after the pooling operation, while the depth will remain constant. This results in a output volume with only 25% of the spatial information. However, since information is not averaged over the entire input but only in local regions, far less information is lost.

Pooling serves to limit overfitting by downsampling the dimensionality in a relatively non-destructive
way, as well as limiting the number of parameters. Particularly in the case of the initial fully connected layer, an input volume with the original spatial dimensions of the input image would introduce a massive number of parameters in this layer. For example, a fully connected layer with 1000 units and an input size of $128 \times 128 \times 96$ (reshaped into a vector) would require over 1.5 billion parameters. Pooling also condenses spatial information which is distributed over a large area into a smaller area which is accessible within the receptive fields of individual filters.

Following a sequence of convolution and pooling operations, the final output volume is stretched into a $1 \times (HWK)$ vector and acts as input to a standard feed-forward neural network comprised of fully connected layers (Figure 2.6), which proceeds normally to perform classification or regression on the output. The convolution and pooling layers are therefore sometimes referred to as the architecture’s feature extractor.

### 2.4.3 Object Detection

One common task of interest with image data is the detection of objects within an image. While the general-form CNN described in Section 2.4.2 is capable of classifying the contents of an image, this is only relevant for images which contain a single object. In order to detect multiple discrete objects of various classes, specific extensions of the vanilla CNN have been developed. These techniques can generally be categorized as Single-Shot Detectors (SSD) [72, 109], or region-based methods (R-CNN) [36, 111].

Single-shot methods combine the processing of the image into a feature representation and the detection of objects into a single end-to-end training objective. In order to do this, the image is divided into equally sized regions using a grid. At each grid position, the detector predicts a dense array of anchor boxes of various sizes and aspect ratios for each object class. Non-maximal suppression can then be used to find the strongest detections within local neighbourhoods. Single-shot methods are recognized as among the fastest object detection meta-architectures, making them appropriate for applications where the speed of detection is an important factor.

In contrast to single-shot methods, region-based techniques for object detection consist of a two-stage process – one component creates region proposals, defining areas of the image which are likely to contain an object, and the other classifies the contents of each region proposal to identify the class of object within it. The original meta-architecture for this purpose is simply called Regions with CNN features, or R-CNN [36]. Figure 2.7 shows an example of the R-CNN pipeline.

The original R-CNN used an existing image processing method called selective search for finding region proposals within the image. After the region proposals are detected, the respective regions are reshaped into a consistent size and run through a standard CNN for classification. Since the publication of R-CNN, however, the method has been extended by various authors. One of the most popular R-CNN based methods is Faster R-CNN [111]. Faster R-CNN modifies the standard R-CNN meta-architecture by replacing the image processing based selective search method for generating region proposals with a region proposal network (RPN). The image is processed by a fully-convolutional network (FCN), and the RPN generates region
proposals from this learned representation. Instead of reshaping an area for input to the classifier, a new technique called region of interest pooling, or RoI pooling, is used, allowing the classification network to operate directly on the features learned by the FCN. Since both the classification network as well as the RPN use the same learned representation, they can be trained simultaneously in an end-to-end fashion. Integrating region proposals as part of the network not only performs better than selective search, but eliminating the reliance on the external module also makes training and inference faster. Most applications in plant phenotyping favour region-based methods such as Faster R-CNN over single-shot methods because they tend to perform better with small, densely clustered objects, although there is also evidence that RoI pooling is not well-suited to small objects [26].

2.4.4 Recurrent Neural Networks

Although CNNs are a powerful tool for modelling local structures in data with spatial information, both feed-forward neural networks and CNNs have no built-in capacity for modelling temporal or sequential structure. It is for this purpose that recurrent neural networks (RNNs) and the more modern long-short term memory networks (LSTMs) were proposed [115, 47].

RNNs are a popular tool for time series, video, and natural language problems, for which sequence is an important factor. While problems involving sequence are sometimes modelled using methods such as hidden Markov models and Markov chains, RNNs have more capacity for modelling complicated sequential relationships and impose fewer assumptions as part of the model (e.g. the Markov property). Briefly, RNNs maintain an internal state which is updated through the sequence, allowing them to incorporate information about the past into the current time point. Formally, instead of considering a particular input vector \( \mathbf{x} \), we consider a sequence of input vectors \( \{ \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T \} \). An additional parameter \( h_t \) representing the hidden internal state of the RNN at time \( t \) is updated for each new timepoint by
\[ h_t = \sigma \left( W \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \right) \]  

(2.11)

where \( \sigma \) is an element-wise activation function, and \( W \) is a learnable parameter. The initial hidden state can either be set to zero, or considered as a learnable parameter. In this way, the current hidden state is a non-linear function of the current timepoint as well as the previous hidden state.

While RNNs enable the modelling of sequential information, the model of memory which they incorporate is insufficient in many cases. In particular, RNNs suffer from catastrophic forgetting, meaning that the state variable tends to not retain information from time steps before the previous one. This information is known as long-range structure, and in theory RNNs are capable of packing this information into the state variable to be retrieved by time steps in the distant future. However, RNNs tend to not preserve long-range structure in practice [6]. LSTMs are an extension to RNNs which incorporate a more complicated internal state which is capable of selectively retaining information about the past [47]. The memory of an LSTM is stored in a memory cell \( \{C_0, C_1, \ldots, C_T\} \). The cell can add or remove from the current state, mediated by “gates” which control the process of selectively remembering and forgetting information over time.

At time step \( t \), vectors \( f_t \) and \( i_t \) are calculated to determine which information should be removed from the cell and what information should be added to the cell. These vectors are given by

\[
\begin{align*}
  f_t &= \sigma \left( W_f \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_f \right), \\
  i_t &= \sigma \left( W_i \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_i \right)
\end{align*}
\]  

(2.12)

where \( W_f, W_i \) and \( b_f, b_i \) are weight and bias parameters, and \( \sigma \) is the sigmoid function. Next, it is determined what candidate values should be added to the cell by calculating

\[
\tilde{c}_t = \tanh \left( W_c \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_c \right) 
\]  

(2.13)

where \( W_c \) and \( b_c \) are weight and bias parameters. The cell is multiplied by \( f_t \) to forget information, and then the new values from \( \tilde{c}_t \) are added, modulated by the update mask \( i_t \):

\[
c_t = f_t c_{t-1} + i_t \tilde{c}_t
\]  

(2.14)

and the current state \( h_t \) is output as

\[
h_t = \sigma \left( W_o \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_o \right) \times \tanh(C_t) 
\]  

(2.15)
LSTMs have provided state-of-the-art results in many different learning tasks involving sequential data. LSTMs have also appeared in the plant phenotyping literature, demonstrating that they are able to successfully learn a model of temporal growth dynamics in an accession classification task [129]. LSTMs have also been used as a model of spatial attention in leaf segmentation [113].

2.4.5 Latent Variable Models

A latent variable model seeks to explain variance in observations $X$ through modelling unseen latent factors $Z$. Traditional methods for latent analysis of continuous data include factor analysis and partial least-squares regression (PLSR). In general, a latent variable model assumes that $X$ is derived from a generative model

$$
\begin{align*}
& z \sim p(z) \\
&p_{\theta}(x|z) = \psi(x|f_{\theta}(z))
\end{align*}
$$

(2.16)

where $\psi$ is some observation model, and $f_{\theta}$ is a decoder or generator parameterized by $\theta$.

Many deep learning models are concerned with learning latent factors, and these models are often referred to as Deep Latent Variable Models (DLVMs). Perhaps the simplest and most well-known latent variable model in deep learning is the deep autoencoder. An autoencoder is a latent variable model which takes as input some sample $x \in X$, and outputs a reconstruction of the input $\hat{x}$. It does this by compressing the input to a lower-dimensional space called the latent space in a process known generally as encoding. The encoding is of dimensionality $d$, called the intrinsic dimensionality. Encoding is followed by a transformation of this $d$-dimensional latent representation, termed $z$ (sometimes called a code), back to an output with the same dimensionality as the input in a process known generally as decoding. Both the encoding ($p(z|x)$) and the decoding ($p(x|z)$) are learned functions. The encoder and decoder are trained in an end-to-end fashion by minimizing the difference between the input and the output, known as the reconstruction loss. In this way, the model learns a marginal distribution over the observed data.

When an autoencoder is trained using linear models for the encoder and decoder and mean squared error (MSE) as the loss function, the autoencoder reduces to an approximation of principal components analysis (PCA), although the weights are not guaranteed to be orthogonal. However, when used with non-linear models such as feed-forward and convolutional neural networks, autoencoders are able to perform non-linear dimensionality reduction onto a lower-dimensional manifold. Figure 2.8 shows an autoencoder architecture which is designed for image data, using convolutional and pooling layers for the encoder and convolutional and transposed convolutional (learned upsampling) layers for the decoder.

The goal of the autoencoder is to learn a latent distribution over the inputs which is able to explain the data by encoding enough information about it to approximately reconstruct samples from it. This new, lower-dimensional latent space representation can be used in place of the original data for various tasks such
as classification and information retrieval.

Although the autoencoder’s marginal distribution over observations fits the data, the autoencoder in general does not define a generative model.\footnote{However, it has been demonstrated that a variant called denoising autoencoders (DAE) are able to be cast as an energy-based model which allows operations such as sampling \cite{139}.} This is because calculating \( p(\mathbf{x}) \) exactly would require integrating over

\[
p(\mathbf{x}) = \int p_0(\mathbf{x}|\mathbf{z}) p(\mathbf{z})
\]

which is intractable since it requires calculating the full joint distribution over \( X \) and \( Z \). Variational AutoEncoders (VAEs), on the other hand, allow us to define a generative model over data \cite{56}. The name of this technique comes from the field of variational inference, which aims to model a complex posterior distribution through the process of optimizing the divergence between the distribution of latent variables \( p(\mathbf{z}|\mathbf{x}) \) and a variational distribution \( q(\mathbf{z}) \). The objective function of a VAE can be expressed as

\[
\mathcal{L}_{VAE} = \log p(\mathbf{x}|\mathbf{z}) - D_{KL} [p(\mathbf{z}|\mathbf{x})||q(\mathbf{z})]
\]

where the first term is the maximum likelihood estimate of the data, modelled via the reconstruction loss from the decoder, and the second term is the Kullback-Leibler divergence between the prior \( q(\mathbf{z}) \), which we define as a unit Gaussian, and the latent distribution \( p(\mathbf{z}|\mathbf{x}) \). The derivation of Equation 2.18 is given in Section 6.8.

Because both \( p(\mathbf{x}) \) and \( q(\mathbf{z}|\mathbf{x}) \) are modelled as Gaussians, we can solve the second term analytically as

\[
D_{KL} [q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] = \frac{1}{2} \Sigma \left( \exp(\Sigma(\mathbf{X})) + \mu^2(\mathbf{X}) - 1 - \Sigma(\mathbf{X}) \right)
\]

meaning that we can model the distribution of latent representations as a simple Gaussian. In this case, the encoder outputs two separate vectors, one for mean and one for (log) variance. Then, a sampling operation samples the latent vector from this multi-dimensional Gaussian parameterized by the mean and variance calculated from the encoding process. During training, the learned distribution is forced as close as possible
to the unit Gaussian (with zero mean and unit variance) by penalizing the divergence between the two distributions.

Since the sampling operation is a stochastic node, it is impossible to backpropagate gradients through it. This effectively prevents training as the encoder cannot receive gradients during the backwards pass. To address this issue, VAEs use a modified sampling method termed the *reparameterization trick*. The purpose of this method is to separate the stochastic part of the sampling operation into a separate node, which is not back propagated through, but allowing the gradient flow to continue through the latent mean and variance vectors. In order to do this, the sampling operation samples from a unit Gaussian, and then this sampled point is added to the mean vector and then multiplied by the variance vector.

The advantage of VAEs compared to regular autoencoders is their ability to control the distribution of latent representations \( Z \sim q \). This is desirable due to the fact that an autoencoder with no variational encoding will typically spread samples thinly over the latent space. This means that operations such as sampling the latent space and interpolating between points in the latent space are more difficult, as regions of the latent space will be sparsely defined and will not yield semantically meaningful outputs. The latent space defined by a VAE, on the other hand, consists of a multi-dimensional Gaussian compactly clustered around the origin. To draw a meaningful sample from the learned distribution, one only has to sample from a standard unit Gaussian. The trade-off is the restricted representational power resulting from the variational encoding. Since codes must be normally distributed, there is a wide range of the latent space which is unused. For this reason, VAEs often require a higher intrinsic dimensionality than autoencoders to recover this lost capacity and reach a similar level of reconstruction accuracy.
3 Deep Learning in Plant Phenotyping

3.1 Introduction

Due to its recent dominance in popular image-based problems such as image classification, semantic segmentation, instance segmentation, and object counting, deep learning has experienced an explosion in popularity in the plant phenotyping field in recent years. Although its applications in the literature to date are mostly in plant and organ counting, it has also been applied to the detection of plant stress [123].

3.1.1 Organ and Plant Segmentation and Counting

One of the fundamental problems in image-based phenotyping is segmenting and/or counting the leaves, fruit, heads, tillers, pods, or other organs from an image. This is done with the goal of obtaining two biologically relevant phenotypic measurements: organ count, and organ growth trajectory over time. Unlike phenotypic measurements based on geometric descriptions or pixel intensity, these phenotypes are often more complex functions of the image. For example, leaf counting is relatively simple in instances where a skeleton can be constructed from the object and where leaf tips can accurately be represented by apex vertices, such as is the assumption in some image processing tools. However, this is often not the case for rosette plants, which are characterized by (often overlapping) leaves radiating from the centre of the plant. The phyllotactic angle (the angle between consecutive leaves) varies slightly, creating different arrangements, including smaller leaves which are completely surrounded by a lower leaf. In cases such as these, the creation of a skeleton is difficult and other methods must be considered. To counteract these cases, PlantCV [27] implements leaf segmentation and counting for rosette plants by using watershed segmentation to distinguish between leaves. However, there is a demonstrated potential to improve on such techniques using deep learning methods [136, 113].

The simplest way to obtain organ counts from an image using deep learning is to view it as a global regression problem using a CNN (as in Figure 2.6), where the input to the CNN is the image and the output is a single scalar value representing the number of organs. Leaf counting via such a regression method has been done on Arabidopsis thaliana leaf counting benchmarks, with strong results [136, 23]. Although phrasing the organ counting problem as a single-valued global regression problem provides an easy objective and very fast convergence, it is not without drawbacks. The main disadvantage of such a method is its susceptibility to suffering from both covariate shift, where the input distribution changes between training and testing data, or dataset shift, where the joint probability between the inputs and outputs changes. For example, training
such a model on the leaf counting task using one dataset and testing on a dataset with a disjoint distribution of leaf counts can result in severe under-performance [135].

An alternative to counting by global regression is counting via local regression, where local counts are obtained over patches of the image. In these cases, the training ground truth is typically provided as a density map in which each instance is represented by a gaussian density function. Since the total density for each instance annotation sums to one, a global count for the image can be obtained by simply integrating over the output from all image patches, regardless of whether or not any particular patch contains a fraction of an instance, or multiple fractional instances [91]. Although it is potentially more robust to domain shift than a global regression problem, local regression may struggle to distinguish between overlapping organs. Additionally, organs such as leaves which vary widely in size may pose a problem as it is difficult to distinguish the surface of a large organ as representing the same amount of density as that of a smaller one.

Instead of outputting a density map, the same local regression problem can also be phrased to output a binary segmentation map. In the organ segmentation problem, the organ count is obtained for free, a technique which can broadly be described as counting by segmentation. Counting by segmentation is also more robust to the mentioned shifting problems of global regression-based methods, due to the fact that an arbitrary number of objects can be isolated from a single image. A similar technique can be used where the image is first subdivided into superpixels, and then classification is done on these superpixels [117]. If organs do not overlap, then a semantic segmentation method can be used and the organs can be counted independently as connected regions in the image. In the case where organs do overlap, such as in the leaves of rosette plants, a (typically more complex) instance segmentation method can be used [113, 5]. Unlike using a single regression head, this means that an image is not mapped directly to a scalar output, but rather output as a masking image where classification is performed on a per-pixel basis, making the inference a local problem, as with local regression.

The main disadvantage to such counting by segmentation methods is that they are more computationally expensive in both model training as well as at inference time, due to the problem being composed of multiple local problems instead of one global calculation. Additionally, they are not immune to dataset shift, as changes in structure or colour in the organs between the training and testing data can still cause under- or over-detection of image regions. Such counting by segmentation methods have been applied to detect and localize ear tip, leaf tip and root tip areas in images of winter wheat in one of the original deep learning papers in image-based plant phenotyping [96]. The method employed was a simple patch-wise classification using a sliding window and a CNN. A follow-up study to this work additionally located wheat spikes and spikelets [97]. Unlike in the initial study, these experiments were performed using a stacked hourglass network which was able to perform both classification tasks in parallel [90]. Both studies report accuracy in excess of 90%, indicating strong performance of deep learning on these datasets. Similar techniques have been applied to outdoor imagery, by doing local regression to count tassels in field images of maize, with state of the art results [75]. This implementation uses a CNN to perform local count regression in patches over a density map.
Similarly, panicles have been successfully segmented from outdoor plots using a combination of superpixel methods for region proposals and a CNN for binary classification of these regions [144].

A third method of image-based counting is *counting by detection*. In this paradigm, object detection is performed over the image to detect individual instances of the object. The object count is then given by the sum of the detected objects. In recent literature, object counting by detection has been shown to be effective for counting sorghum and wheat heads [35, 77]. In one case, the state-of-the-art RetinaNet object detection system [69] was used to detect sorghum heads in high-resolution aerial images. The authors compare the counting results using this method to a previously reported result using standard classifiers on classical image processing features extracted from image regions [35]. The authors find slightly reduced performance on one dataset ($R^2 = 0.82$ vs. 0.84), but significantly improved performance on the other ($R^2 = 0.76$ vs. 0.56). In a similar study in wheat, the authors used an earlier object detection system known as Faster R-CNN to detect wheat heads in field conditions [77]. The authors compare this result to the previously discussed TasselNet architecture, which is a density estimation-based counting method. The authors find that Faster-RCNN significantly outperforms TasselNet, particularly in cases where the wheat plants have senesced. As opposed to detecting organs, R-CNN based methods have also been used to detect seedlings in field environments for purposes such as emergence counting [51].

All deep learning methods, whether counting by global regression, counting by local regression or segmentation, or counting by detection, should be viewed in the context of image processing results for similar problems. Segmentation has a long history in the image processing literature, and has seen applications to both indoor and outdoor plant phenotyping problems. For example, counting wheat heads in images of outdoor plots has been addressed previously by using Bag-Of-Visual-Words with an SVM classifier [116], resulting in performance in excess of 95%. The PlantCV package includes a semi-automatic segmentation function based on naive Bayes and pixel intensity values [2]. Ideally, the evaluation of deep learning techniques should be relative to image processing baselines in order to justify the additional computational cost, data acquisition cost, systemic complexity, and often poorer interpretability as compared to classical image processing techniques [136, 118].

### 3.1.2 Stress Detection

A majority of the work on stress detection using deep learning techniques has been on the topic of disease and pest detection [123]. Deep learning can be useful in this task because some pathologies result in not only uniform differences in colour due to senescence, but also localized patterns of symptoms. For example, a plant may develop lesions or pustules as part of its disease, which are surrounded by circular patterns of chlorosis or flecking [86]. Although it is often possible to determine these different regions by more traditional methods such as histogram analysis [128] or naive Bayes classification [2], deep learning approaches have the potential to outperform these classical methods by incorporating learned representations of disease structures. Typically, disease detection is performed using images of leaves or entire plants. One study evaluated the use of
two different CNN architectures for classifying the species-disease pair for images of diseased leaves, a total of 38 different classes [83]. Performance on this task was reported to be in excess of 99% in terms of classification accuracy. Another evaluation on the same dataset with a different collection of architectures reported performance between 80% and 90.4% [141]. Another study specialized in tomato plants (S. lycopersicum), and aimed to detect both disease as well as pests [32]. This study took a different approach to others, which usually approach disease detection as a classification problem. The authors instead phrased the problem as a detection and localization problem (Section 2.4.3), testing multiple localization techniques including Faster R-CNN [111], Single-Shot Detector (SSD) [72], and a region-based fully convolutional network (a fully convolutional variant of Faster R-CNN [18]). The authors report a peak mean average precision (mAP) of 0.8598 on their own dataset.

3.2 Deep Plant Phenomics Software Platform

To encourage the progress of deep learning methods in plant phenotyping, we developed the open-source Deep Plant Phenomics (DPP) deep learning platform [136]. The global regression functionality of the package was initially demonstrated on the PRL dataset [81] for leaf counting in Arabidopsis thaliana rosettes, outperforming previous, hand-crafted methods.

3.2.1 Dataset and Tests

For the three experiments presented in this section, the PRL image-based plant phenotyping dataset was used [81]. This dataset includes multiple computer vision benchmarks for tasks such as plant and leaf segmentation, leaf counting, classification, and others. The PRL dataset includes several different image sets multiple contexts: images of individual plants, and trays containing multiple plants. For the experiments described here, we focus on the images of individual plants, which are subdivided into three datasets - two datasets of Arabidopsis thaliana (A1, A2) and one dataset of Nicotiana tabacum (tobacco) (A3). We perform the leaf counting task on all three datasets, as well as the mutant classification and and age regression tasks on the A2 A. thaliana dataset for which ground truth is available. All tasks use only the RGB images from each of the three datasets. The sizes of each of the three datasets are 120, 165, and 62 examples respectively. Examples from the A2 dataset show a wide variation in image resolution, the size of the visual field, as well as morphological differences such as leaf shape and size (Figure 3.1). The number of leaves varies between five and twenty leaves per plant for the A. thaliana examples, and between two and thirteen for the Tobacco examples. To determine the ground truth for leaf counts, the authors of the dataset extrapolated the count from human expert provided leaf segmentations for each image. Further description of the dataset and the methodology used in its construction is provided in the publication [81].

The phenotyping tasks evaluated in the present study represent challenging traits to measure from images. Leaf count is an important phenotype because of its correlation with such features as yield, drought tolerance,
and flowering time [81]. This makes leaf count not only distinct from shoot area or biomass, but a useful phenotype in its own right. Mutant classification is related to identifying morphological differences between plant varieties. While an experiment may not explicitly want to classify mutants (as these would already be known), classifying images of plants based on morphological differences is important because the same morphological changes to a plant that are observed (induced) in a mutant may be relevant phenotypes for natural plants, i.e. the morphological changes present in certain mutants may be caused by other pathways such as from pests or disease; therefore mutant classification can be a demonstration of more challenging disease classification that have morphological features rather than color, etc. Age regression, measured in hours after germination, relates to plant maturity, which is an important phenotype in plant breeding. While an experiment may not directly need to estimate age, since it is known a priori, estimating the maturity of different varieties is important. For example, which variety matures earlier or more rapidly at certain growth phases.

In order to demonstrate that the proposed method is robust to changes in scene lighting, an additional experiment was performed on the A2 leaf counting dataset. In this robustness experiment, the brightness and contrast of images in the test were randomly adjusted. Since the model is also trained with brightness and contrast adjustments as a form of augmentation (detailed below), different parameters for this adjustment were used to bring the distortions out of the range seen by the model during training. During the training, brightness was modified with a maximum delta of 63 and contrast was modified with a lower range of 0.2 and an upper range of 1.8. For testing, the delta for brightness was set to 75, and the lower and upper parameters for contrast were set to 0.5 and 2.1 respectively.
3.2.2 Approach

Convolutional neural networks (CNNs) were constructed and trained from scratch to perform each of the three benchmark tasks. The structure of the network varied slightly between tasks, as the model was tailored to the problem and the data. This tailoring of the architecture is not necessary; however, we perform the modifications here in order to obtain higher performance results and demonstrate the capabilities of the method.

For the A2 leaf counting dataset, a convolutional neural network was constructed with two 5×5 convolutional layers, three 3×3 convolutional layers, and an output layer. Each convolutional layer was followed by a max pooling layer with a 3×3 spatial size and a stride of 2 pixels. The Xavier (Glorot) initialization scheme [38] was used in each case, with tanh used as the activation function. Images were resized to 128×128 and cropped to 96×96 randomly during training, and to centre during testing. For all experiments, the only pre-processing applied to the images was per-image standardization, which subtracts the mean from the image matrix and divides by the standard deviation. The A1 dataset includes only one accession of Arabidopsis (Col-0), which tends to have smaller and more tightly packed leaves. Therefore, we increased the input size to 256×256 pixels and added an additional 3×3 convolutional and pooling layer to the network. We reduced the automatic cropping from 25% to 10% to avoid losing leaves near the edges of the image, as images in this dataset seem to be more tightly cropped. We also added a fully connected layer with 1024 units. For the A3 dataset, we used the same modifications as for the A1 dataset, with the exception of the fully connected layer. For the mutant classification task, the network used a feature extractor comprised of four 5×5 convolutional layers, each followed by a pooling layer as before. The output was fed into a classifier with two fully connected layers, each having 4096 units and each followed by a Dropout layer (p = 0.5). We used a 128×128 input size, and the ReLU activation function in all layers. The age regression network was comprised of two 3×3 convolutional layers, each followed by a max pooling layer, and a single fully connected layer with 2048 units. We retained the 128×128 input size and the ReLU activation function for this task.

Performing deep learning with small datasets can be particularly challenging, as small training sets can be easy for a deep network to memorize, resulting in problematic overfitting. This results in low training error, but high testing error. This discrepancy is termed the generalization error. One way to protect against overfitting when performing learning with images is to perform dataset augmentation. By applying distortions to images in the training set with some probability, the size of the training set is artificially but effectively increased. In all experiments, brightness, contrast, cropping, and flipping distortions were applied randomly to augment the training set.

For testing, a random 80-20 train-test split was used in all experiments. It is considered good practice to implement “early stopping” during training, by withholding a portion of the dataset (called the validation set) to test on and stopping training once the network attains a certain level of performance on these samples. This helps to prevent overfitting, where performance on the test set may subsequently drop as training continues past this point. Since the 80-20 split used by previous published results does not include any validation set
that could be used to implement early stopping, we stop training after the training loss appears to plateau. The gradient-adaptive Adam algorithm was used for optimization in all experiments [55].

3.2.3 Results

The mean absolute difference results for the three different leaf counting datasets are provided in Table 3.1. We do not include results for training accuracy, because a deep convolutional network with sufficient capacity is able to fit the training data with an arbitrary degree of accuracy. We also do not report the (non-absolute) count difference (CountDiff), which does not directly measure performance since over-prediction and under-prediction are able to negate each other. We compare against two results from the literature using similar data, with the caveat that these results are from the CVPPP Leaf Counting Competition\footnote{https://www.plant-phenotyping.org/datasets-home} and as such may have a different number of training images, as well as different training and testing images. We provide the comparison for context, if not for direct comparison.

The mutant classifier model proved effective in distinguishing between five different mutants of Arabidopsis, with a measured 96.88% mean test accuracy. This is an encouraging result, and it sets the baseline performance for this task as the first published result.

For the age regression task, our model achieves a mean absolute difference of 20.8 hours with a standard deviation of 14.4 hours. The ground truth labels for this task range between 392 hours and 620 hours. Like for the mutant classification task, this result is the first published result for this task.

<table>
<thead>
<tr>
<th></th>
<th>[37]</th>
<th>[93]</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1.27 (1.15)</td>
<td>2.2 (1.3)</td>
<td>0.41 (0.44)</td>
</tr>
<tr>
<td>A2</td>
<td>2.44 (2.28)</td>
<td>1.2 (1.3)</td>
<td>0.61 (0.47)</td>
</tr>
<tr>
<td>A3</td>
<td>1.36 (1.37)</td>
<td>2.8 (2.5)</td>
<td>0.61 (0.54)</td>
</tr>
</tbody>
</table>

Table 3.1: Mean (std) absolute difference for the three leaf counting benchmarks.

The results for the experiment investigating robustness to variance in scene were a mean absolute difference of 0.64, with a standard deviation of 0.51. These results are comparable with the unmodified test set (Table 3.1) which suggests that the network is indeed robust to changes in lighting conditions.

3.2.4 Discussion

For the leaf counting task, the proposed method shows significantly better performance on each of the three benchmark datasets in terms of the absolute difference in count compared to previous methods. In addition, both the mean and standard deviation are more consistent between tasks using the proposed method. Both results from the literature show significantly degraded performance on a selection the three benchmark tasks — multi-accession Arabidopsis for [37], and both Col-0 Arabidopsis and Tobacco for [93]. In contrast, the
proposed CNN method shows that it is capable of learning representations of the training data which are effective for each of the three datasets.

Tests with artificially modulated image brightness and contrast demonstrate that the CNN method can be made robust to changes in scene lighting conditions through data augmentation during training. Therefore, the proposed technique has better potential than classical image analysis methods for translation to field phenotyping where scene conditions are more variable, e.g. for measuring emergence counts from aerial images of rosettes in field plots. It also means that the method can be used in indoor environments such as greenhouses, where the lighting modality cannot be controlled.

It is common to visualize the filters of the first convolutional layer, since these filters often contain some interpretable structure as they correspond to operations over the input image. Later convolutional layers
are more difficult to interpret, as they correspond to abstract output from the previous convolutional layer. Since the leaf counter network uses $5 \times 5 \times 3$ filter weights, not much interesting structure appears in the filter weights of the first convolutional layer during training. However, by increasing the filter size to $11 \times 11 \times 3$, some interesting structure appears in these filters (Figure 3.4). The trained filters result in mostly green and violet pixels. Violet pixels respond to high values in the red channel and low values in the green channel; therefore, it is likely that the presence of leaves is being suppressed in these regions of the receptive field.

It is noteworthy that both previous leaf counting algorithms to which we compare our method (Table 3.1) require pre-segmented plant images, and presumably the performance of their algorithm is dependent on the accuracy of this segmentation. In contrast, the CNN technique requires no such pre-processing and only the raw RGB images are required as input. The authors of the dataset describe the inputs to the age estimation task to be the RGB images as well as the labels for mutant type; however, we use only the RGB images and rely on the network to learn representations which are robust to differences in growth rate between mutants. Experiments using the mutant labels actually performed worse, as it allowed the network to use the label to fit the training data more aggressively and this was detrimental to generalization performance.

The advantage of supervised learning methods over hand-engineered image analysis techniques in tasks such as leaf counting is their capacity for complex representation learning. For example, a hand-engineered image processing pipeline must be designed to accommodate leaves of different shapes and sizes (for plants of different ages and genotypes), leaves with different length petioles, as well as partially overlapping leaves. A supervised representation learning algorithm such as a CNN is capable of automatically learning a representation of the data which takes into account all of these factors, and any others which are present in the training data.

Although designing CNN architectures requires less hand-engineering than image processing pipelines, the process is not completely automated. Building a network architecture to perform any computer vision task involves some iterative optimization in two areas: the number and size of network layers, and the values of hyperparameters such as learning rate and regularization strength. For hyperparameter tuning, some automated methods are available such as simple grid search and Bayesian optimization [124]. Although training a CNN requires such considerations, it is less cumbersome than tuning the alternative image processing pipeline. For example, the leaf counting pipeline described in [93] contains 14 discrete image processing steps, the majority of them having tuneable parameters such as noise area limit and gap fill size limits.

There are several promising directions for future research for which development of the DPP platform is ongoing. Support for additional types of network architectures, may offer more utility for future applications. Implementing recurrent models such as Long Short Term Memory (LSTM) networks [47] would allow for the prediction of temporal features, which may be a relevant class of features.

Implementing transfer learning in the platform has the potential to provide higher accuracy and lower training times. Transfer learning involves starting with a network pre-trained on large datasets, such as the ImageNet database [21], and then fine-tuning the network with a smaller set of images tailored to the task.
of interest, e.g. rosette images. This technique is widely accepted in the literature for bootstrapping the learning process, and has proven successful in plant disease diagnosis [83]. Although the DPP platform has only been tested with data collected in a controlled environment, further testing can be done to explore applications with outdoor, field-level applications. There are also opportunities to test the performance of the system on larger datasets, such as those collected from automated greenhouses.

Since the initial introduction of the DPP platform, it has been extended by various contributors. The package is now also capable of performing counting via density estimation [67] or fully-redundant counting [95], binary and multi-class semantic segmentation, and object detection using a single-shot method [110]. It incorporates many additional network types as pre-defined architectures, including residual networks [44], fully convolutional networks [143], and others. The tool is used regularly by the public and has also resulted in an ongoing research collaboration [78]. Although the development of the tool has been performed within the Plant Phenotyping and Imaging Research Center at the University of Saskatchewan, the global plant phenotyping and computer vision community is well positioned to continue the maintenance and development of the package in the future.
Figure 3.3: Example training (red) and testing (blue) curves for several benchmark tasks.
Figure 3.4: Visualization of filter weights in the first convolutional layer of a leaf counting network.
4 Synthetic Data for Deep Learning Using Plant Modeling

4.1 Introduction

Non-destructive, image-based plant phenotyping has emerged as an active area of research in recent years. This is due in part to a gap in capability between genomics and phenomics, as well as the complexity of genotype-to-phenotype mapping [33]. The ability to correlate heritable traits with genetic markers relies on the accurate measurement of phenotypes. In order to achieve statistical power, this measurement typically needs to be done at a large scale which makes measurement by hand intractable. Image-based phenotyping is an important tool for genotype-phenotype association as it allows for the required automation. High-throughput imaging is aided by imaging technologies available in some automated greenhouses, as well as low-cost imaging tools which can be made with off-the-shelf parts [82]. An appropriate software environment is also required for the automatic extraction of phenotypic features from the image data. Ideally, such software should be highly automated, scalable, and reliable. Although high-throughput phenotyping is typically conducted in circumstances where the scene can be controlled, for instance on rotating stages in imaging booths, computer vision algorithms should be invariant to changes in the scene if they are to be used in greenhouse or field environments. These algorithms should also take into account other factors, such as the structural variation between different species or accessions, the shape and color of leaves, and the density and geometric eccentricity of the shoots. Therefore, any algorithm that contains parameters which are hand-tuned to a specific collection of plants is at risk of being overly specified.

Unlike engineered computer vision pipelines, deep neural networks learn a representation of the data without image parameters specified by hand. This makes them potentially more robust to different types of variations in the image data, as the network can adapt to be invariant to such differences. However, the transition from hand-engineered computer vision pipelines to deep learning is not without limitations. While so-called “deep” networks have the representational capacity to learn complex models of plant phenotypes, the robustness of these representations relies on the quality and quantity of the training data. In most vision-based tasks where deep learning shows a significant advantage over engineered methods, such as image segmentation, classification, and detection and localization of specific objects in a scene, the size of the dataset is typically on the order of tens of thousands to tens of millions of images [21]. This allows for much variety in the training data, and very robust learned representations as a consequence.
Datasets of plant images, labeled with corresponding phenotypic data, are not yet available on a large scale due to the considerable expense involved in collecting and annotating this type of data. In addition, any supervised machine learning method, including deep learning, requires that the data used to train the model is representative of the data used at test time. Plant phenotyping tasks are vulnerable to such problems with incomplete training data due to the difficulty of generating a dataset in which a comprehensively wide range of phenotypes are represented. The small size of existing plant phenotyping datasets, the expense of generating new data, and the limitations of naturally-generated datasets motivate the use of an alternative source of data to train deep networks for plant phenotyping tasks. For this purpose we propose the use of synthetic plants – images of computer-generated plant models – to augment datasets of plant images or to be used alone as a large and rich source of training data. Compared to generating new data using real plants, once a model is developed, the generation of new data is essentially without cost. Moreover, models can be parameterized to generate an arbitrary distribution of phenotypes, and ground-truth phenotype labels can be automatically generated without any measurement errors and without any human effort or intervention.

4.1.1 Deep Learning and Dataset Shift

For some applications, the construction of large data sets of labeled images can be facilitated by crowd-sourcing images freely available on the Internet [21]. Unfortunately, this approach is not possible for plant phenotyping datasets, due to their specificity. The creation of these datasets requires sampling a wide range of accessions, and many individual plants need to be cultivated from germination to maturity. Along with the agricultural work involved, each plant must be imaged individually (or segmented from a tray image containing multiple plants), and each image needs to be annotated with ground truth data, measured manually and/or specified by an expert. Although high-throughput imaging systems do exist to expedite the process of collecting large sets of plant images, the end-to-end phenotyping process remains prohibitively time consuming and expensive, limiting the size of the available datasets. Existing plant image datasets are available for a wide range of applications, including both roots and shoots [73]. These public collections are a valuable source of data for many applications, and often do include annotations for ground truth. However, we find it compelling to offer a source of new, additional data alongside these public collections which is free of the aforementioned limitations.

Even for large training datasets, the network can still fail to properly recognize phenotypes if the distribution of testing data differs significantly from that of the training data. In the case of leaf counting, the distribution of leaf numbers in the training data must be similar to that of the testing data: if the rosettes used for training have significantly fewer leaves than the rosettes used for testing, the learned model will likely be *misspecified* and mis-predict the number of leaves. In technical terms, the learning process infers a conditional model \( p(y|x) \): the conditional distribution of the outputs given the inputs. Differences between training and testing data can result in two related problems known as *covariate shift*, where \( p(x) \) changes between training and testing, and *dataset shift*, a different conditional distribution \( p(y|x) \) of the outputs and
inputs in the test data, compared to that in the training data. This problem is common in machine learning and can be difficult to mitigate [84]. Available techniques often focus on statistically modeling the difference between the training and testing distributions. However, finding such a mapping is not only practically infeasible for complex vision-based tasks, but also assumes the availability of samples drawn from the test distribution. These issues are unique to supervised learning, as hand-engineered pipelines containing a priori information typically do not have to model the conditional distribution explicitly. The problem of dataset shift is almost inevitable when using supervised learning for plant phenotyping tasks, due to the limitations of generating new plant phenotyping datasets. It is not possible to specify the domain of phenotypes to be represented in the data, and so this limitation will tend to expose problems of dataset shift when using models of phenotypes learned from this data. We investigate the use of computational plant models to mitigate this problem.

4.1.2 Computational Plant Models

Computational modeling has become an inherent part of studies of plant physiology, development, architecture, and interactions with the environment. Diverse concepts and techniques exist, applicable to constructing models at spatio-temporal scales ranging from individual cells to tissues, plant organs, whole plants, and ecosystems [102, 106, 120]. The formalism of L-systems [70], augmented with a geometric interpretation [99, 104] provides the basis for a class of specialized programming languages [104, 53, 11] and software (e.g. [100, 45, 98]) widely used to model plants at different levels of abstraction and for a variety of purposes. In the domain of phenotyping, Benoit et al. [8] employed an L-system-based root model [65] to generate testing data for validating image-based root system descriptions. To create or augment training data sets for image-based leaf counting tasks considered in this paper, we constructed a descriptive model that reproduces early developmental stages of the plant shoot on the basis of direct observations and measurements (without accounting for the underlying physiological processes). Applications of L-systems to construct such models are presented, for example, in [104]; the subsequent enhancements include gradual modifications of the organ shapes as a function of their age [103, 101] and position in the plant [105], as well as the use of detailed measurements of shape [85]. The model of rosettes used in this paper is the first application of L-systems to model plant shoots for phenotyping purposes.

4.1.3 Related Work

The use of synthetic or simulation data has been explored in several visual learning contexts, including pose estimation [89, 17] as well as viewpoint estimation [127]. In the plant phenotyping literature, models have been used as testing data to validate image-based root system descriptions [8], as well as to train machine learning models for root description tasks [74]. However, when using synthetic images, the model was both trained and tested on synthetic data, leaving it unclear whether the use of synthetic roots could offer advantages to the analysis of real root systems, or how a similar technique would perform on shoots.
The specialized root system models used by Benoit et al. [8] and Lobet et al. [74] are not applicable to tasks involving the aerial parts of a plant — the models have not been generalized to produce structures other than roots. Nonetheless, for image-based tasks Benoit et al. [8] were the first to employ a model [65] based on the L-system formalism. Because of its effectiveness in modelling the structure and development of plants, we chose the same formalism for creating our Arabidopsis rosette model.

4.2 Methods

In the present work, we seek to demonstrate that realistic models of synthetic plants are a sufficient replacement for real data for image-based plant phenotyping tasks. We show that a model of the Arabidopsis thaliana rosette can be used either in conjunction with real data, or alone as a replacement for a real dataset, to train a deep convolutional neural network to accurately count the number of leaves in a rosette image. We also discuss how the concept of model-based data augmentation may extend to other plants and phenotyping tasks.

Image Sources and Processing

For the images of real plants used in the leaf counting task, we use a publicly available plant phenotyping dataset from the International Plant Phenotyping Network (IPPN)
\(^1\), referred to by its authors as the PRL dataset [81]. The PRL dataset is a multi-purpose phenotyping dataset that includes ground truth labels for several different phenotyping tasks, including leaf counting and segmentation, age estimation (hours after germination), and mutant classification. Two annotated image subsets are available within PRL for the leaf counting task using Arabidopsis rosettes considered in this paper. These subsets, referred to as Ara2012 and Ara2013-Canon, vary in the several ways, including the accessions of the subjects, lighting, level of zoom, image sizes, leaf size and shape, and the distributions of the number of leaves (Table 4.1). The full datasets, as well as several alternative versions, are downloadable at https://figshare.com/articles/SATLC-28-09-17_zip/5450080.

When training on synthetic images and testing on real images (as in Table 4.3 rows 3, 4, and Table 4.4 rows 1, 3), we set the background pixels to black using the segmentation masks provided with the PRL dataset. This was done to prevent the network from reacting to objects in the background of the image, which were not accounted for in the plant model. Although training on images of real plants with a variety of non-uniform backgrounds results in a model which is conditioned to be invariant to such backgrounds, these backgrounds are more difficult to control for when using synthetic plants as the training data. Although we use the foreground-background segmentations provided by the authors of the dataset, automatic segmentation methods targeting plants [19, 80, 42] or general-purpose [148] could be used on these data as the rosettes appear on a uniform soil background, and indeed have been demonstrated in previous studies [113, 94].

\(^1\)https://www.plant-phenotyping.org/datasets-home
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Number of images</th>
<th>Range of leaf counts</th>
<th>Accessions</th>
<th>Image size</th>
<th>Scale&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Background</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ara2012</td>
<td>120</td>
<td>12-20</td>
<td>Col-0</td>
<td>varied</td>
<td>1:1</td>
<td>soil/tray</td>
</tr>
<tr>
<td>Ara2013-Canon</td>
<td>165</td>
<td>5-13</td>
<td>Col-0/mutants</td>
<td>varied</td>
<td>1:1</td>
<td>soil</td>
</tr>
<tr>
<td>S1</td>
<td>1000</td>
<td>12-20</td>
<td>N/A</td>
<td>256 × 256</td>
<td>1:1 – 1:2</td>
<td>soil</td>
</tr>
<tr>
<td>S2</td>
<td>1000</td>
<td>5-13</td>
<td>N/A</td>
<td>256 × 256</td>
<td>1:1 – 1:2</td>
<td>soil</td>
</tr>
<tr>
<td>S12</td>
<td>1000</td>
<td>5-20</td>
<td>N/A</td>
<td>256 × 256</td>
<td>1:1 – 1:2</td>
<td>varied</td>
</tr>
</tbody>
</table>

<sup>1</sup> Scale denotes the ratio of the plant diameter to the image size.

Table 4.1: Real and synthetic training datasets.

### 4.2.1 CNN Architectures

In the augmentation experiment, we replicated the architecture used in conjunction with the Ara2013-Canon dataset in the reference experiment [136], in order to compare our results with those published previously. This architecture uses three convolutional layers, each with a 5 × 5 spatial resolution and a stride size of one pixel, and each followed by a 3 × 3 pooling layer with a stride size of two pixels. In the remaining experiments (generalization and interoperability), we employed a larger CNN architecture, used in conjunction with the Ara2012 dataset in [136]. This architecture uses four convolutional layers, each followed by a pooling layer, and a single fully connected layer with 1024 units, followed by the output layer. The tanh activation function was used in all cases, and λ = 10<sup>-4</sup> was used for the L<sub>2</sub> weight decay when training on synthetic data to limit overfitting. In all experiments, the static learning rate was 10<sup>-3</sup>. The training dataset was augmented with standard image-based techniques. Image variation was increased using vertical and/or horizontal flips, and cropping by 10% to a window randomly positioned within the input image. The brightness and contrast were also randomly modified. As in previous work, we split the data randomly into training (80%) and testing (20%) for each experiment.

### 4.2.2 An L-System Model of the Arabidopsis thaliana Rosette

To augment the PRL dataset of Arabidopsis rosette images, we developed a model of Arabidopsis in the vegetative stage based on an existing model [85]. The model was implemented using the L-system-based plant simulator 1pfg included in the Virtual Laboratory plant modeling environment [100, 1]. The full model code is available in the dataset file which has been provided for download. The rosette was constructed as a monopodial structure with leaves arranged on a short stem in a stochastically determined phyllotactic pattern. The length of a leaf, l<sub>n</sub>(t), at node number n and age t was computed as l<sub>n</sub>(t) = f<sub>lmax</sub>(n) · f<sub>t</sub>(t), where f<sub>lmax</sub>(n) is the final length given the node number, and f<sub>t</sub>(t) controls the leaf length over time. Leaf blades were modeled as flat surfaces, fitted to an arbitrarily chosen image of an Arabidopsis leaf from the Ara2012 dataset. The width of the leaf blade was scaled proportionally to its length, w<sub>n</sub>(t, x) = l<sub>n</sub>(t) · f<sub>tw</sub>(x),

40
where \( f_{lw}(x) \) is the leaf contour function and \( x \) is the distance from the leaf base along the midrib. Petiole length was set to be proportional to leaf length, and petiole width was assumed to be constant. The leaf inclination angle was specified as a function of node number \( f_{ang}(n) \).

All functions were defined using the Virtual Laboratory graphical function editor funcedit (Figure 4.1). The shapes of the functions were drawn (by manual placement of control points) such that the final leaf length, leaf length over time, inclination angle, and leaf shape agreed with the published measurements [85].

We modeled the diversity of Arabidopsis rosettes by modifying the final leaf length (and, proportionally, the leaf width) using normally distributed random variables. Specifically, for each leaf along the stem, we multiplied \( f_{max}(n) \) by a variable \( X_n \) taken from normal distribution with mean \( \mu = 1 \) and standard deviation \( \sigma = 10^{-2} \). Likewise, the divergence (phyllotactic) angle between consecutive leaves \( n \) and \( n+1 \) was calculated as a normally distributed random variable \( \theta_n \) with mean \( \mu = 137.5 \) and standard deviation \( \sigma = 2.5 \). Finally, the time of development of the rosette was varied using a uniform random variable for each simulation run, such that the final number of leaves was in the range from 5 to 20.

Our model was implemented using parametric L-systems, in which each component of a plant (apex, leaf, and internode) has a corresponding module with associated parameters [104]. For example, in the module \( A(n) \) representing the apex, the parameter \( n \) is the node number. We simulated the development of the plant by a set of rewriting rules, which specify the fate of each module (component) over an increment of time. An apex, for instance, produces a new internode and new leaf at regular time intervals. To account for diversity of rosettes, we generated 1000 images with a random variation. Figure 4.2 shows three example renderings alongside three real images for visual comparison. Figure 4.3 shows the model at maximum maturity.
Figure 4.2: Synthetic rosettes (left) generated by the L-system and real rosettes (right) from the public dataset [81]

4.3 Results

To validate the use of models with deep learning, we conducted three leaf counting experiments using images of both real and synthetic Arabidopsis rosettes. The mean absolute count difference, and the standard deviation of absolute count difference, were measured in each experiment. The experiments were conducted as follows:

4.3.1 Augmentation

This experiment tested the usefulness of synthetic plants in augmenting the Ara2013-Canon dataset of real plants for the leaf counting task. For this purpose, we generated a set of one thousand synthetic rosettes (S2) and added them to the training set. The model’s background was set to a brown color approximating the soil in the real dataset. Using synthetic rosettes to augment the training set, we observed a reduction of approximately 27% in the mean absolute count error (Table 4.2).

<table>
<thead>
<tr>
<th></th>
<th>AbsCountDiff</th>
<th>CountDiff</th>
<th>MSE</th>
<th>$R^2$</th>
<th>agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubbens and Stavness, 2017 [136]</td>
<td>0.61 (0.52)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Synthetically augmented (S2)</td>
<td><strong>0.48 (0.58)</strong></td>
<td>0.15 (0.82)</td>
<td>0.73</td>
<td>0.92</td>
<td>80%</td>
</tr>
</tbody>
</table>

Table 4.2: Augmentation results, Ara2013-Canon dataset.
4.3.2 Generalization

In this experiment we investigated whether the ability of the model to generate an arbitrary range of phenotypes may be used to mitigate the problem of dataset shift. To this end, we trained a leaf counting network on purely synthetic data and tested it on two real datasets, each with a different distribution of leaf numbers. These datasets exhibit both covariate shift in the different distributions of leaf counts, as well as dataset shift in the intersection between the two as described in the background on deep learning. For brevity, we will address both problems as dataset shift in our discussion. The synthetic training data consisted of one thousand synthetic rosettes with a uniform distribution of leaf numbers between five and twenty (S12). The model was then tested on the Ara2012 dataset (with a range of between 12 and 20 leaves) and the Ara2013-Canon dataset (between 5 and 13 leaves). A synthetic training set which is easy for the network to fit will result in poor generalization due to overfitting; in order to introduce more variance to the synthetic data with the goal of reducing overfitting, the model’s background was set to either a soil color or a random color in RGB space ($p = 0.5$). Although the images the network was tested on were segmented onto a black background, the addition of different background colors in the model varied the contrast between the leaves and background in the individual color channels, which showed to be beneficial for generalization when using synthetic images, in terms of the test error.

When training on dataset Ara2012 and testing on Ara2013-Canon, or vice versa, we observed significantly degraded performance due to dataset shift. However, when training on a purely synthetic rosettes, dataset

Figure 4.3: Top view of the synthetic *A. thaliana* model at maximum maturity.
shift is mitigated with mean count error more closely centered around zero (Table 4.3). The distributions of relative count errors for both real datasets when trained on real and synthetic data are shown in Figure 4.4. Although the mean absolute count errors are similar in each case, the coefficient of determination shows that the predictions made on Ara2012 are much more strongly correlated with the ground truth measurements ($R^2=0.42$) than those on Ara2013-Canon ($R^2=-0.33$).

<table>
<thead>
<tr>
<th>Training Data</th>
<th>Testing Data</th>
<th>AbsCountDiff</th>
<th>CountDiff</th>
<th>MSE</th>
<th>$R^2$</th>
<th>agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ara2013-Canon</td>
<td>Ara2012</td>
<td>5.45 (2.04)</td>
<td>-5.45 (2.04)</td>
<td>33.9</td>
<td>-4.79</td>
<td>0%</td>
</tr>
<tr>
<td>Ara2012</td>
<td>Ara2013-Canon</td>
<td>5.39 (1.99)</td>
<td>5.39 (1.99)</td>
<td>33.13</td>
<td>-6.15</td>
<td>0%</td>
</tr>
<tr>
<td>S12</td>
<td>Ara2012</td>
<td>1.38 (1.03)</td>
<td>-0.25 (1.7)</td>
<td>2.97</td>
<td>0.42</td>
<td>22%</td>
</tr>
<tr>
<td>S12</td>
<td>Ara2013-Canon</td>
<td>1.82 (1.38)</td>
<td>0.46 (2.24)</td>
<td>5.25</td>
<td>-0.33</td>
<td>20%</td>
</tr>
</tbody>
</table>

Table 4.3: Performance when training and testing on different datasets. Training on a single dataset of synthetic rosettes performs significantly better than training on a dataset of real rosettes with a different distribution of phenotypes.

Figure 4.4: Distributions of relative count difference in the generalization experiment. Training on one dataset and testing on another exhibits severe dataset shift (top), while training on synthetic data significantly reduces this error by encompassing a comprehensive range of leaf counts (bottom).

4.3.3 Interoperability

This experiment tested the interoperability between real and synthetic plants by training a network on real plants (Ara2013-Canon) and testing it on synthetic plants (S2) containing the same range of leaf numbers,
or vice versa: training on the set S2 and testing on Ara2013-Canon. A small error value in this experiment signifies that the model is a suitable stand-in for real plants for the leaf counting task. Statistics are provided for both cases (Table 4.4), as well as scatter plots illustrating the correlation between ground truth and predicted value (Figure 4.5). Although the $R^2$ statistics are substantially lower when using synthetic data, this is partially due to a small number of outliers which are highly penalized due to the squared error term in the $R^2$ calculation. The scatter plots (Figure 4.5) show these outliers as well as a line of best fit, which shows better correlation with ground truth than the $R^2$ statistics would suggest.

<table>
<thead>
<tr>
<th>Training Data</th>
<th>Testing Data</th>
<th>AbsCountDiff</th>
<th>CountDiff</th>
<th>MSE</th>
<th>$R^2$</th>
<th>agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>S2</td>
<td>Ara2013-Canon</td>
<td>1.29 (1.01)</td>
<td>-0.02 (1.64)</td>
<td>2.7</td>
<td>0.26</td>
<td>24%</td>
</tr>
<tr>
<td>Ara2013-Canon</td>
<td>S2</td>
<td>0.81 (0.54)</td>
<td>0.28 (0.93)</td>
<td>0.95</td>
<td>0.82</td>
<td>34%</td>
</tr>
<tr>
<td>S1</td>
<td>Ara2012</td>
<td>1.70 (1.21)</td>
<td>0.67 (1.98)</td>
<td>4.39</td>
<td>0.27</td>
<td>25%</td>
</tr>
</tbody>
</table>

Table 4.4: Interoperability between real and synthetic rosettes.

![Figure 4.5: Scatter plots of actual and predicted leaf counts in the interoperability experiments. Training on synthetic and testing on real (left), and training on real and testing on synthetic (right).](image)

4.4 Discussion

Deep learning models, including the deep CNNs used in the experiments presented here, have a large capacity for fitting the training data. This is essential to their learning ability, but also makes them susceptible to overfitting in the case of small datasets, or large datasets with an insufficient level of variation. Therefore, it is important to consider how to introduce as much variation as possible into the model and the scene. For example, we found that generalization improved when plants were randomly scaled, with the ratio of the plant diameter to the size of the entire image varying between 1:1 and 1:2. This helped prevent the network from using the number of green pixels as a proxy for the number of leaves, which could be a viable strategy if the model lacked enough variance in leaf size. Other considerations include varying the contrast between background and foreground pixels. Such variations in the model, the scene, as well as secondary image-based augmentations such as modifications of the brightness and contrast all contribute to preventing overfitting.
**Figure 4.6:** Comparison of training and testing loss on real (red) and synthetic (blue) rosettes. Real plants show significantly higher generalization error, while the synthetic dataset is relatively easy to fit.

Comparing the counting errors during training and testing, we observed that their difference (the generalization error) is larger for real data than for synthetic data (Figure 4.6). This means that, despite attempts to capture specimen-to-specimen variation using a stochastic model, our synthetic plants are significantly easier to fit and therefore do not fully capture the diversity of real rosettes. The network’s performance in the task of counting real leaves could thus be improved by adding more variation to the set of synthetic plants used for training. However, even with the limited variation, networks trained on the synthetic rosettes do seem to benefit from larger training sets (Figure 4.7), which is a characteristic typically seen in natural datasets as well.

**Figure 4.7:** Test performance on purely synthetic data when using increasing sizes for the training set. Like with datasets of natural images, we see that generalization performance improves with larger training sets.

Another consequence of overfitting is the network’s tendency to discriminate between different types of data. In tests with both real and synthetic data, if these datasets had different leaf distributions, the network would learn to map each type of data to an individual output distribution, with a detrimental effect on generalization performance. This means that the use of synthetic data in conjunction with real data is only
advisable if the distributions of phenotypes of the real and synthetic data overlap. Although this could be seen as a disadvantage, we have also shown that the use of synthetic data alone is sufficient and avoids this effect.

We observed that models which are not sufficiently realistic resulted in degraded performance compared to more accurate models. For example, an initial rosette model in which all leaves were assumed to be of the same size showed significantly lower interoperability with the images of real rosettes. Taking into account not only the differences in leaf size, but also in shape as a function of their position [85], as well as capturing differences in leaf colour and texture, may further contribute to the realism and diversity of synthetic images used for training purposes. Future work includes the inclusion of a more detailed model of leaf shape which includes serrations and sinuses. These considerations were not included in the present model due to limited variance in leaf shape in the available images of real rosettes. Ultimately, the most accurate images of plants under different conditions may be provided by mechanistic models relating plant appearance to the underlying physiological processes.

Future directions for research could further explore the relationship between models trained on real data and those trained on synthetic data, including techniques such as transfer learning. Using a feature extractor learned on synthetic data and re-training a regressor with these features may shed light on differences in learned representations between the two types of data.

In summary, the results presented in this paper show promise for the use of models in image-based plant phenotyping tasks. The existing body of work on L-system modeling of plants is extensive, with models available for many different species. These existing models are well positioned to take the results demonstrated here on Arabidopsis forward towards other applications. One potentially important application area is the modeling of entire plots of crops. A simulated plot of plants could potentially make it possible to train algorithms for detecting biologically meaningful traits such as flowering time or response to stress with a reduced number of real (annotated) crop images. Other directions for future work could include augmentation using synthetic data for other supervised learning problems, such as leaf segmentation. Other applications, such as disease detection, would be possible if future plant models were able to model such phenomena.

4.5 Conclusion

We applied a computer-generated model of the Arabidopsis rosette to improving leaf counting performance with convolutional neural networks. Using synthetic rosettes alongside real training data, we reduced mean absolute count error with respect to results obtained previously using only images of real plants [136]. We also demonstrated that — due to the model’s ability to generate an arbitrary distribution of phenotypes — a network trained on synthetic rosettes can generalize to two separate datasets of real rosette images, each with a different distribution of leaf counts. Finally, the interoperability experiments have shown, in particular,
that a CNN trained only on synthetic rosettes can be successfully applied to count leaves in real rosettes. 3D plant models are thus useful in training neural networks for image-based plant phenotyping purposes.
5 Latent Space Phenotyping

5.1 Introduction

Developing crop varieties that maintain a consistent yield across different environmental conditions is an important target for plant breeding as weather patterns become more variable due to global changes in climate. Breeding for yield stability requires characterization of an individual plant’s response to biotic and abiotic stress [123] relative to a breeding population. Treatment studies, where individuals in a treated population are subjected to different growing conditions than a control population, play an important role in uncovering the genetic potential for resistance to stress. Such experiments include genotype-by-environment (G×E) studies, where the treatment is often abiotic stress, such as water limited growing conditions, or genotype-by-management (G×M) studies, where the treatment is different application of inputs, such as herbicide application to assess herbicide tolerance or nitrogen application to assess nitrogen use efficiency. A core challenge for this broad class of experiments, is the ability to quantify and characterize the physical changes observed in the treated plant population relative to the control population, i.e. to phenotype a plant’s response-to-treatment.

A number of factors make response-to-treatment a difficult phenotype to quantify. In general, stress affects multiple plant traits simultaneously. Stressors can also have a substantially different type and magnitude of effect on different plant species and different cultivars within the same species. Finally, quantifying response and recovery to stress is sensitive to the timing of observations, and often requires repeated observations over a plant’s life cycle in order to capture important phenological features. An accurate and quantitative assessment of response-to-treatment is particularly important for genomic association studies.

The use of association mapping techniques, such as genome-wide association studies (GWAS) have yielded many candidate loci for agronomically important quantitative traits in plants [13]. For food crops, genome-wide analysis of susceptibility or resistance to abiotic stress factors such as drought [29], nitrogen deficiency [88], salinity [15], or other factors leads to the discovery of genetic differences underlying these agronomically important characteristics. These treatment-based GWAS studies are capable of identifying resistance alleles which could result in a tolerance to a wider variety of environmental conditions if, for example, introgressed into commercial cultivars. GWAS studies, however, require large datasets of phenotypic data in order to map associations with genomic data [33].

High-throughput phenotyping (HTP) technologies have advanced rapidly in the past five years to meet the demand for large phenotypic datasets. Recently, image-based HTP has gained popularity, because pho-
tographing plants in greenhouses or fields with robots and drones has allowed data collection at yet larger scales. The phenotyping bottleneck has shifted from collecting images, which can now be done routinely, to making sense of those images in order to extract phenotypic information. Although there is a wide selection of software tools available for extracting phenotype information from images [28, 145], the design and implementation of specific phenotyping pipelines is often required for individual studies due to inconsistencies between datasets. This is true of both traditional image analysis where thresholds and parameters need to be adjusted, as well as more recent machine learning techniques which require the time-consuming manual annotation of training data. In addition, some phenotypes are difficult to measure from images, and ad-hoc solutions tailored to a particular imaging modality or dataset are often required in place of more general ones.

To overcome the many challenges associated with image-based phenotyping, we propose Latent Space Phenotyping (LSP), a novel image analysis technique for automatically quantifying response to a treatment from sequences of images in a treatment study. LSP is related to a broad family of techniques known as latent variable models. These models have been previously used for modelling variation in image data via variational inference, using variational autoencoders (VAEs) [56]. LSP instead constructs a latent representation that best discriminates between image sequences of control and treated samples of a plant population and then measures differences among individuals within the latent space to quantify the temporal progression of the effect of the treatment. The key characteristic of LSP in comparison to existing image-based phenotyping methods is that the phenotype estimated using an image analysis pipeline is replaced with an abstract learned concept of the response-to-treatment, inferred automatically from the image data using deep learning techniques. In this way, any visually consistent response can be detected and differentiated, whether that response is a difference in size, shape, colour, or morphology. By abstracting the visual response to the treatment, LSP is able to detect and quantify complex morphological changes and combined changes of multiple phenotypes which would not only be extremely difficult to quantify using an image processing pipeline, but may not even be apparent to a researcher as correlating with the treatment. In this study, we use a combination of natural and simulated datasets to demonstrate that LSP is effective across different plant species (Setaria, sorghum, Brassica napus L., simulated Arabidopsis thaliana) and different types of treatment studies (drought stress, nitrogen deficiency, simulated changes in leaf elevation, and simulated changes in growth rate).

5.2 Related Work

Section 2.1 describes previous work on plant stress phenotyping in the literature. There are many different techniques for describing biotic or abiotic plant stress, demonstrated in a variety of different plant species. However, all of the studies reviewed in Section 2.1 are specific to a particular application, making them more like case studies for detecting particular symptoms or pathologies than general-purpose stress quantification
systems. Principal component analysis in images for plant phenotyping has previously been used by Turner et al. for describing variance in carrot roots [133]. However, the proposed method is significantly different than simple dimensionality reduction in images using techniques such as PCA or autoencoders, as described in Section 6.4.

Although automatic plant stress phenotyping has not yet been described in the literature, there is some existing work on general shape description. Persistent homology (PH) is a morphometric technique which has recently been applied to plant phenotyping tasks [68]. Persistent homology is named for the topographical concept of homology groups, which is concerned with entities such as path-connected components ($H_0$ homologies) and 2D holes ($H_1$ homologies). In [68], the authors demonstrate an application to leaf shape and an application to root shape in tomato (Solanum pennelli). In the leaf shape application, PH is applied in a three-step process. First, the point density of the object contour (after segmentation and binarization) is estimated using a Gaussian density estimator. Next, this density object is multiplied by a kernel (a ring...
structure) which highlights density at a particular scale (i.e. the radius of the ring), producing a new density map. Lastly, a “homology barcode” is generated by examining a particular “slice” of the density map in the vertical axis, as new connected components first arise and then merge together while the slice descends to zero, resulting in a single connected component at the last step. Root complexity is estimated in a similar way, by dilating the root pixels and then tracking the number of loops and holes as the object is dilated [68].

Persistent homology shares some characteristics with the proposed method. Both methods are able to incorporate many different characteristics into a single descriptor. Both are robust to noise, and are exploratory methods which do not require a priori direction on which traits are to be quantified. However, for the stated goal of automatic plant stress phenotyping, PH is of limited utility. It is only capable of shape description, and will not analyze colour or intensity characteristics. It is a discriminative paradigm which does not consider temporal factors such as stress propagation. It is not specific to detecting differences induced by multiple conditions. Finally, it is not an automated system, requiring various pre-processing steps and a unique and potentially complicated implementation on a per-dataset basis (such as for the leaf and root examples given in the paper).

The temporal aspect of plant stress makes plant stress phenotyping a sequential task. Some work has been published on the topic of latent space models for sequential data. Perhaps the most well-known is the variational encoding for full sentences in English proposed by Bowman et al. [12]. In this application, the authors build a natural language model which uses an RNN and a variational autoencoder to build a latent space representing not individual words, but full sentences. The authors show that the global latent space of sequences allows the model to overcome the limitations of models which perform inference one word at a time on a missing word imputation task. However, this finding in natural language processing is not relevant to the current application as the sequence of images defines a trajectory through a semantic space, not a global context.

5.3 Methods

In order to realize the goal of automatic stress phenotyping, I propose a latent variable model called Latent Space Phenotyping (LSP). This proposed method consists of a three-stage process:

- Embedding samples from the population into a latent space using a supervised classification model.
- Decoding these latent samples back into image space, recovering the relationship between movement in the latent space and visual changes.
- Tracing geodesic paths between timepoints in the latent space, observing and quantifying changes in the image space.

The geodesic path length can be used as trait values with any existing genome-wide association software tools, or interpreted as objective response ratings to inform breeding decisions. A diagram showing the
complete process is shown in Figure 5.1. A complete implementation of LSP, called LSP-Lab, is provided at https://github.com/p2irc/lspLab.

5.3.1 Data

Performing an LSP analysis requires an image dataset, comprised of images taken at an arbitrary number \((U)\) of time points during cultivation for each individual in each of the treatment and control conditions. There should be no missing time points, otherwise the entire sequence cannot be included. Although sequences of differing length within the same experiment could be used in principle, this has not yet been observed and so we do not support this in our implementation to avoid unspecified behavior. The initial timepoint should ideally be zero days after stress (DAS), in order to establish this as the baseline for determination of the effect of the treatment. Controlled imaging (using imaging booths, stages, or incubators) is recommended in order to maintain consistency in image characteristics such as distance from the camera and the position of the specimen in the frame. However, the method is robust to noise in the images (such as variations in lighting) as long as the noise is consistent between both treatment and control samples, not specific to one condition.

5.3.2 Embedding Network

In order to measure a plant’s response to treatment, it is first necessary to determine which visual characteristics in the images indicate the presence of this effect. To learn the visual features correlating with treatment, LSP utilizes a learned projection of images from a population into an \(n\)-dimensional latent space, a process known as embedding. The embedding is shaped by a supervised learning task, which trains a convolutional neural network (CNN) to extract visual features relevant to the discrimination of treatment and control samples. Performing this embedding allows the method to learn the latent structure of the response, and gives the method the ability to overlook any morphological or temporal characteristics that may be different between accessions, but do not correspond to response to the treatment.

The process of training the embedding network requires only treatment/control labels for each sample. The input to the training process is a sequence of images taken for each individual in the treatment and control conditions. The genotypes are divided into training and validation sets with a random 80-20 split. Images are standardized by subtracting the mean pixel value and dividing by the standard deviation, and then used as input to a CNN. For each time point image, the activations of the last fully connected layer in this CNN are used as input to a Long-Short Term Memory (LSTM) network (Figure 5.2). The CNN and LSTM are trained simultaneously in an end-to-end fashion.

We describe both CNNs and LSTMs briefly here, but refer to the literature for more detailed summaries of deep learning in general and these network variants in particular [63, 129, 136]. A CNN can be used to learn local feature extractors from image data. The capability of a CNN to learn a complex representation of the data in this way allows the technique to perform well in many complicated image analysis tasks, such as
Figure 5.2: The deep network used in learning an embedding. A CNN takes a sequence of images at various timepoints and feeds outputs to an LSTM, which in turn is used to predict the treatment. The LSTM is removed and the CNN is retained to embed new samples.

image classification, object detection, semantic and instance segmentation, and many other application areas [63]. CNNs have been used extensively in the recent literature on image-based plant phenotyping, showing promise in several areas, including disease detection and organ counting [52, 75, 83, 96, 136]. For the process of learning an embedding, we implement a simple four-layer convolutional neural network as described in Table A.6. Larger architectures were tested and found to show no difference in the experiments reported in this study.

Recurrent neural networks (RNNs) are an extension to neural networks which allows for the use of sequential data. RNNs are a popular tool for time series, video, and natural language problems, for which sequence is an important factor. Briefly, RNNs maintain an internal state which is updated through the sequence, allowing them to incorporate information about the past into the current time point. LSTMs are an exten-
sion to RNNs which incorporate a more complicated internal state which is capable of selectively retaining information about the past. LSTMs have also appeared in the plant phenotyping literature, demonstrating that they are able to successfully learn a model of temporal growth dynamics in an accession classification task [129]. LSTMs have also been used as a model of spatial attention in the segmentation of individual leaves from images of rosette plants [113].

The final time point of the LSTM feeds into a two-layer feed-forward neural network, the output of which uses the treatment/control labels of the training images as classification targets, using a standard sigmoid cross-entropy loss for training. The loss on the validation set is monitored during training to detect whether or not the embedding network has learned a general concept of the response to the treatment, as opposed to simply overfitting the training data. For the purposes of our application, we prefer embeddings which create only the minimum variance in the latent space necessary for performing the supervised classification task. That is, we prefer embeddings for which variance in most dimensions is close to zero. This helps the subsequent phase of training (Section 5.3.3) to recover differences in the images which correspond to a generalized concept of response-to-treatment, instead of learning features which are specific to one sample or to a group of samples. To incentivize this, we include an additional loss term for the embedding process alongside the cross-entropy loss and $L_2$ regularization loss, called the variance loss ($\mathcal{L}_v$),

$$C = \frac{E^T E}{mU}$$  
$$\mathcal{L}_v = \det(C)$$  

where $E$ is the mean-centered matrix of embeddings for a batch of $m$ sequences of $U$ images. The result is sometimes called the generalized variance [142]. In addition, we add a small constant $\lambda_v$ to the diagonal of $C$. This is for two reasons — first, it prevents the case where zero variance in a dimension causes $C$ to be non-invertible, stopping training. Secondly, it stops the optimization from shrinking the variance in one dimension to an infinitesimally small value, effectively pushing the determinant to zero regardless of the variance in the other dimensions and allowing the optimization to ignore the $\mathcal{L}_v$ term altogether. Ordinarily, we would find it necessary to restrict the multi-dimensional variance in the latent space by constraining the size of the latent space $n$ to the minimum size necessary for convergence. We find that using the variance loss term allows us to use a standard latent space size of $n = 16$ for all experiments, and individual datasets will utilize as few of these available degrees of freedom as necessary as dictated by this term in the loss function. A value of $\lambda_v = 0.2$ was used in all experiments. Further tuning of $n$ and $\lambda_v$ did not appear to have an affect on the results and so it can be concluded that the method is not highly sensitive to the choice of these hyperparameters, up until extremely high $\lambda_v$ values around 1.0 start to prevent convergence of the embedding network. The Adam optimization method [55] is used for training with an initial learning rate of 1e-3.

After the network has finished training, the images of the training and testing sets are then projected into latent space. This embedding is given by the activations of the final fully connected layer in the CNN. In this
way, each of the images in each of the treatment and control sequences can be encoded as $n$-dimensional points in the latent space. The final result of the embedding step is all images projected into the same $n$-dimensional space, which can be visualized using a dimensionality reduction technique (e.g., PCA). The embedding plot is used only for visualization purposes, since distances on the embedding plot do not correspond to semantic distance between samples, an issue discussed in Section 5.3.4. Creating an embedding plot with exact distances between accessions would require calculating on the order of $(Um)^2$ pair-wise paths in the latent space. This is intractable given the computational expense involved for determining each path length, a process described in Section 5.3.4. However, generating the embedding plot using Euclidean distances between embeddings often illustrates stratification of samples in the latent space, albeit with approximate accuracy.

### 5.3.3 Decoding Network

The second phase of the method involves training a decoder which performs the same function as the embedding process described in Section 5.3.2, but in reverse. The purpose of the decoder is to define the mapping from latent vectors to image space, discovering the latent structure in the image space, and allowing us to calculate paths in the latent space during the subsequent phase (Section 5.3.4). The structure of the decoder network consists of a series of convolutional layers followed by transposed convolution layers, which increase the spatial resolution of the input (Table A.7). This architecture is similar to those used in other generative tasks, with the exception that there is no linear layer before the first convolutional layer, to prevent the decoder from overfitting. Samples in the training set are projected by the finalized embedding CNN into the latent space, and then the decoder projects these latent space vectors back into the input space (Figure 5.3). A reconstruction loss function quantifies the difference between the original image and its reconstruction provided by the decoder in terms of mean squared error (MSE). Compared to training the embedding network, a lower learning rate of 1e-4 is used for training the decoder. The Adam optimization method is used in both cases [55].

Since the embeddings are derived from the supervised classification task, the only features which are encoded in the latent representation are those which are correlated with the response to the treatment. For example, in Figure 5.3 (middle) the induced angle of the synthetic rosette (the plant leans slightly to the left) is not reflected in the decoder’s prediction, since plant angle is not encoded in the latent space due to it not being correlated with the simulated response-to-treatment. The leaf elevation angle, however, does match between the real and the predicted images. More examples of encoded and decoded images are shown in Figure A.1. In practice, the decoder’s output for an input with support in the latent space will tend towards the mean of all images which embed to a point near this location. This mean image should be free of the specific characteristics of any particular accession or individual. The use of MSE creates decoded images which appear blurry — this is an expected result, and helps produce smooth interpolations in the image space when calculating paths in the latent space as described in Section 5.3.4.
Figure 5.3: Real images from the Setaria, synthetic Arabidopsis, and sorghum datasets (left), and the same images predicted from their latent space encodings by a decoder network (right).

5.3.4 Measuring Response-to-Treatment using the Latent Space

In the third and final part of the process, we seek to quantify the change in the decoder’s output as we travel in the latent space over time, with respect to the embeddings of the images at each time point for a given individual. In other words, we are interested in characterizing the semantic distance between decoded images at the initial and final timepoints - that is, the distance between these images in terms of stress propagation. This characterization of semantic distance needs to be considered in terms of the geodesic path on the latent space manifold, rather than Euclidean distance in the latent space or in the image space. Figure 5.4 illustrates the difference between the Euclidean distance and the geodesic distance in a hypothetical latent space for a toy example.

In Section 5.3.3 we defined a decoder (or a generator function) $g : \mathcal{Z} \rightarrow \mathcal{X}$ where $\mathcal{Z}$ is the latent space and $\mathcal{X}$ is the input space of the CNN (Figure 5.3). Since $g$ is trivially non-linear, this implies that $\mathcal{Z}$ is not a Euclidean space, but a pseudo-Riemann manifold [4]. A pseudo-Riemann manifold is a manifold which is differentiable everywhere, and equipped with a metric tensor which allows calculations analogous to the dot-
Figure 5.4: Distance between embeddings of images of lines at different angles in a hypothetical latent space. Here, the \textit{semantic distance} is the difference in the interior angle. The Euclidean distance between images in the image space is constant, and the Euclidean distance (dotted arrow) between their encodings in the latent space is evidently not representative of the semantic distance. However, the geodesic path (solid arrow) between images represents the semantic distance well.

product of vectors in Euclidean space. The geodesic distance of a path $\gamma$ on a latent space pseudo-Riemann manifold mapped through a generator $g$ in the continuous case is given by

$$\text{Length}(\gamma_t) = \int_0^1 \| J_{\gamma_t} \frac{d\gamma_t}{dt} \| \, dt,$$

mapping the path through $g$ via the Jacobian $J_{\gamma_t}$ \cite{Fadili2003}. Minimizing this path in the discrete case can be accomplished by optimizing the squared pair-wise distance between a series of intermediate path vertices, minimizing

$$\arg\min_{\gamma} \sum_{i=1}^{j} h(g(z_i), g(z_{i-1}))^2$$

where $h$ is a difference function \cite{Luo2017}. Performing this optimization on the latent spaces generated by LSP is possible using a standard choice for $h$ such as $L_2$ distance in the image space, since this distance in the image space of the decoder is what we seek to optimize when determining paths through the latent space.

Using the embeddings of the images for the initial and final timepoints provides the start and end points for a path through the latent space. The embeddings of the intermediate timepoints are also computed, and these are used as stationary vertices on the path. Since more vertices means a more accurate discrete approximation of the geodesic path, we interpolate additional intermediate vertices between the stationary vertices. These vertices are calculated by optimizing Equation 5.3. For all experiments we use as close to, but not more than, 30 vertices for the path, with an equal number of intermediate vertices between each pair of stationary vertices. In general, the choice of the number of vertices is limited by GPU memory. Instead of performing progressive subdivision as in \cite{Luo2017}, we start from a linear interpolation between stationary...
vertices. This allows us to perform the optimization all at once, instead of dividing the task into multiple successive optimizations which is potentially more expensive. Calculating the total path length as in the sum in Equation 5.3 describes the individual in a single unitless scalar value, indicating the difference in semantic distance travelled over the course of the treatment.

Intuitively, the process can be thought of as tracing a path through latent space from where the initial timepoint embeds to where the final time point embeds. In order to find this path, the current position in latent space is decoded into image space by the decoder. Then, the position is moved in the direction which creates the smallest change in this decoded image. As the path is traced in this way, watching the output of the decoder reveals a smooth “animation” where the number of animation frames corresponds to the number of path vertices. The trait value corresponds to how much change there is between each frame and the next, summed up over the entire path. It is important to note that the distance travelled in latent space is irrelevant – the measurement occurs in the output space of the decoder.

5.4 Results

We evaluated the proposed method using three natural datasets across different plant species and different treatment types, including a population of recombinant inbred lines of Foxtail grass (Setaria) treated with drought stress [29], a panel of sorghum treated with nitrogen deficiency [138], and the founders of a nested association mapping population of Canola Brassica napus treated with drought stress. We performed two additional experiments using synthetic datasets, including a model of Arabidopsis thaliana, where ground-truth candidate loci were verified.

5.4.1 Setaria RIL (S. italica x S. viridis)

We used a published dataset of a recombinant inbred line (RIL) population of the C₄ model grass Setaria [30, 29]. The dataset includes drought and well-watered conditions and has been used to detect QTL relevant to water use efficiency and drought tolerance [108, 29].

The dataset was used as provided by the authors of the original study [29] with a few modifications. The image data was downsampled to 411 by 490 pixels, to allow for a more practical input size for the CNN. Since the camera varies levels of optical zoom over the course of the trial, it is also necessary to reverse the optical zoom by cropping and resizing images to a consistent pixel distance. In order to minimize the effect of perspective shift, the plants were cropped from the top of the pot to the top of the imaging booth, between the left and right scaffolding pieces. This effectively removes the background objects and isolates the plant on a white background. Removing the background is not necessary in the general case – that is, if the background does not change over time. However, since the optical zoom creates differences in background objects, it is practical to remove the background to remove this potential source of noise. The February 1st time point was selected as the initial time point, since many of the earlier time points were taken before
emergence (before the plants broke the soil). In total, 1,138 individuals representing 189 genotypes and six
time points were used. The SNP calls were used as provided by the authors, resulting in a collection
of 1,595 SNPs for this experiment (available at NCBI: SRX2899280). The latent distance values generated by
the proposed method were used as trait values for the multiple QTL biparental linkage mapping pipeline
provided by the authors of the dataset, in order to replicate the methodology used in the published results.

A histogram of latent distance values for individuals in each of the water-limited and well-watered con-
ditions is shown in Figure 5.6 (left). A total of four QTL were detected with respect to the ratio of the
trait under the two conditions. However, we discard these QTL as potentially spurious under the guidance of
the original paper, which found that most of the QTL found using the ratio were not also recovered using
the difference in trait values [29]. For the difference in trait values between conditions, we identified two
QTL associated with drought resistance in the Setaria RIL population (Figure 5.6, right). These loci are
reported by Feldman et al. as corresponding to plant size and water use efficiency ratio (5@15, within the
95% confidence interval of the reported peak of 5@13.7), and plant size and water use efficiency model fit
(7@34), respectively. Although we were only successful in replicating two of the genotype by environment
QTL from the published study, many of these previously reported QTL correspond to a water use model
incorporating evapotranspiration, not a single trait derived directly from the images such as vegetation area.
The normality criterion for ANOVA is violated, and so we use a non-parametric Kruskal-Wallis test. Running
this test, we observe high significance for the effect of genotype on the trait \( p = 3.77e^{-9} \). The same was
found for the interaction \( p = 2.2e^{-16} \).
5.4.2 Sorghum (S. bicolor)

For this experiment, we used an existing study of nitrogen deficiency in sorghum [138]. The authors of the dataset applied a nitrogen treatment to a panel of 30 different sorghum genotypes. Individuals were placed into the control condition with 100% ammonium and 100% nitrate (100/100), the 50% ammonium and 10% nitrate condition (50/10), or the 10% ammonium and 10% nitrate condition (10/10). Images were analyzed with respect to various shape and color features to detect the presence of a response to the treatment. No association mapping was performed in the published study, and so GWAS results cannot be compared against. Due to the small size of the dataset, we use data augmentation to help prevent overfitting. This involves introducing random horizontal flips, randomly adjusting brightness and contrast, and cropping to a random area of the image during training. Images were downsampled to 245 by 205 pixels.

In the published study, the authors found that a PCA of 17 different shape features was able to distinguish the control condition (100/100), the high-intensity treatment (10/10), and the low-intensity treatment (50/10). A PCA of various hue and intensity features of segmented vegetation pixels was able to distinguish between the 100/100 and 10/10 treatments, but unable to distinguish between the 50/10 and 10/10 conditions. An LSP analysis of the same dataset was able to distinguish between the 100/100 and 10/10 conditions (Figure 5.7), but failed to converge when tasked with differentiating between the 50/10 and 10/10 treatments as it was unable to fit the training data. This implies that differences between the two lower nitrogen conditions were too subtle to be detected by either LSP or the collection of pixel intensity features. The LSP method could be adapted to analyze all three conditions in a single experiment by replacing the sigmoid cross-entropy operation in the encoder with a softmax cross-entropy operation. However, the analysis was split into pairs of conditions because the non-convergence of the 50/10-10/10 pair also prevents convergence of the three-condition experiment.

5.4.3 Canola (Brassica napus)

Next, we performed validation on the founder panel of a nested association mapping population of B. napus. In total, 50 genotypes were used in three replications in each of the treated and control conditions, for a total of 300 individuals. Images were taken daily during the early growth period and subsequently every other day, and were downsampled to 245 by 205 pixels. As with the previous datasets, the plants were imaged in a Lemnatec indoor plant imaging system for a total of 40 days and treated individuals were subjected to a drought treatment. In contrast to the Setaria RIL experiment, the Canola study involved three phases. First, an initial growth phase which lasted 14 days where no treatment was applied. Next, a 20-day drought phase was applied where watering was reduced from 100% to 40% field capacity, while the pots were imaged every two days. Lastly, a 6-day recovery phase took place where individuals were again watered uniformly across conditions. The results of the LSP analysis are shown in Figure 5.9. Because this experiment involved three distinct treatment phases, the analysis was performed on each of the three relevant portions of the latent
space path in series. This gives a separate set of results for each phase, where the performance of individuals can be assessed within each phase. Interpolation was not used on this dataset as it already contains the target number of 30 path vertices.

As with the sorghum trial, the NAM panel of canola is too underpowered to find QTL underlying resistance to drought. However, the trait values output by the proposed method distinguish between the two conditions in the treatment phase. Phenotypic response as measured by geodesic distance was also readily observed when inspecting additional factorial variables, such as flowering time (Figure 3.3, right panel). It has been established that drought affects productivity in canola differently depending on the developmental stage [112, 16, 3, 22], with the onset of flowering time being one of the most sensitive stages. Most of the late flowering varieties only started to flower after the drought treatment was complete and we thus expected to see a difference in their drought response. There were 15 genotypes in the early flowering time category, 18 genotypes in the intermediate flowering time category and 17 genotypes in the late flowering time category. Observations of flowering time for these genotypes were conducted in replicate in controlled growth conditions in 2012 at the Agriculture and Agri-Food Canada research facility. Only one genotype consistently produced outliers, and only during the post-treatment recovery phase. In order to determine whether multiple replications of the same line were clustered together in the output, a one-tailed F test was performed using the within-group variance for each of the 100 genotype-treatment pairs. For the pre-treatment, treatment, and post-treatment stages, we found that 26, 34, and 29 of the 100 groups were significantly grouped in the output, respectively ($p < .05$). We also explored the effect of treatment by running two-way ANOVAs for the interaction of genotype and treatment on the trait value ($p = 6.9 e^{-4}$) as well as flowering time group and treatment on the trait value ($p = 2.29 e^{-8}$).

5.4.4 Synthetic Arabidopsis thaliana Model

Synthetic images of rosettes [135] and roots [74] have been used previously to train models for phenotyping tasks. Here we used synthetically generated image data as it allows us to introduce specific variance in the imagery based on a simulated casual SNP, and then investigate the method’s ability to recover that variance on the other end by running a GWAS on the simulated population. We use FaST-LMM [71] to perform this analysis and generate the Manhattan plots.

For this purpose, we used an existing L-system-based model of an A. thaliana rosette [135], based on observations and measurements of the development of real A. thaliana rosettes [85]. The model was run in the lpf model program [1], which simulated the development of the plant over time, and rendered the resulting images. We selected seven of these images corresponding to different time points of the simulation for the LSP analysis.

To generate the synthetic A. thaliana genetic dataset, we begin from a real A. thaliana genotype database known as the A. thaliana polymorphism database [48]. This dataset includes 214,051 SNPs for 1,307 different accessions of A. thaliana. A single causal SNP was chosen at random, and we let that SNP represent a
polymorphism which confers resistance to a hypothetical treatment that affects the plant’s leaf elevation angle. The elevation angle of the plant’s leaves is sampled from a normal distribution which is parameterized according to whether the sample is untreated, treated-and-resistant, or treated-and-not-resistant. Figure 5.10 shows the effect of the simulated treatment where the angle of the leaves on the treated plant is increased relative to the untreated sample. Other parameters in the model, such as growth rate, are normally sampled for each accession.

It should be noted that, the growth rate of the simulated *A. thaliana* plant is completely uncorrelated from the treatment, as are multiple other model parameters. This means that, although the effect of the treatment is still visually apparent, the embedding network must learn a complex visual concept and cannot rely on measuring the number of plant pixels to discriminate between treated and untreated samples. Since the leaf elevation is modulated as a function of plant maturity, the effect of the treatment is not visible in plants with a low growth rate, adding considerable noise and further increasing the complexity of the task. Also note that performing phenotyping on this image dataset would be challenging, since estimating leaf angle from images is a nontrivial image processing task, especially in the absence of depth information [9, 24].

The method is able to successfully determine the simulated causal locus on chromosome one with no false positives (Figure 5.11). Figure 5.12 shows a comparison between the proposed method and a naive solution where the image distance is calculated between each pair of images, with no embedding or decoding step. Such an approach would be successful on the simple synthetic imagery described in Section 5.4.5, but fails in this more complex case.

### 5.4.5 Synthetic Circles Model

Lastly, we performed an experiment with synthetic imagery intended to show how the LSP method performs under basic conditions, and how its outputs relate to manually-measured phenotypes in this case. For this purpose we use a simple model of the *A. thaliana* rosette which depicts individuals as white circles on black backgrounds, with a hypothetical treatment causing a decreased growth rate of the circle over time in this simple model.

For each of the control and treatment conditions, a sequence of six time points is generated, with images representing a circle growing from an initial diameter (sampled from a normal distribution) to a final diameter. The growth rate of the diameter is drawn from a normal distribution, parameterized according to condition. For untreated samples, the growth rate is sampled from $\mathcal{N}(38.5, 0.5)$ and for treated samples it is sampled from $\mathcal{N}(36, 0.45)$. Additionally, the growth rate under the treated condition is influenced (increased by a small amount) by seven hypothetical background SNPs drawn from a Bernoulli distribution, as well as the presence of the minor allele at a randomly chosen locus in the SNP data. For each simulated resistance locus (the hypothetical resistance locus from the genome or drawing a 1 from the Bernoulli distribution) the growth rate is additionally increased by 0.25. The simulated ground truth growth rate values were used to
generate the synthetic imagery by increasing the diameter of the circle at each time point by the amount of the growth rate.

Performing an LSP analysis of this dataset allows us to forego phenotyping and use the synthetic image data directly. The embedding plot representing the learned embedding of the image data as well as the manhattan plot are shown in Figure 5.13. LSP is able to recover the simulated causal locus with no false positives in this simple application. Relating LSP to the established method of using image processing to extract the growth rate phenotype, we examine the correlation between pair-wise distances in the latent space and differences in measured phenotype between the same accessions. There is significant correlation between calculated geodesic distances in the latent space and the relative growth in the number of white pixels in the synthetic circles dataset ($R = 0.93, p < 0.01$).
Figure 5.6: Results for the Setaria RIL experiment. Top left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later timepoints. Top right: histogram of output trait values. Bottom: LOD plot showing QTL for comparison between water-limited and well-watered conditions.
**Figure 5.7:** Embedding plot (left) and histogram of generated latent distance values (right) for the sorghum nitrogen treatment dataset for the treated (10/10) and control (100/100) conditions.

**Figure 5.8:** Well-watered (left) and water-limited (right) examples of a particular line from the *B. napus* L. NAM population
Figure 5.9: Result for the B. napus experiment. Left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later timepoints. Right: analysis of the output for the three experimental phases, categorized by flowering time.

Figure 5.10: Untreated (left) and treated non-resistant (right) synthetic Arabidopsis plants at the final timepoint, showing differences in leaf elevation angle.
Figure 5.11: Results for LSP on the synthetic A. thaliana model. Top left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later time-points. Top right: histogram of output trait values. Bottom: manhattan plot showing the simulated causal locus on chromosome 1. The Bonferroni-corrected $p < 0.01$ significance threshold is shown as a dashed line.
Figure 5.12: Ablation experiment using Euclidean image distance between each pair of images in the sequence for the synthetic Arabidopsis dataset. The naive solution fails to recover the simulated resistance QTL on chromosome 1.

Figure 5.13: Result of LSP on the synthetic circles dataset. Top left: embedding plot. Treated samples are shown in red and control samples are shown in blue. Darker points indicate later timepoints. Top right: histogram of output trait values. Bottom left: manhattan plot. Bottom right: ground truth growth rate data used to generate the image dataset.
6 Focused Analysis of LSP

This Chapter provides discussion around issues arising from the Latent Space Phenotyping technique, its limitations, and additional experiments designed to articulate various aspects of the method.

6.1 Limitations

The latent space phenotyping method as described has some limitations, including increased computational requirements compared to the majority of image-based phenotyping techniques. Since the method involves multiple deep neural networks, the use of GPUs is required to perform these optimizations in a tractable amount of time. The experiments presented here were performed on two Nvidia Titan V GPUs and the time required per experiment ranged from two to eight hours depending on the number of accessions and the number of timepoints in the dataset.

Beyond computational requirements, another limitation of the method is a substantial difference in interpretability compared to GWAS using standard image-based phenotyping techniques. The traits measured with these standard techniques often have a direct and interpretable relationship with the response to the treatment – for example, it has been shown that the number of plant pixels in an image can be used as a proxy for biomass [64]. Therefore, the measured phenotype can be directly interpreted as the biomass of the sample and QTL can be found which correlate with the effect of the treatment on biomass. In the case of LSP, the individual’s response to the treatment is abstracted and quantified only relative to other individuals in the dataset. Interpretability techniques such as saliency maps [125] (Figure A.2) can help to elucidate relevant regions in the images, but the measurements still lack a direct biological interpretation in the same way as measurements of biomass. Therefore, candidate loci obtained through LSP must be interpreted differently, and biological explanations must be inferred from the function of the detected loci.

Another limitation of the LSP method is that it is non-deterministic due to randomized initial weights and random mini-batching (as with all deep learning methods) and therefore repeating the same experiment may output different results. This may be because different attempts to train both the encoder and decoder settle in different local minima, or converge at different speeds due to traversing different routes in the loss surface. As demonstrated in Section 6.2, these differences in convergence characteristics could be exacerbated by different architectures. They could also be caused by different hyperparameters such as the learning rate, or regularization strength. When comparing two experiments which vary slightly in their outputs, there is no sense that one is more “correct” than the other, since the phenotype is abstract and not directly interpretable.
Although there is no guarantee that the trait values reported by the method will be consistent between runs, we found the reported QTL to be consistent across runs for both synthetic datasets. However, a repeat of the *Setaria* RIL experiment resulted in a similar histogram and a between-condition p-value on the same order as the results reported in Figure 5.6, but both previously detected QTL fell below the significance threshold. This is an inevitable consequence of using a non-deterministic method. However, it should be noted that deterministic methods are not inherently repeatable either – different thresholds, outlier detection methods, and data transformations affect the detected loci in these cases.

### 6.2 Effect of Different Encoder Architectures

Because it is possible that the embedding is influenced by the architecture of the encoder network used to embed the sequences, we investigate the effect of different encoders on the synthetic datasets presented in this document. In total, three different encoder architectures were investigated. Each architecture was used as a feature extractor, followed by a fully connected layer with 64 units and an embedding layer with *n* units (*n* = 16 in all experiments presented in this dissertation). Global Average Pooling [44] was also tested in the encoder, but caused non-convergence on the sorghum and *B. napus* datasets and so was not used. The same decoder architecture is used throughout (as described in Table A.7). The same variance loss is used as described in Section 5.3.2 (this loss term is also explored separately in Section 6.6).

**4-layer:** The network proposed in Table A.6, this architecture is a simple convolutional network consisting of four blocks. Each of the blocks contains a convolutional layer, a max pooling layer, and a batch normalization layer. The depth output of the convolutional layers increases from 16 to 32 after the first block. A more typical pre-activation batch normalization implementation was also tested, but found to have slightly poorer convergence qualities.

**resnet-7:** A standard choice in modern vision tasks, residual networks (Resnets) use residual connections which allow the gradient signal to skip blocks of layers [44]. This allows the networks to be extremely deep, with the error signal propagating through the layers of the network more easily on the backwards pass. The original Resnet paper proposes architectures with between 18 and 152 layers – however, we choose a more appropriately sized 7-layer Resnet for testing. This network is identical to resnet-18, but with only one block before downsampling instead of two.

**vgg-16:** One of the most popular feature extractors in computer vision applications, vgg-16 is a computationally expensive architecture [121]. The full feature extractor from vgg-16 is used, with the addition of batch normalization used following each pooling layer (not following pre-activations due to memory constraints).

Figure 6.1 shows the effect of different encoder architectures on the encoder and decoder loss for the synthetic circles and synthetic Arabidopsis datasets. The three architectures represent three very different model capacities, each with different convergence properties during the encoder phase. On the simplistic synthetic circles dataset, the two higher capacity models show that they are able to better distinguish between
Figure 6.1: The effect of encoder architecture on the encoder and decoder loss for the synthetic datasets. Only the moving average is shown on the encoder loss figures for readability.
samples with different diameters, evident from the lower reconstruction loss from the embeddings given by those architectures. For the more complex synthetic Arabidopsis dataset, the different architectures again show different convergence characteristics, but the resulting embeddings all result in similar reconstruction loss. In both synthetic datasets, with all three encoder architectures, the simulated causal locus is recovered, showing that the technique works despite differences in model capacity.

6.3 Analyzing Trait Values Through Time

While most experiments presented in Section 5.4 use the full duration of the experiment in the analysis, in some circumstances it may be useful to analyze particular regions of time. This type of analysis is of particular interest for studies which involve multiple stages, as is the case for the Canola NAM validation study presented in Section 5.4.3 which includes pre-treatment, treatment, and post-treatment stages. In cases such as these, the intermediate path length between arbitrarily selected timepoints can be analyzed. In this way, the embedding can be completed on the full dataset as usual, but the results can be provided on specific time horizons, including highly granular ones such as between each time step or even between interpolated time steps.

![Histograms of trait values for the synthetic Arabidopsis dataset separated into the five periods between each of the six timepoints.](image)

**Figure 6.2:** Histograms of trait values for the synthetic Arabidopsis dataset separated into the five periods between each of the six timepoints.

For another demonstration of an analysis done at multiple time points, Figure 6.2 shows the analysis of the synthetic Arabidopsis dataset described in Section 5.4.4, broken into the results for the area between each of the six timepoints in the dataset. If the method is accurately measuring the response to the treatment, then the trait values intuitively correspond to change in leaf elevation angle between each of these five periods of time. Interestingly, between the first and second timepoints, the control samples exhibit more change than the treated timepoints. This is likely an artifact of how the method responds to, and decodes, juvenile
individuals before differences in leaf elevation angle can be assessed. This is because leaf elevation angle and maturity are encoded into the same continuous space. Following this initial growth phase, a long tail can be seen in the trait values of the treated individuals as leaf elevation angle continues to increase over the course of the trial. This effect appears to fall off towards the final timepoint as the samples approach the end of their growth cycle.

6.4 Comparison to Classical Dimensionality Reduction

Performing the embedding step can be seen as a type of dimensionality reduction, from the high-dimensional image space to a lower-dimensional embedding space. Doing dimensionality reduction in images has been performed before using techniques such as principal components analysis (PCA) or autoencoders. By performing dimensionality reduction on images, it is possible to recover factors which correspond to pixel variance in the images. For example, the Eigenfaces technique uses PCA on small, greyscale images of faces to learn a series of principle components (PCs) [132]. A new image of a face can be encoded as a linear combination of these PCs and this representation can be used to compare the new face against a database of examples to determine similarity. Since then, the use of PCA in images has been deprecated in favour of autoencoders. Autoencoders are better suited to pixel data, since they are capable of capturing non-linear relationships in the data, while PCA assumes a linear relationship between variables. This has been shown to provide better embeddings of image data.

While methods such as Eigenfaces provide a feature vector describing the most major points of variance in the image space, LSP specifically avoids this approach. This is because the most major variations in the images are likely to be from sources completely unrelated to the treatment. For example, the emergence of a new organ creates significant variance in the images, even if the emergence of that organ is not due to the treatment. Using methods such as PCA or autoencoders results in the form of an arbitrary number of features, some or none of which may be useful to the description of the effect of the treatment. Attempting to embed images using other manifold learning techniques such as Multi-Dimensional Scaling (MDS) or Locally Linear Embedding (LLE) suffers from a similar problem – they will attempt to preserve likely meaningless pair-wise relationships in the pixel space.

Let us imagine that one is able to accept the above shortcomings of dimensionality reduction methods such as PCA and autoencoders. Performing the analysis on the full dataset is likely to mostly identify features related to maturity, since this is often the largest cause of variance in the images (and using the full dataset with techniques such as PCA which do not use mini-batching is likely intractable due to memory restrictions). To circumvent this, one could imagine taking the high-dimensional features provided by such methods in separate analyses at each time point. Although most of these features are likely irrelevant, one could use non-parametric significance testing to determine which of the features are correlated with the presence of the treatment. This was validated on both synthetic datasets as well as the Canola dataset using
a Mann-Whitney U test. Various PCs were shown to contain information relevant to the condition, and even appeared as describing the most variance (PC1 and PC2) in the treatment phase of the Canola trial.

A more subtle problem with a simplistic latent model and lack of a temporal component is that there cannot be a measurement of the progress of a single, continuous, non-linear process through time, especially if that process contains multiple different stages (such as wilting, followed by senescence). Both PCA and autoencoders are able to encode images and provide reconstructions, making it possible to determine difference in the pixel space given two encoded samples. However, differences in the pixel space can only be calculated between two individuals at discrete time points, and the evolution of these differences across time points cannot be assessed. LSP, on the other hand, integrates stress and maturity in a single continuous space which can be smoothly interpolated through. This space can represent complex, continuous, non-linear changes in different regions. Although PCA was able to detect the presence of the treatment in both the synthetic datasets as well as the Canola experiment, the loadings on these significant PCs predictably failed to discriminate between the treatment and control conditions across time.

6.5 Effect of Temporal Sampling Rate and Interpolation

The proposed method requires a sequence of images through time. The number of these time points ($U$), however, is not prescribed. Therefore, it is of interest to examine how the results may vary with the choice of $U$. In addition to the supplied time points, the continuous manifold learned by LSP also allows for an arbitrary number of intermediate timepoints to be interpolated on the path between real time points encoded from images. Of interest for this experiment are two separate questions. Firstly, does a dataset with a very low temporal sampling rate create a manifold in the latent space with insufficient support? A lack of support in this case means that the data in the latent space is too sparse to train a reliable decoder which generates coherent outputs through the entire path between time points. If the embedding step results in discrete clusters of embeddings with largely empty space between them, then interpolation through these empty regions will likely result in unpredictable behaviour, as the intermediate points are searching through a space in which the decoder has no support. Additionally, does a low sampling rate result in an inability to recover QTL? Also of interest is the effect of interpolation. We investigate how increasing the number of path vertices via interpolation affects the trait values output by the method.

In order to evaluate performance with lower temporal resolution, we begin with the synthetic $A.~thaliana$ dataset described in Section 5.4.4. We choose a synthetic dataset for this experiment, because it allows us to evaluate our ability to recover the ground-truth simulated resistance locus with this dataset. Ordinarily this dataset includes images at six distinct time points. In order to downsample this series to three time points, we use only the odd-numbered samples (time points 1, 3, and 5). First, an analysis was performed on the downsampled dataset. This analysis initially appears to be successful. The embedding is performed successfully, resulting in a manifold of the same general shape as the manifold resulting from the full dataset.
with six time points. Although the manifold is more sparse, the surface sampled through the decoder appears to be continuous. This seems to create sufficient support in the latent space for interpolation, and examining the decoded interpolated path vertices shows them to be similar in appearance to those generated from interpolations from the full dataset. However, despite the apparent success of the method with the downsampled synthetic *A. thaliana* dataset, performing the same GWAS analysis fails to recover the simulated causal locus. This result may be surprising, given the similarities in the manifolds and the success in generating interpolations through the latent space. However, inspecting the interpolations shows a high level of change in irrelevant image factors such as the number of leaves compared to development in the leaf elevation angle. This is possibly due to a more sparse latent space, causing the decoder to memorize particularities in the growth of individual samples.

The longitudinal analysis, where the path is calculated from the initial to the final time point for each individual, was negatively impacted by temporal downsampling. However, the results of a cross-sectional analysis, where a path was calculated between the final time point for the treated sample and the final time point for the control sample, was not affected in any way. This may be because doing a cross-sectional analysis in this way does not interpolate through time, making it immune to the decoder’s memorization of individual growth characteristics.

Moving to natural data, the *B. napus* dataset includes 31 time points, making it a good candidate for temporal downsampling. The temporal sampling rate was decreased by half in the same manner and the experiment was repeated. However, the downsampled version of the dataset failed the embedding step. The dataset was over-fit by the embedding network, without learning a generalizable concept of the treatment. This could be because, after downsampling, the embedding network found it easier to overfit this smaller dataset rather than using only the small amount of signal present in the treatment phase, which now consisted of only five time points – however, it is difficult to verify whether or not this is the case. Because of the lack of generalization in the embedding step, further analysis was not possible.
From these experiments using data with a decreased temporal sampling rate, we can say that the sampling rate does affect the results. However, it is still unclear how a minimum sampling rate may be determined in these cases. The synthetic *A. thaliana* experiment already had a low temporal sampling rate with only six time points, and downsampling to only three time points represents an extreme case. The *B. napus* case is unique in that only a small proportion of the time points represent a period when the treatment was present, which is a unique study design among all experiments. We may posit that higher temporal resolution is generally better, if only because it increases the total size of the dataset, preventing overfitting during the embedding step.

![Graphs showing trait values for synthetic experiments](image)

**Figure 6.4:** Trait values for the synthetic *A. thaliana* experiment without interpolation (left) and with interpolation (right).

Another point of interest is the effect of interpolation. To illustrate this effect, Figure 6.4 shows the effect of interpolation on the synthetic *A. thaliana* dataset. In this case, interpolating from six timepoints to 26 slightly alters the distributions of trait values, however, the detection of the simulated causal locus is unaffected. A more pronounced effect of interpolation is shown in the case of the synthetic circles dataset. Figure 6.5 shows an example of the circles dataset with a poor embedding caused by a high learning rate. Without interpolation, the trait values for the control group appear to be bimodal. However, after interpolation, the distribution is somewhat smoothed out. It appears that, not only does interpolation create a more precise approximation of path length, but it also has the potential to correct problems with the embedding.

### 6.6 Effect of the Variance Loss

A novel loss term is proposed in Section 5.3.2 called the *variance loss* or $\mathcal{L}_v$. Intuitively, we would like to counteract the decoder’s capability to memorize individual samples by compacting the manifold such that different images which are consistent only in their response phenotype lie immediately close together in latent space. Therefore, we act on the covariance matrix of the embeddings for each batch, minimizing the variance within and maximizing the covariance between latent dimensions. This new, more densely packed space
Figure 6.5: Trait values for the synthetic circles dataset corresponding to a pathological encoding, without interpolation (left) and with interpolation (right).

(where variance in most dimensions is close to zero, or is highly correlated with other dimensions) improves the output of the decoder in that it now only manifests differences in the response-to-treatment instead of particularities of individual samples.

Figure 6.6: Top, left to right: embedding plots for the synthetic *A. thaliana* experiment with \( \mathcal{L}_v \) coefficients of 1 (default), 0.1, and 0 (no variance loss). Bottom: corresponding trait values for the same experiments.

Figure 6.6 shows results from the synthetic *A. thaliana* experiment with various strengths of variance loss. With a variance loss coefficient of 1 (the default) and 0.1, the simulated causal locus can be found. However, with the variance loss term removed, the locus is not found.
6.7 Evaluating the Output of the Decoder

The embedding step of the process has a built-in safety check in using a validation set to check whether or not the embedding process has learned a concept of the visual response to treatment. This is done by simply monitoring the validation loss during the training of the embedding network. The training of the decoder, however, lacks a way to verify that its outputs are meaningful. Without this, the decoder could theoretically output empty images for all embeddings. However, the evaluation of decoder outputs is not a trivial problem. The primary challenge is the fact that the decoder is not attempting to generate samples from the input distribution, and so generating a sample with high fidelity or high log-likelihood (if it were tractable to calculate) would indeed be viewed as a failure from the perspective of the decoder’s purpose. Extracting such a 196,608-dimensional sample from a 16-dimensional sub-manifold would also be exceedingly improbable. Other methods of evaluating the outputs of deep generative models such as GANs are specifically tooled for class-dependent metrics, such as the Inception Score (IS). Such a metric is not applicable since the proposed method is not attempting to output class-correct examples of particular object classes. Many metrics are also designed to penalize generators which have a low diversity of outputs. Such a metric would see the low-variance outputs of the LSP decoder as representing a problem such as catastrophic mode collapse. Beyond the issues with existing decoder metrics are the issues specific to the nature of temporal stress propagation. One could attempt to find correlations between the images and the condition label (this is done with PCA in Section 6.4). However, the output of the decoder is a time series. The effect of the treatment may be visually apparent at particular time points, but the effect may be significantly different when viewed through the time dimension. This means that finding correlations between decoded time points and the condition label is likely to provide an incomplete picture of stress propagation.

Given the temporal nature of the data, it is most sensible to refer to the output trait values as a proxy for decoder performance. Because the trait values are calculated in the output space of the decoder, and include the temporal progression, the final output values are the best indicator of the decoder’s behavior. For example, in the previously described case where the decoder fails to converge and simply outputs a constant, the trait values will all be zero because of a lack of change in the temporal dimension. To analyze if the effect of treatment is present, one could simply perform a non-parametric significance test on the output values themselves. Ultimately, these final output values are the most principled proxy for analyzing the decoder.

6.8 On Identifiability

Let us consider one principle question about latent space methods in plant phenotyping: if a plant’s stress resistance properties can be modelled as a latent variable, is it theoretically possible to recover the true distribution over this variable? This is a proposition that LSP intentionally side-steps, making no such claims about the posterior distribution of its encoder and instead describing the individuals via changes in
the pixel space. Here we will briefly discuss the theory behind latent variable models and the necessary conditions for making such claims about the posterior distribution of the latent variables.

As a latent variable model, the LSP technique models the input images \( x \in X \) through a latent distribution \( Z \). In general, the evidence for such a latent variable model is

\[
p_{\theta}(x, z) = p_{\theta}(x|z)p(z)
\]

where \( \theta \) are parameters, \( p_{\theta}(x|z) \) is the likelihood of the data, and \( p(z) \) is the distribution over latent variables. In a variational regime (Section 2.4.5), the parameters are learned by minimizing the lower bound of the (log) evidence (the variational or evidence lower bound, or ELBO), since the exact posterior is intractable. The ELBO results from a quick derivation:

\[
\log(p_{\theta}(x, z)) = \log \left( \int p_{\theta}(x|z)p(z)dz \right)
= \log \left( \mathbb{E}_{Z \sim q} \left[ p_{\theta}(x|z) \frac{p(x)}{q(z)} q(z) \right] \right)
= \log \left( \mathbb{E}_{Z \sim q} \left[ \frac{p_{\theta}(x, z)}{q(z)} \right] \right)
\geq \mathbb{E}_{Z \sim q} \left[ \log \left( \frac{p_{\theta}(x, z)}{q(z)} \right) \right] \quad \text{(Jensen’s Inequality)}
= \mathbb{E}_{Z \sim q} \left[ \log(p_{\theta}(x, z)) \right] - \mathbb{E}_{Z \sim q} \left[ \log(q(z)) \right]
\]

where \( q \) is the variational distribution. As discussed in Section 2.4.5, in a variational autoencoder, the first term in the ELBO is modelled via the reconstruction loss and the second term is modelled via the Kullback-Leibler divergence between \( Z \) and \( q \). However, in LSP, the distribution \( Z \) is modelled via a supervised learning task, instead of via a variational encoding. In particular, this encoding is conditioned on a secondary observation \( u \), the condition (treatment/control) label. This means that the model evidence is given as

\[
p_{\theta}(x, z|u) = p_f(x|z)p_{\theta,\lambda}(z|u)
\]

where \( \theta = (f, T, \lambda) \) defines a generative model where \( T \) are the sufficient statistics of the distribution, \( \lambda \) and \( f \) are arbitrary functions, and \( p_{\theta,\lambda}(z|u) \) is known as a conditionally factorized prior distribution [54]. Interestingly, it has been shown that there are families of generative models based on such a conditionally factorized prior which can be identifiable under certain conditions [54].

Identifiability in this context means that if two parameterizations \( \theta, \theta' \) are equal, then the evidence \( p_{\theta}(x, z), p_{\theta'}(x, z) \) (Equation 6.1) under these two parameterizations is also equal. Identifiability is relevant because, if the model is unidentifiable, then the true joint distribution over observed and latent variables is unknowable [54]. However, if identifiability conditions are met, then the independent “true sources” \( z_{0,\ldots,1}^* \) of the data can be uncovered up to a trivial linear transformation. This finding and language (referring to latent variables as “sources”) is derived from the non-linear ICA literature. [54] links variational autoencoders to
non-linear ICA under a unified framework to articulate the identifiability requirements of a VAE model with a conditionally factorized prior distribution.

Under such a framework, the true sources $\mathbf{z}^*$ are considered to be drawn from univariate exponential distributions, conditioned on $\mathbf{u}$. Then, the observed variables $\mathbf{x}$ are considered to be a result of these hidden sources modulated by some invertible process $\mathbf{x} = f(\mathbf{z}^*) + \epsilon$. Therefore, the sources can be recovered by inverting the process $\mathbf{z}^* \approx f^{-1}(\mathbf{x})$ (under the simplifying assumption of no noise). The parameter $\lambda(\mathbf{u})$ is a function which maps $\mathbf{u}$ to the distribution parameters $\lambda_{i,j}$.

In the context of plant phenotyping, we might conject that a plant is the product of a generative process from latent factors such as genetic and environmental variables. Making such an assertion about the natural world is likely not motivated – however, this sort of generative assumption is the exact mechanism behind the construction of the structural-functional $A$. *thaliana* rosette model described in Chapter 4 and demonstrated in Section 5.4.4. A synthetic plant is the product of a simulation ($f$) and a vector of model parameters ($\mathbf{z}$) which are drawn from Gaussian distributions. However, the identifiability criterion described by [54] require $f$ to be injective, which we are unable to assume about this process of simulation and rendering. With both real and synthetic plants, we are unable to make functional assumptions about their growth as a generative process. With the current state of identifiability research in deep learning, this is as close as we can get to describing the latent variables which arise from the sort of systems described in this dissertation.

### 6.9 On Disentanglement

Much of the recent research on latent variable models is concerned with a concept known as *disentanglement*. Although there is some disagreement as to what constitutes disentanglement, a commonly cited definition is that a disentangled representation is able to recover the major sources of variation in the data, which vary independently from one another in the input distribution [7]. For example, learning a disentangled $\mathbb{R}^3$ representation of faces may result in a representation where variance in the first dimension corresponds to varying face angle, the second to varying skin tone, and the third to varying hair length. Since all of these sources of variance are independent from one another, one is able to interpolate through just one dimension in this space and a paired decoder built on this representation will vary only the corresponding trait in the output image. The issue of disentanglement is of great interest to the representation learning field, as learning such a disentangled representation means that the model has learned each of these discrete concepts which are significant and interpretable to humans.

One of the most successful latent variable models which specifically enforces disentangled representations is the $\beta$-VAE [46]. While VAEs do not generally enforce disentangled representations, this can be attained by adding a coefficient $\beta$ to the second term of the VAE loss function (Equation 2.18):

$$
\mathcal{L}_{\beta\text{VAE}} = \log p(\mathbf{x}|\mathbf{z}) - \beta D_{KL} [p(\mathbf{z}|\mathbf{x})||q(\mathbf{z})].
$$

(6.4)
With $\beta = 1$, the $\beta$-VAE simply reduces to a normal VAE. However, by setting $\beta > 1$, additional weight is placed on the variational term. This induces increased independence since this prior $q(z)$ is a unit gaussian with diagonal covariance, meaning that the dimensions are completely independent. The trade-off for increasing disentanglement with increased values of $\beta$ is that the reconstruction loss suffers due to less weight being placed on the first term in the loss function. In the degenerate case, the posterior can collapse completely to the prior, meaning that $z$ carries no information about $x$.

Disentanglement is an important subject in representation learning, but one may also recognize its relevance to the topic of latent variable models in plant phenotyping. For instance, are there multiple independent factors of variance which one can extract from a population? Latent Space Phenotyping is concerned with elucidating only one such factor, the response to treatment, as a single continuous visual process. It is a plausible assumption that individual visual elements of this response to stress, for example, changes in color such as chlorosis, and in shape, such as wilting, are not conditionally independent, making them difficult or impossible to disentangle. Since stress propagation is a temporal process, this is also true of maturity and stress response. Future research may investigate disentangling multiple independent responses, or other uses for disentanglement in plant phenotyping.

6.10 On the Manifolds Generated by LSP

The latent space emitted by the embedding network is visualized using PCA, resulting in the embedding plots shown in Section 5.4. Given that PCA is used to visualize the space, it may be tempting to perform a PCA sweep of the space to better understand what the principal components in the latent space represent. However, it is important to note that this PCA is simply a visualization of a space which has no meaning relative to the visual appearance of the images it embeds, except as mapped through the decoder. This means that the principal components shown in the embedding plots do not correspond to principal components in the images (linear, orthogonal changes in pixel intensity). In addition, the output of the decoder is only defined on the supports of the manifold. That is to say that picking a point on the manifold and then sweeping left or right in one of the PC axes in the embedding space will almost certainly move off of the manifold (i.e. the principal components in the embedding space do not correspond to the tangent vectors of the generator), into an area where the decoder yields either noise or nothing at all. Therefore, the PCA of the latent space is only useful for (approximate) visualization of the latent space, and performing a PCA sweep is not motivated.

Section 5.3.2 cautions against the over-interpretation of the embedding plots, and mentions that pairwise Euclidean distances in these plots are not directly meaningful. This assertion is based on existing results which have proven this fact about the latent space of deep generative models [4, 61]. It is possible that two embeddings could exist for the same data, with different pair-wise distances between points and an entirely different topological structure, which result in the same outputs from the method (or, equivalently, two iden-
tical manifolds which yield different outputs). While other techniques such as the popular word2vec [79] in the natural language processing domain do make use of (cosine) distances in a latent space, these applications are materially different. The distance measurements between embeddings are not used to quantify characteristics of the embedded data or interpolate between them in a continuous fashion. The entities to be embedded are one-hot vectors representing discrete items in the vocabulary, not a complex high-dimensional pixel space. While various tasks such as clustering could be performed in the latent space of LSP, these tasks theoretically have no meaningful relevance to the visual characteristics of the subjects in the pixel space.
7 Conclusions and Future Work

In this chapter I review the contributions made in this dissertation. Future directions are identified for continuing research from these foundations.

7.1 Conclusions

In this dissertation, I have proposed the concept of abstract plant phenotypes learned from image datasets using a latent variable model, and demonstrated its application to stress phenotyping. This contribution is enabled by further technical contributions around deep learning and plant modelling in plant phenotyping, which facilitate its implementation as well as its validation.

In Chapter 3 I described the first open-source software package for performing deep learning with image-based phenotyping, with the intention of accelerating interdisciplinary research in this domain. Image processing methods for plant phenotyping have been in use for over 20 years, and they continue to play an important role in the field to this day. Many software packages have been released to allow researchers access to image processing tools for plant phenotyping, and they have seen widespread use in the discipline [34]. However, there are some phenotyping tasks which test the limits of what is possible with image processing methods alone. As reviewed in Chapter 3, these tasks include disease and stress quantification, as well as organ detection and localization. As with other visual domains such as image classification, object detection, and image segmentation, deep learning has shown progress in complicated plant phenotyping tasks such as these, where convolutional neural networks and their variants often outperform hand-crafted image processing pipelines for the same task. Deep Plant Phenomics was introduced as the first software platform for deep learning research in plant phenotyping. Three benchmark tasks using a public dataset were demonstrated using this software, showing strong performance using convolutional neural networks. Since it was first introduced, the Deep Plant Phenomics platform has been improved by contributors to expand its capabilities. It now supports semantic segmentation, object detection, and object counting in addition to classification and regression. Deep learning underlies the three-stage process of Latent Space Phenotyping.

I demonstrated in Chapter 4 that structural-functional plant models are capable of mitigating some of the issues with limited intra-dataset variability in plant phenotyping datasets. Since the initial papers on deep learning for plant phenotyping were published in 2016, the topic has exploded in popularity and the number of papers published per year is still gaining momentum. However, these new deep learning techniques come with new challenges. The manual annotation of training data is a tedious and time-consuming process.
In addition, domain shift is an issue when training networks on images with a limited range of variance in imaging conditions and a high degree of visual similarity between samples. In response to these issues, synthetic data provided by an L-system model of the *A. thaliana* rosette was shown to not only improve performance when used to augment real training data, but was also shown to be significantly interoperable with the images of real rosettes in a public dataset when used for leaf counting via global regression. The contributions of this work include the model and several public datasets of images of synthetic *A. thaliana* rosettes. These findings provided the ground work for subsequent studies using synthetic data in the plant domain [137, 60]. This same rosette model is used to simulate a synthetic plant population for use as a validation case study for Latent Space Phenotyping.

In Chapter 5 I described Latent Space Phenotyping, the first step towards the automatic quantification of fully abstract phenotypes from images. Although image-based analysis in plant phenotyping has a long history, the evolution of techniques has been gradual. In their seminal paper about phenomics, Houle et al. describe this status quo regime of “measuring a limited set of phenotypes that seem the most relevant” as “phenotyping as usual” and urge the field to develop new phenotyping techniques to advance progress in the biological sciences [49]. Beginning around 2016, the emergence of deep learning in the plant phenotyping literature represented one of the most radical changes for the discipline. However, the application of deep learning simply provides more powerful tools for image-based phenotyping, not a new paradigm. It requires the time-consuming manual annotation of training data and, like previous methods, presupposes and rigidly defines the response phenotype. As a profoundly different technique, LSP attempts to initiate the break from “phenotyping as usual” and introduces the discipline to the concept of abstract phenotypes derived directly from images. Through validation using both natural and synthetic datasets, the method is shown to quantify various visual responses. The limitations of the method should be acknowledged, including issues with interpretability, computational expense, and consistency due to a non-deterministic pipeline. Despite these issues which we reserve for future work, LSP has been shown to be a powerful tool which has the potential to significantly impact research in plant science and breeding going forward.

### 7.2 Future Work

It should be noted that, although we have endeavoured to present the application of Latent Space Phenotyping on range of datasets in this work, we have still only scraped the surface of plant phenotyping image data. It remains to be seen how the method responds to other plants with significantly different architectures to those shown here. Of particular interest for future work are plant and root structures with highly branched and articulated forms. Also, although the method is theoretically designed to be robust to the visual differences between genotypes, it is unknown how the method responds if these differences are significantly larger than the differences due to treatment. The testing of LSP in less controlled imaging contexts such as in outdoor field conditions is an important direction for future work. LSP should also be validated in experiments with
larger natural datasets, involving thousands of genotypes. Since the computational expense scales linearly with the number of individuals, such experiments should be feasible.

Future work on the topic of latent space methods in plant phenotyping will likely involve further validation by way of replicating stress tolerance QTL from past experiments. In addition, while the experiments presented in this dissertation only involve one treatment condition, many genotype by environment studies involve multiple treatment conditions representing a range of treatment intensities. The technique may be extended to such studies by replacing the sigmoid cross-entropy operation in the encoder with a softmax cross-entropy operation. Finally, other application areas may be discovered where the three-stage process described here is able to detect and quantify other visual phenomena outside of plant phenotyping.

7.3 Concluding Remarks

This dissertation began with a roadmap of image-based plant phenotyping, from the description of classical image processing techniques to an introduction of deep learning. The strength of deep learning was demonstrated, and structural-functional plant models were shown to mitigate some generalization issues common to plant phenotyping datasets. Ultimately, the concept of an abstract phenotype was proposed, using deep neural networks to quantify the progression of a non-linear response phenotype. The previously described plant model aided in studying the behaviour of this proposed method by simulating a population with particular sources of variance. This work provides foundations for the application of deep learning in plant phenotyping, and offers new directions for quantifying the progression of abstract physiological changes through time.
REFERENCES


Figure A.1: Additional examples of synthetic Arabidopsis rosettes (left) decoded from their latent space vectors (right).
Figure A.2: Example images from the *Setaria*, synthetic Arabidopsis, and synthetic circles datasets (left) and corresponding saliency maps generated using guided backpropagation (right). Intensity is higher for the pixels which have high saliency with respect to the $L_2$ norm of the latent space embedding of the image. The *Setaria* image is from an experiment carried out without cropping to include the background in the saliency demonstration.
<table>
<thead>
<tr>
<th></th>
<th>HTPheno</th>
<th>Rosette Tracker</th>
<th>IAP</th>
<th>PlantCV</th>
<th>Image Harvest</th>
<th>LemnaGrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto. thresholding</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Background subtraction</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Supervised learning</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

**Table A.1:** Segmentation methods provided by popular image-based phenotyping libraries and platforms.

<table>
<thead>
<tr>
<th></th>
<th>HTPheno</th>
<th>Rosette Tracker</th>
<th>IAP</th>
<th>PlantCV</th>
<th>Image Harvest</th>
<th>LemnaGrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area/coverage</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>X/Y extent</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Convex hull</td>
<td>•</td>
<td></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Center of mass</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Compactness/solidity</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Leaf angles</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Bounding ellipse stat.</td>
<td></td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
<td>•</td>
</tr>
<tr>
<td>Stockiness</td>
<td></td>
<td></td>
<td></td>
<td>•</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* Diameter of minimum bounding circle only.

**Table A.2:** Object descriptions provided by popular image-based phenotyping libraries and platforms.
<table>
<thead>
<tr>
<th></th>
<th>HTPheno</th>
<th>Rosette Tracker</th>
<th>IAP</th>
<th>PlantCV</th>
<th>Image Harvest</th>
<th>LemmaGrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locate leaf tips</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Leaf count</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Per-leaf segmentation</td>
<td></td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

**Table A.3:** Advanced segmentation/localization functionality provided by popular image-based phenotyping libraries and platforms.

<table>
<thead>
<tr>
<th></th>
<th>HTPheno</th>
<th>Rosette Tracker</th>
<th>IAP</th>
<th>PlantCV</th>
<th>Image Harvest</th>
<th>LemmaGrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIS (RGB/HSV)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Near Infrared (NIR)</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Fluorescence</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

**Table A.4:** Imaging types supported by popular image-based phenotyping libraries and platforms.
<table>
<thead>
<tr>
<th>Input Size</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>Mutant</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 × 256</td>
<td>128 × 128</td>
<td>256 × 256</td>
<td>128 × 128</td>
<td></td>
<td>128 × 128</td>
</tr>
<tr>
<td>Conv 5 × 5</td>
<td>Conv 5 × 5</td>
<td>Conv 5 × 5</td>
<td>Conv 5 × 5</td>
<td></td>
<td>Conv 3 × 3</td>
</tr>
<tr>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td></td>
<td>Pool 3 × 3</td>
</tr>
<tr>
<td>Conv 5 × 5</td>
<td>Conv 5 × 5</td>
<td>Conv 5 × 5</td>
<td>Conv 5 × 5</td>
<td></td>
<td>Pool 3 × 3</td>
</tr>
<tr>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td></td>
<td>Pool 3 × 3</td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>Conv 3 × 3</td>
<td>Conv 3 × 3</td>
<td>Conv 5 × 5</td>
<td>FC 2048</td>
<td></td>
</tr>
<tr>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Output 1</td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>Conv 3 × 3</td>
<td>Conv 3 × 3</td>
<td>Conv 5 × 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>Conv 3 × 3</td>
<td>Conv 3 × 3</td>
<td>FC 4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>DropOut (0.5)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv 3 × 3</td>
<td>Output 1</td>
<td>Conv 3 × 3</td>
<td>FC 4096</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pool 3 × 3</td>
<td>Pool 3 × 3</td>
<td>DropOut (0.5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC 1024</td>
<td>Output 1</td>
<td>FC 4096</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output 1</td>
<td>Output 5</td>
<td>Softmax</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table A.5:** The network architectures used for each of the phenotyping datasets and tasks. All pooling operations are max pooling with a stride of 2.

<table>
<thead>
<tr>
<th>conv + relu</th>
<th>3 × 3</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>max pool</td>
<td>3 × 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv + relu</td>
<td>3 × 3</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>max pool</td>
<td>3 × 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv + relu</td>
<td>3 × 3</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>max pool</td>
<td>3 × 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv + relu</td>
<td>3 × 3</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>max pool</td>
<td>3 × 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fully connected + relu</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fully connected</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table A.6:** Architecture details for the convolutional neural network used in the embedding. All pooling layers are followed by batch normalization.
<table>
<thead>
<tr>
<th>Layer</th>
<th>Size</th>
<th>Depth</th>
<th>Stride</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>upsample</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$3 \times 3$</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>conv + relu</td>
<td>$1 \times 1$</td>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

*Table A.7:* Architecture details for the decoder network. All pooling layers are followed by batch normalization.